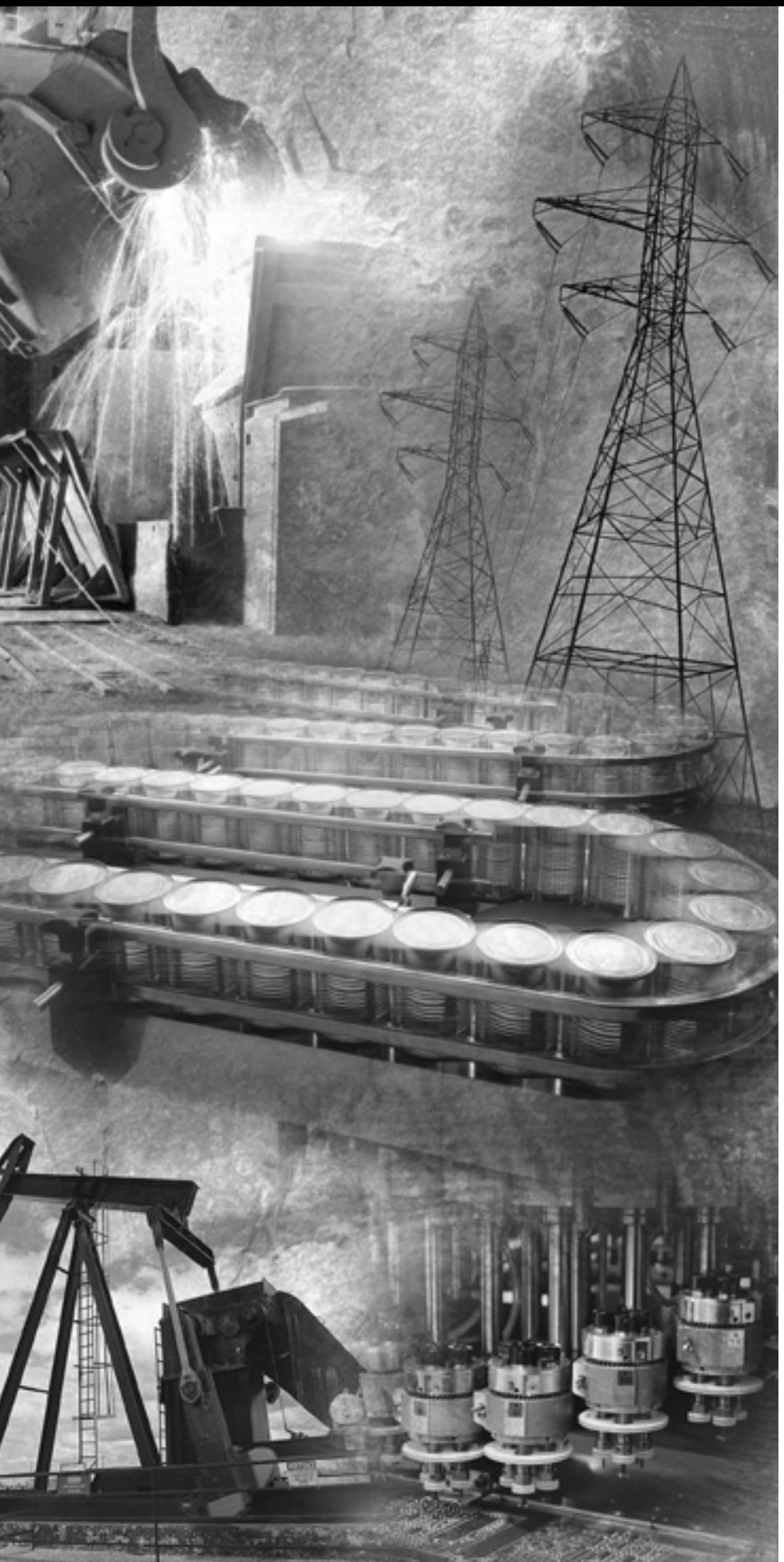


Allen-Bradley

Logix5000™ 控制器基本指令

1756 ControlLogix®,
1769 CompactLogix™,
1789 SoftLogix™,
1794 FlexLogix™,
20D PowerFlex® 700S with DriveLogix

参考手册

Rockwell Automation

重要用户信息

固态设备与机电设备的操作特性不尽相同。 *Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls*（固态控制设备的应用、安装与维护安全指南，出版号 SGI-1.1，可以从 Rockwell Automation 销售办事处或从 http://www.ab.com/manuals/gi 联机获得）介绍了固态设备和硬连线的机电设备之间的重要区别。鉴于这一区别，以及固态设备的广泛用途，凡是负责使用本设备的所有人员，都必须证明本设备的每种专用场合均满足要求。

在任何情况下，对于因使用本设备而带来的附带性或结果性损害，Rockwell Automation, Inc. 将概不负责。

本手册中的示例和图表仅作举例用途。因为与所有的特定安装有关的变数和要求很多，所以对于根据这些示例和图表的实际使用情况，Rockwell Automation, Inc. 概不负责。

对于本手册中介绍的信息、电路、设备或软件的用法，Rockwell Automation, Inc. 将不负任何专利责任。

未经 Rockwell Automation, Inc. 的书面许可，不得全部或部分复制本手册的内容。

在本手册中，我们将根据需要使用各种说明来提醒您有关安全注意事项。

警告



标识与可能在危险的环境中导致爆炸的操作或应用场合有关的信息。这些操作或应用场合可能会造成人身伤害或死亡、财产或经济损失。

重要事项

标识对成功应用和掌握产品至关重要的信息。

注意



标识与可能导致人身伤害或死亡、财产或经济损失的操作或应用场合有关的信息。注意事项将有助于：

- 识别危害
- 避免危害
- 认识后果

有电危险



设备 （如驱动器或发动机）表面或内部的标签用于提醒使用人员可能存在危险电压。

高温危险



设备 （如驱动器或发动机）表面或内部的标签用于提醒使用人员设备表面可能存在高温危险。

简介

本文档的这个版本包含新增和更新的信息。欲查找新增和更新的信息，请查看更新栏，如本段文字旁边所示。

更新信息

本文档包含下列更改：

更改	页码
指令定位器表 — 添加新阶段指令。有关这些指令，请参见 *PhaseManager User Manual*（PhaseManager 用户手册，出版号 LOGIX-UM001）	指令定位器
计时器 — 说明计时器如何累计时间。还提示至少每隔 69 分钟对计时器扫描一次。	2-3、 2-7、 2-11、 2-16、2-20、2-24
消息状态位 — 阐明何时缓冲某些状态位	3-2
消息状态位 — 不要更改 消息 (Message, MSG) 指令的某些状态位	3-2…3-3
消息超时 — 说明连接消息超时和未连接消息超时之间的区别。	3-5
消息 (Message, MSG) 指令 — 关于如何指定路径的更新信息	3-26…3-28
提示某些指令将超出用户自定义的数据类型的成员边界进行读写。	3-34、 7-29、 7-35、 7-44、 8-2、 8-5、 8-9、 8-15、 8-21、 8-27、 9-11、 12-3、 12-11
BTD 指令 — 更正：算术状态标志不受影响	6-11
PID 过程 图表 — 已将 PV-min 更正为 PV-mini	12-41
具有主 / 从属循环的 PID 过程 图表 — 已将 PV-min 更正为 PV-mini	12-41
功能块的计时模式： • 对于定期任务中的定期计时， DeltaT 此时包括小数值。 • 如果进程输入的时间戳与其更新关联，**《且》**需要精确协调，请使用实时采样功能。	B-9
结构化文本 — 添加了与保留的关键字有关的信息	C-12

说明：

何处查找指令

使用此定位器可以查找有关 Logix 指令的参考详细信息（灰显的指令在其他手册中提供）。此定位器还列出了可用于指令的编程语言。

如果定位器列出：	在以下手册中说明该指令：
页号	本手册
动作	*Logix5000 Controllers Motion Instruction Set Reference Manual*（Logix5000 控制器运动指令集参考手册），出版号 1756-RM007
PhaseManager	*PhaseManager User Manual*（PhaseManager 用户手册），出版号 LOGIX-UM001
过程控制	*Logix5000 Controllers Process Control and Drives Instruction Set Reference Manual*（Logix5000 控制器过程控制和驱动器指令集参考手册），出版号 1756-RM006

指令：	位置：	语言：
ABL 缓冲区行的 ASCII 测试	第 16 章 -5	梯形图 结构化文本
ABS 绝对值	第 5 章 -29	梯形图 结构化文本 功能块
ACB 缓冲区中的 ASCII 字符	第 16 章 -8	梯形图 结构化文本
ACL ASCII 清零缓冲区	第 16 章 -10	梯形图 结构化文本
ACOS 反余弦	第 13 章 -14	结构化文本
ACS 反余弦	第 13 章 -14	梯形图 功能块
ADD 加法	第 5 章 -6	梯形图 结构化文本 功能块
AFI 恒假指令	第 10 章 -23	梯形图
AHL ASCII 握手行	第 16 章 -12	梯形图 结构化文本
ALM 警报	过程控制	结构化文本 功能块
AND 按位与	第 6 章 -23	梯形图 结构化文本 功能块
ARD ASCII 读取	第 16 章 -16	梯形图 结构化文本
ARL ASCII 读取行	第 16 章 -19	梯形图 结构化文本

指令：	位置：	语言：
ASIN 反正弦	第 13 章 -11	结构化文本
ASN 反正弦	第 13 章 -11	梯形图 功能块
ATAN 反正切	第 13 章 -17	结构化文本
ATN 反正切	第 13 章 -17	梯形图 功能块
AVE 文件平均值	第 7 章 -38	梯形图
AWA ASCII 写入附加	第 16 章 -23	梯形图 结构化文本
AWT ASCII 写入	第 16 章 -28	梯形图 结构化文本
BAND 布尔与	第 6 章 -35	结构化文本 功能块
BNOT 布尔非	第 6 章 -44	结构化文本 功能块
BOR 布尔或	第 6 章 -38	结构化文本 功能块
BRK 中断	第 11 章 -5	梯形图
BSL 位左移	第 8 章 -2	梯形图
BSR 位右移	第 8 章 -5	梯形图
BTD 位域分配	第 6 章 -11	梯形图

指令：	位置：	语言：
BTDT 带目标的位域分配	第 6 章 -14	结构化文本 功能块
BTR 消息	第 3 章 -2	梯形图 结构化文本
BTW 消息	第 3 章 -2	梯形图 结构化文本
BXOR 布尔异或	第 6 章 -41	结构化文本 功能块
CLR 清零	第 6 章 -17	梯形图 结构化文本
CMP 比较	第 4 章 -2	梯形图
CONCAT 字符串串联	17-3	梯形图 结构化文本
COP 复制文件	第 7 章 -28	梯形图 结构化文本
COS 余弦	第 13 章 -5	梯形图 结构化文本 功能块
CPS 同步复制文件	第 7 章 -28	梯形图 结构化文本
CPT 计算	第 5 章 -2	梯形图
CTD 递减	第 2 章 -32	梯形图
CTU 递增	第 2 章 -28	梯形图
CTUD 递增 / 递减	第 2 章 -36	结构化文本 功能块
D2SD 离散 2 状态设备	过程控制	结构化文本 功能块
D3SD 离散 3 状态设备	过程控制	结构化文本 功能块
DDT 诊断检测	第 12 章 -10	梯形图
DEDT 最后期限	过程控制	结构化文本 功能块
DEG 角度	第 15 章 -2	梯形图 结构化文本 功能块
DELETE 删除字符串	17-5	梯形图 结构化文本
DERV 微分	过程控制	结构化文本 功能块

指令：	位置：	语言：
DFF D 触发器	过程控制	结构化文本 功能块
DIV 除法	第 5 章 -15	梯形图 结构化文本 功能块
DTOS DINT 到字符串	18-8	梯形图 结构化文本
DTR 数据转换	第 12 章 -18	梯形图
EOT 转换结束	第 10 章 -25	梯形图 结构化文本
EQU 等于	第 4 章 -7	梯形图 结构化文本 功能块
ESEL 增强的选择	过程控制	结构化文本 功能块
EVENT 触发事件任务	10-31	梯形图 结构化文本
FAL 文件算术和逻辑	第 7 章 -7	梯形图
FBC 文件位比较	第 12 章 -2	梯形图
FFL FIFO 加载	第 8 章 -8	梯形图
FFU FIFO 卸载	第 8 章 -14	梯形图
FGEN 函数生成器	过程控制	结构化文本 功能块
FIND 查找字符串	17-7	梯形图 结构化文本
FLL 文件填充	第 7 章 -34	梯形图
FOR FOR	第 11 章 -2	梯形图
FRD 转换为整数	第 15 章 -9	梯形图 功能块
FSC 文件搜索和比较	第 7 章 -19	梯形图
GEQ 大于或等于	第 4 章 -11	梯形图 结构化文本 功能块
GRT 大于	第 4 章 -15	梯形图 结构化文本 功能块
GSV 获取系统值	第 3 章 -33	梯形图 结构化文本

指令：	位置：	语言：
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 输入线路连接器	B-1	功能块
INSERT 插入字符串	17-9	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	3-57	梯形图 结构化文本
IREF 输入参考	B-1	功能块
JKFF JK 触发器	过程控制	结构化文本 功能块
JMP 跳到标号	第 10 章 -2	梯形图
JSR 跳转到子例程	第 10 章 -4	梯形图 结构化文本 功能块
JXR 跳到外部例程	第 10 章 -14	梯形图
LBL 标号	第 10 章 -2	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 小于或等于	第 4 章 -19	梯形图 结构化文本 功能块
LES 小于	第 4 章 -23	梯形图 结构化文本 功能块
LFL LIFO 加载	第 8 章 -20	梯形图
LFU LIFO 卸载	第 8 章 -26	梯形图
LIM 限制	第 4 章 -27	梯形图 功能块
LN 自然对数	第 14 章 -2	梯形图 结构化文本 功能块

指令：	位置：	语言：
LOG 底为 10 的对数	第 14 章 -4	梯形图 结构化文本 功能块
LOWER 小写	第 18 章 -14	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	动作	梯形图 结构化文本
MAFR 运动轴故障复位	动作	梯形图 结构化文本
MAG 运动轴齿轮	动作	梯形图 结构化文本
MAHD 运动应用挂起诊断	动作	梯形图 结构化文本
MAH 运动轴主位置	动作	梯形图 结构化文本
MAJ 运动轴缓动	动作	梯形图 结构化文本
MAM 运动轴移动	动作	梯形图 结构化文本
MAOC 运动臂输出凸轮	动作	梯形图 结构化文本
MAPC 运动轴位置凸轮	动作	梯形图 结构化文本
MAR 运动臂对齐	动作	梯形图 结构化文本
MASD 运动轴关闭	动作	梯形图 结构化文本
MAS 运动轴停止	动作	梯形图 结构化文本
MASR 运动轴关闭复位	动作	梯形图 结构化文本
MATC 运动轴时间凸轮	动作	梯形图 结构化文本
MAVE 移动平均	过程控制	结构化文本 功能块
MAW 运动臂观察	动作	梯形图 结构化文本
MAXC 最大捕捉值	过程控制	结构化文本 功能块
MCCD 运动协调更改动态值	动作	梯形图 结构化文本
MCCM 运动协调循环移动	动作	梯形图 结构化文本

指令：	位置：	语言：
MCCP 运动计算凸轮参数	动作	梯形图 结构化文本
MCD 运动更改动态	动作	梯形图 结构化文本
MCLM 运动协调线性移动	动作	梯形图 结构化文本
MCR 主控复位	第 10 章 -19	梯形图
MCSD 运动协调关闭	动作	梯形图 结构化文本
MCS 运动协调停止	动作	梯形图 结构化文本
MCSR 运动协调关闭复位	动作	梯形图 结构化文本
MDF 运动直接驱动关闭	动作	梯形图 结构化文本
MDOC 运动解除输出凸轮	动作	梯形图 结构化文本
MDO 运动直接驱动打开	动作	梯形图 结构化文本
MDR 运动解除对齐	动作	梯形图 结构化文本
MDW 运动解除观察	动作	梯形图 结构化文本
MEQ 掩码等于	第 4 章 -33	梯形图 结构化文本 功能块
MGSD 运动组关闭	动作	梯形图 结构化文本
MGS 运动组停止	动作	梯形图 结构化文本
MGSP 运动组闸门位置	动作	梯形图 结构化文本
MGSR 运动组关闭复位	动作	梯形图 结构化文本
MID 提取字符串	17-11	梯形图 结构化文本
MINC 最小值捕获	过程控制	结构化文本 功能块
MOD 模	第 5 章 -19	梯形图 结构化文本 功能块
MOV 移动	第 6 章 -3	梯形图

指令：	位置：	语言：
MRAT 运动运行轴调整	动作	梯形图 结构化文本
MRHD 运动运行挂起诊断	动作	梯形图 结构化文本
MRP 运动重新定义位置	动作	梯形图 结构化文本
MSF 运动伺服关闭	动作	梯形图 结构化文本
MSG 消息	第 3 章 -2	梯形图 结构化文本
MSO 运动伺服打开	动作	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	第 5 章 -12	梯形图 结构化文本 功能块
MUX 多路复用器	过程控制	功能块
MVM 掩码移动	第 6 章 -5	梯形图
MVMT 带目标的掩码移动	第 6 章 -8	结构化文本 功能块
NEG 求反	第 5 章 -26	梯形图 结构化文本 功能块
NEQ 不等于	第 4 章 -38	梯形图 结构化文本 功能块
NOP 空操作	第 10 章 -24	梯形图
NOT 按位非	第 6 章 -32	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 输出线路连接器	B-1	功能块
ONS 单触发	第 1 章 -12	梯形图
OR 按位或	第 6 章 -26	梯形图 结构化文本 功能块
OREF 输出参考	B-1	功能块
OSFI 带输入的下降沿单触发	第 1 章 -22	结构化文本 功能块

指令：	位置：	语言：
OSF 下降沿单触发	第 1 章 -17	梯形图
OSRI 带输入的上升沿单触发	第 1 章 -19	结构化文本 功能块
OSR OSR (One Shot Rising)	第 1 章 -15	梯形图
OTE 输出激励	第 1 章 -6	梯形图
OTL 输出锁存	第 1 章 -8	梯形图
OTU 输出解锁存	第 1 章 -10	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 增强的 PID	过程控制	结构化文本 功能块
PID 按比例积分微分	第 12 章 -21	梯形图 结构化文本
PI 按比例 + 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

指令：	位置：	语言：
RAD 弧度	第 15 章 -4	梯形图 结构化文本 功能块
RESD 复位支配	过程控制	结构化文本 功能块
RES 复位	第 2 章 -40	梯形图
RET 返回	第 10 章 -4 和 第 11 章 -6	梯形图 结构化文本 功能块
RLIM 变化率限制器	过程控制	结构化文本 功能块
RMPS 上升 / 保持	过程控制	结构化文本 功能块
RTO 保持计时器打开	第 2 章 -10	梯形图
RTOR 带有复位的保持计时器打开	第 2 章 -23	结构化文本 功能块
RTOS REAL 到字符串	18-10	梯形图 结构化文本
SBR 子例程	第 10 章 -4	梯形图 结构化文本 功能块
SCL 范围	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 置位支配	过程控制	结构化文本 功能块
SFP SFC 暂停	第 10 章 -27	梯形图 结构化文本
SFR SFC 复位	第 10 章 -29	梯形图 结构化文本
SIN 正弦	第 13 章 -2	梯形图 结构化文本 功能块
SIZE 元素大小	7-53	梯形图 结构化文本
SNEG 选定的求反	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

指令：	位置：	语言：
SQI 定序程序输入	第 9 章 -2	梯形图
SQL 定序程序加载	第 9 章 -10	梯形图
SQO 定序程序输出	第 9 章 -6	梯形图
SQR 平方根	第 5 章 -23	梯形图 功能块
SQRT 平方根	第 5 章 -23	结构化文本
SRT 文件排序	第 7 章 -43	梯形图 结构化文本
SRTP 分割范围时间比例	过程控制	结构化文本 功能块
SSUM 选定的加法器	过程控制	结构化文本 功能块
SSV 设置系统值	第 3 章 -33	梯形图 结构化文本
STD 文件标准偏差	第 7 章 -48	梯形图
STOD 字符串到 DINT	18-4	梯形图 结构化文本
STOR 字符串到 REAL	18-6	梯形图 结构化文本
SUB 减法	第 5 章 -9	梯形图 结构化文本 功能块
SWPB 交换字节	6-19	梯形图 结构化文本
TAN 正切	第 13 章 -8	梯形图 结构化文本 功能块
TND 临时结束	第 10 章 -17	梯形图
TOD 转换为 BCD	第 15 章 -6	梯形图 功能块
TOFR 带有复位的计时器关闭延迟	第 2 章 -19	结构化文本 功能块
TOF 计时器关闭延迟	第 2 章 -6	梯形图
TONR 带有复位的计时器打开延迟	第 2 章 -15	结构化文本 功能块

指令：	位置：	语言：
TON 计时器打开延迟	第 2 章 -2	梯形图
TOT 累加器	过程控制	结构化文本 功能块
TRN 截断	第 15 章 -11	梯形图 功能块
TRUNC 截断	第 15 章 -11	结构化文本
UID 禁止用户中断	第 10 章 -21	梯形图 结构化文本
UIE 允许用户中断	第 10 章 -21	梯形图 结构化文本
UPDN 递增 / 递减累加器	过程控制	结构化文本 功能块
UPPER 大写	第 18 章 -12	梯形图 结构化文本
XIC 检查是否已关闭	第 1 章 -2	梯形图
XIO 检查是否已打开	第 1 章 -4	梯形图
XOR 按位异或	第 6 章 -29	梯形图 结构化文本 功能块
XPY X 的 Y 次方	第 14 章 -6	梯形图 结构化文本 功能块

简介

本手册是 Logix5000 指令手册系列之一。

任务 / 目的：	文档：
针对序列应用程序对控制器进行编程 即本手册 	*Logix5000 Controllers General Instructions Reference Manual*（Logix5000 控制器指令集参考手册），出版号 1756-RM003s
针对过程或驱动器应用程序对控制器进行编程	*Logix5000 Controllers Process Control and Drives Instructions Reference Manual*（Logix5000 控制器过程控制和驱动器指令参考手册），出版号 1756-RM006
针对运动应用程序对控制器进行编程	*Logix5000 Controllers Motion Instruction Set Reference Manual*（Logix5000 控制器运动指令集参考手册），出版号 1756-RM007
对控制器编程使其可以使用设备相位	*PhaseManager User Manual*（PhaseManager 用户手册），出版号 LOGIX-UM001
将文本文件或标记导入项目	*Logix5000 Controllers Import/Export Reference Manual*（Logix5000 控制器导入 / 导出参考手册），出版号 1756-RM084
将项目或标记导出到文本文件	
将 PLC-5 或 SLC 500 应用程序转换为 Logix5000 应用程序	*Logix5550 Controller Converting PLC-5 or SLC 500 Logic to Logix5550 Logic Reference Manual*（Logix5550 控制器 PLC-5 或 SLC 500 逻辑到 Logix5550 逻辑转换参考手册），出版号 1756-6.8.5

本手册的使用对象

本文档为编程人员提供了关于 Logix 型控制器每条可用指令的详细信息。用户应该已经熟悉了 Logix 型控制器存储和处理数据的方式。

使用指令之前，初学编程的人员应该阅读与指令有关的所有详细信息。经验丰富的编程人员可以参考指令信息以确认详细内容。

本手册的用途

本手册采用下列格式说明每条指令。

在以下部分：	提供以下类型的信息：
指令名称	标识指令 定义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  如果在梯形图中可用，则描述操作数  如果在结构化文本中可用，则描述操作数  如果在功能块中可用，则描述操作数 默认功能块上显示的引脚只是默认引脚。操作数表列出了功能块的所有可能引脚。
指令结构	列出指令的控制状态位和值 （如果有）
说明	说明指令的用法 定义指令启用和禁用时的所有区别 （如果需要）
算术状态标志	定义指令是否影响算术状态标志 请参见附录常用属性
错误条件	定义指令是否产生轻微错误或严重错误 如果产生，则定义错误类型和代码
执行	定义指令操作方式的具体信息
示例	使用每种可用的编程语言至少提供一个编程示例 每个示例都包含解释说明

下列图标有助于识别与语言有关的信息：

图标：	指示的编程语言：
	梯形图
	结构化文本
	功能块

所有指令的常用信息

Logix5000 指令集具有一些常用属性：

对于以下信息：	请参见以下附录：
常用属性	附录常用属性定义了： • 算术状态标志 • 数据类型 • 关键字
功能块属性	附录功能块属性定义了： • 程序和操作员控制 • 计时模式

约定和相关项

置位和清零

本手册使用置位和清零来定义位 （布尔型）和数值 （非布尔型）的状态：

项：	含义：
置位	将位设置为 1 (ON) 将数值设置为任何一个非零数
清零	清零位 (OFF) 清零数值中的所有位

如果操作数或参数支持多个数据类型，**黑体字**数据类型是最佳数据类型。如果指令的所有操作数都使用同一最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

梯形图梯级条件

控制器根据指令前面的梯级条件 （梯级输入条件）判断梯形图指令。然后，根据梯级输入条件和指令，控制器可以设置指令后面的梯级条件（梯级输出条件），进而影响任何后续指令。



如果输入指令的梯级输入条件为 true，控制器将会对指令进行判断，然后根据指令的结果设置梯级输出条件。如果指令被判断为 true，则梯级输出条件为 true ；如果指令被判断为 false，则梯级输出条件为 false。

控制器还可以对指令进行预扫描。预扫描是一种在控制器中对所有例程进行的特殊扫描。控制器在预扫描期间会扫描所有的主例程和子例程，但是会忽略可以跳过执行指令的跳转。控制器执行所有的 FOR 循环和子例程调用。如果多次调用一个子例程，每次调用该例程时，都会执行该例程。控制器可以通过预扫描梯形图指令重置非保留的 I/O 和内部值。

在预扫描期间，输入值不是最新的，且输出值不能被写入。在下列条件下，将会生成预扫描：

- 从 “编程”模式切换到 “运行”模式
- 开机时自动进入 “运行”模式。

在下列条件下，不会对程序进行预扫描：

- 在控制器运行时排定程序。
- 控制器进入 “运行”模式时取消排定程序。

功能块状态

重要事项 在功能块中编程时，请将执行单元的范围限制到 $+/-10^{+/-15}$，原因是内部浮点计算是使用单精度浮点数进行的。如果结果超出单精度浮点数的限制 ($+/-10^{+/-38}$)，此范围外的执行单元可能会有精度损失。

控制器将会根据各种情况的状态对功能块指令进行判断。

可能的条件：	说明：
预扫描	预扫描功能块例程和预扫描梯形图例程是相同的。唯一的区别在于，在预扫描期间会对每个功能块指令的 EnableIn 参数进行清零。
指令第一次扫描	指令第一次扫描指的是预扫描后指令的首次执行。控制器利用指令第一次扫描读取当前的输入，并确定相应的输入状态。
指令第一次运行	指令第一次运行指的是第一次利用数据结构的新实例执行指令。 控制器利用通过指令第一次运行生成系数和其他数据存储，这些系数和数据存储在功能块首次下载后不会变化。

每个功能块指令还包括 EnableIn 和 EnableOut 参数：

- 设置 EnableIn 之后，功能块指令可以正常执行。
- EnableIn 清零之后，功能块指令可以执行预扫描逻辑、扫描后逻辑或仅跳过正常的算术执行。
- 但是，如果执行功能块时检测到还对溢出条件 EnableOut 进行了清零， Enableout 将镜像 EnableIn。
- 如果将 EnableIn 从清零状态切换到置位状态，将从停止位置继续执行功能块。但是，如果将 EnableIn 从清零状态切换到置位状态，将存在一些指定特殊功能的功能块指令，如重新初始化。对于包含时间基参数的功能块指令，只要计时模式是超采样，EnableIn 从清零状态切换为置位状态时，将始终从停止位置继续执行指令。

如果未连接 EnableIn 参数，指令将始终照常执行，且 EnableIn 保持设置状态。如果对 EnableIn 清零，下次执行指令时候，该参数将会更改为置位状态。

说明：

目录

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

第 1 章

简介 1-1
检查是否已关闭 (Examine If Closed, XIC) 1-2
检查是否已打开 (Examine If Open, XIO) 1-4
输出激励 (Output Energize, OTE) 1-6
输出锁存 (Output Latch, OTL) 1-8
输出解锁 (Output Unlatch, OTU) 1-10
单次触发 (One Shot, ONS) 1-12
上升沿单次触发 (One Shot Rising, OSR) 1-15
下降沿单次触发 (One Shot Falling, OSF) 1-17
带输入的上升沿单次触发 (One Shot Rising with Input, OSRI) 1-19
带输入的下降沿单次触发 (One Shot Falling with Input, OSFI) 1-22

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

第 2 章

简介 2-1
计时器打开延迟 (Timer On Delay, TON) 2-2
计时器关闭延迟 (Timer Off Delay, TOF) 2-6
保持计时器打开 (Retentive Timer On, RTO) 2-10
带有复位的计时器打开延迟 (Timer On Delay with Reset, TONR) 2-15
带有复位的计时器关闭延迟 (Timer Off Delay with Reset, TOFR) 2-19
带有复位的保持计时器打开 (Retentive Timer On with Reset, RTOR) 2-23
递增 (Count Up, CTU) 2-28
递减 (Count Down, CTD) 2-32
递增 / 递减 (CTUD) 2-36
复位 (Reset, RES) 2-40

输入 / 输出指令（MSG、GSV、SSV、IOT）

第 3 章

简介 3-1
消息 (Message, MSG) 3-2
MSG 错误代码 3-9
- 错误代码 3-9
- 扩展错误代码 3-11
- PLC 和 SLC 错误代码 (.ERR) 3-13
- 块传输错误代码 3-15

指定配置详细信息 3-16
- 指定 CIP Data Table Read （CIP 数据表读取）和 CIP Data Table Write （CIP 数据表写入）消息 3-17
- 重新配置 I/O 模块 3-18
- 指定 CIP Generic （CIP 通用）消息 3-19
- 指定 PLC-5 消息 3-20

指定 SLC 消息 3-22
指定块传输消息 3-22
指定 PLC-3 消息 3-23
指定 PLC-2 消息 3-24
MSG 配置示例 3-25
指定通信详细信息 3-26
指定路径 3-26
对于块传输 3-28
指定通信方法或模块地址 3-29
选择缓存选项 3-30
指导原则 3-32
获取系统值 （Get System Value， GSV）和设置系统值（Set System Value， SSV） 3-33
GSV/SSV 对象 3-36
访问 CONTROLLER 对象 3-37
访问 CONTROLLERDEVICE 对象 3-37
访问 CST 对象 3-39
访问 DF1 对象 3-40
访问 FAULTLOG 对象 3-43
访问 MESSAGE 对象 3-44
访问 MODULE 对象 3-46
访问 MOTIONGROUP 对象 3-47
访问 PROGRAM 对象 3-48
访问 ROUTINE 对象 3-49
访问 SERIALPORT 对象 3-49
访问 TASK 对象 3-51
访问 WALLCLOCKTIME 对象 3-53
GSV/SSV 编程示例 3-54
获取错误信息 3-54
设置启用和禁用标记 3-56
立即输出 (Immediate Output, IOT) 3-57

第 4 章

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

简介 4-1
比较 (Compare, CMP) 4-2
CMP 表达式 4-4
有效运算符 4-4
格式化表达式 4-5
确定运算次序 4-5
在表达式中使用字符串 4-6
等于 (Equal to, EQU) 4-7
大于或等于 (Greater than or Equal to, GEQ) 4-11
大于 (Greater Than, GRT) 4-15
小于或等于 (Less Than or Equal to, LEQ) 4-19
小于 (Less Than, LES) 4-23

限制 (Limit, LIM) 4-27
掩码等于
(Mask Equal to, MEQ) 4-33
输入立即数掩码值 4-34
不等于 (Not Equal to, NEQ) 4-38

第 5 章

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

简介 5-1
计算 (Compute, CPT) 5-2
有效运算符 5-4
格式化表达式 5-4
确定运算次序 5-5
加法 (Add, ADD) 5-6
减法 (Subtract, SUB) 5-9
乘法 (Multiply, MUL) 5-12
除法 (Divide, DIV) 5-15
取余 (Modulo, MOD) 5-19
平方根 (Square Root, SQR) 5-23
求反 (Negate, NEG) 5-26
绝对值 (Absolute Value, ABS) 5-29

第 6 章

移动 / 逻辑指令（MOV、MVM、BTD、MVMT、BTDT、CLR、SWPB、AND、OR、XOR、NOT、BAND、BOR、BXOR、BNOT）

简介 6-1
移动 (Move, MOV) 6-3
掩码移动 (Masked Move, MVM) 6-5
输入立即掩码值 6-6
带目标的掩码移动 (Masked Move with Target, MVMT) 6-8
位域分配 (Bit Field Distribute, BTD) 6-11
带目标的位域分配
(Bit Field Distribute with Target, BTDT) 6-14
清零 (Clear, CLR) 6-17
交换字节 (Swap Byte, SWPB) 6-19
按位与 (Bitwise AND, AND) 6-23
按位或 (Bitwise OR, OR) 6-26
按位异或 (Bitwise Exclusive OR, XOR) 6-29
按位非 (Bitwise NOT, NOT) 6-32
布尔与 (Boolean AND, BAND) 6-35
布尔或 (Boolean OR, BOR) 6-38
布尔异或 (Boolean Exclusive OR, BXOR) 6-41
布尔非 (Boolean NOT, BNOT) 6-44

第 7 章

数组（文件）/ 综合指令（FAL、FSC、COP、CPS、FLL、AVE、SRT、STD、SIZE）

简介 7-1
选择运算模式 7-2
整体模式 7-2
数值模式 7-3
增量模式 7-5
文件算术和逻辑 (File Arithmetic and Logic, FAL) 7-7
FAL 表达式 7-16
有效运算符 7-17
格式化表达式 7-17
确定运算次序 7-18
文件搜索和比较 (File Search and Compare, FSC) 7-19
FSC 表达式 7-24
有效运算符 7-25
格式化表达式 7-25
确定运算次序 7-26
在表达式中使用字符串 7-27
复制文件 (COPY FILE, COP) 同步复制文件 (Synchronous Copy File, CPS) 7-28
文件填充 (File Fill, FLL) 7-34
文件平均值 (File Average, AVE) 7-38
文件排序 (File Sort, SRT) 7-43
文件标准偏差 (File Standard Deviation, STD) 7-48
元素的大小 (Size In Elements, SIZE) 7-53

第 8 章

数组（文件）/ 移位指令（BSL、BSR、FFL、FFU、LFL、LFU）

简介 8-1
位左移 (Bit Shift Left, BSL) 8-2
位右移 (Bit Shift Right, BSR) 8-5
FIFO 加载 (FIFO Load, FFL) 8-8
FIFO 卸载 (FIFO Unload, FFU) 8-14
LIFO 加载 (LIFO Load, LFL) 8-20
LIFO 卸载 (LIFO Unload, LFU) 8-26

定序程序指令（SQI、SQO、SQL）

第 9 章

简介 9-1
定序程序输入 (Sequencer Input, SQI) 9-2
输入立即数掩码值 9-3
只用 SQI 不用 SQO 9-5
定序程序输出 (Sequencer Output, SQO) 9-6
输入立即数掩码值 9-7
SQI 与 SQO 一起使用 9-9
复位 SQO 的位置 9-9
定序程序加载 (Sequencer Load, SQL) 9-10

程序控制指令（JMP、LBL、JSR、RET、SBR、JXR、TND、MCR、UID、UIE、AFI、NOP、EOT、SFP、SFR、EVENT）

第 10 章

简介 10-1
跳转至标号 (Jump to Label, JMP) 标号 (Label, LBL) 10-2
跳转到子例程 (Jump to Subroutine, JSR)
子例程 (Subroutine, SBR) 返回 (Return, RET) 10-4
跳转到外部例程 (Jump to External Routine, JXR) 10-14
临时结束 (Temporary End, TND) 10-17
主控复位 (Master Control Reset, MCR) 10-19
禁止用户中断 (User Interrupt Disable, UID) 允许用户中断 (User Interrupt Enable, UIE) 10-21
恒假指令 (Always False Instruction, AFI) 10-23
空操作 (No Operation, NOP) 10-24
转换结束 (End of Transition, EOT) 10-25
SFC 暂停 (SFC Pause, SFP) 10-27
SFC 复位 (SFC Reset, SFR) 10-29
触发事件任务 (Trigger Event Task, EVENT) 10-31
以编程的方式确定 EVENT 指令是否触发了任务 10-31

For/Break 指令（FOR、FOR...DO、BRK、EXIT、RET）

第 11 章

简介 11-1
循环 (For, FOR) 11-2
终止 (Break, BRK) 11-5
返回 (Return, RET) 11-6

特殊指令（FBC、DDT、DTR、PID）

第 12 章

简介 12-1
文件位比较 (File Bit Comparison, FBC) 12-2
选择搜索模式 12-4
诊断检测 (Diagnostic Detect, DDT) 12-10
选择搜索模式 12-12
数据传送 (Data Transitional, DTR) 12-18
输入立即掩码值 12-19
比例积分微分 (Proportional Integral Derivative, PID) 12-21

配置 PID 指令 12-26
指定调节 12-27
指定配置 12-27
指定警报 12-28
指定比例 12-29
使用 PID 指令 12-29
防止积分饱和及由手动向自动的无冲击转换 12-31
PID 指令计时 12-32
无冲击重新启动 12-36
微分平滑 12-37
设置死区 12-38
使用输出限制 12-38
前馈或输出偏置 12-39
级联循环 12-39
控制比率 12-40
PID 原理 12-41
PID 过程 12-41
具有主 / 从属循环的 PID 过程 12-41

第 13 章

三角指令（SIN、COS、TAN、ASN、ASIN、ACS、ACOS、ATN、ATAN）

简介 13-1
正弦 (Sine, SIN) 13-2
余弦 (Cosine, COS) 13-5
正切 (Tangent, TAN) 13-8
反正弦 (Arc Sine, ASN) 13-11
反余弦 (Arc Cosine, ACS) 13-14
反正切 (Arc Tangent, ATN) 13-17

第 14 章

高级数学指令（LN、LOG、XPY）

简介 14-1
自然对数 (Natural Log, LN) 14-2
以 10 为底的对数 (Log Base 10, LOG) 14-4
X 的 Y 次幂 (X to the Power of Y, XPY) 14-6

第 15 章

数学转换指令（DEG、RAD、TOD、FRD、TRN 和 TRUNC）

简介 15-1
角度 (Degrees, DEG) 15-2
弧度 (Radian, RAD) 15-4
转换为 BCD (Convert to BCD, TOD) 15-6
转换为整数 (Convert to Integer, FRD) 15-9
截断 (Truncate, TRN) 15-11

第 16 章

ASCII 串行端口指令（ABL、ACB、ACL、AHL、ARD、ARL、AWA、AWT）

简介 16-1
指令执行 16-2
ASCII 错误代码 16-4
字符串数据类型 16-4
缓冲区行的 ASCII 测试 (ASCII Test For Buffer Line, ABL) . . 16-5
缓冲区中的 ASCII 字符 (ASCII Chars in Buffer, ACB) 16-8
ASCII 清除缓冲区 (ASCII Clear Buffer, ACL) 16-10
ASCII 握手行 (ASCII Handshake Lines, AHL) 16-12
ASCII 读取 (ASCII Read, ARD) 16-16
ASCII 读取行 (ASCII Read Line, ARL) 16-19
ASCII 写入附加 (ASCII Write Append, AWA) 16-23
ASCII 写入 (ASCII Write, AWT) 16-28

第 17 章

ASCII 字符串指令（CONCAT、DELETE、FIND、INSERT、MID）

简介 17-1
字符串数据类型 17-2
字符串串联 (String Concatenate, CONCAT) 17-3
删除字符串 (String Delete, DELETE) 17-5
查找字符串 (Find String, FIND) 17-7
插入字符串 (Insert String, INSERT) 17-9
提取字符串 (Middle String, MID) 17-11

第 18 章

ASCII 转换指令（STOD、STOR、DTOS、RTOS、UPPER、LOWER）

简介 18-1
字符串数据类型 18-3
字符串到 DINT (String To DINT, STOD) 18-4
字符串到 REAL (String To REAL, STOR) 18-6
DINT 到字符串 (DINT to String, DTOS) 18-8
REAL 到字符串 (REAL to String, RTOS) 18-10
大写 (Upper Case, UPPER) 18-12
小写 (Lower Case, LOWER) 18-14

附录 A

常用属性

简介 A-1
立即数 A-1
数据转换 A-1
SINT 或 INT 转换为 DINT A-3
整数转换为 REAL A-5
DINT 转换为 SINT 或 INT A-5
REAL 转换为整数 A-6

附录 B

功能块属性

简介 B-1
选择功能块元素 B-1
锁存数据 B-2
执行顺序 B-4
解析循环 B-5
解析两个块之间的数据流 B-6
创建一次扫描延迟 B-7
总结 B-7
功能块对溢出情况的响应 B-8
计时模式 B-9
计时模式的通用指令参数 B-10
计时模式概述 B-12
程序 / 操作员控制. B-13

附录 C

结构化文本编程

简介 C-1
结构化文本语法 C-1
赋值 C-2
指定非保持赋值 C-3
将 ASCII 字符赋值给字符串 C-4
表达式 C-4
使用算术运算符和算术函数 C-6
使用关系运算符 C-7
使用逻辑运算符 C-9
使用位运算符 C-10
确定执行顺序 C-10
指令 C-11
结构 C-12
某些关键字保留供将来使用 C-12
IF...THEN C-13
CASE...OF C-16
FOR···DO C-19
WHILE···DO. C-22
REPEAT···UNTIL. C-25
注释 C-28

第 1 章

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

简介

位 （继电类型）指令用于监视和控制位的状态。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
当设置了位时启用输出	XIC	梯形图 结构化文本(1)	第 1 章 -2
当位清零时启用输出	XIO	梯形图 结构化文本 (1)	第 1 章 -4
置位	OTE	梯形图 结构化文本 (1)	第 1 章 -6
置位 （保持）	OTL	梯形图 结构化文本 (1)	第 1 章 -8
位清零 （保持）	OTU	梯形图 结构化文本 (1)	第 1 章 -10
每当梯级变为 true 时，每次扫描都启用输出一次	ONS	梯形图 结构化文本 (1)	第 1 章 -12
每当梯级变为 true 时，每次扫描都置位一次	OSR	梯形图	第 1 章 -15
每当梯级变为 false 时，每次扫描都置位一次	OSF	梯形图	第 1 章 -17
每当在功能块中设置了输入位时，每次扫描都置位一次	OSRI	结构化文本 功能块	第 1 章 -19
每当在功能块中清除了输入位时，每次扫描都置位一次	OSFI	结构化文本 功能块	第 1 章 -22

(1) 没有等效的结构化文本指令。使用其他结构化文本进行编程获得同样的结果。请参见该指令的说明。

检查是否已关闭 (Examine If Closed, XIC)

XIC 指令检查数据位是否已设置。

操作数：



梯形图

操作数：	类型：	格式：	说明：
数据位	BOOL	标记	要测试的位

结构化文本

结构化文本没有 XIC 指令，但您可以使用 IF...THEN 结构实现相同的结果。

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

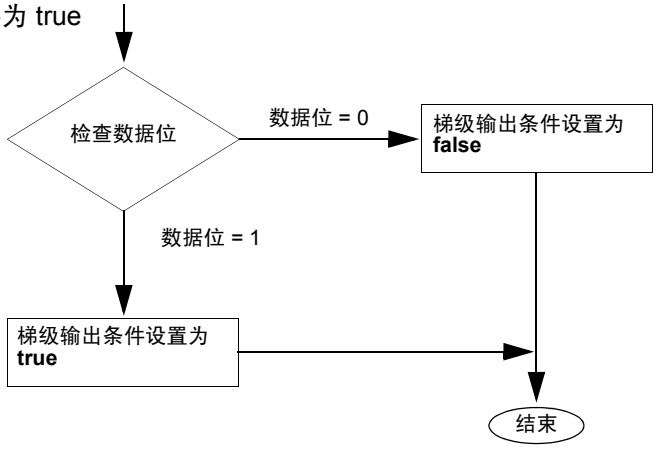
有关结构化文本内的结构语法的信息，请参见附录 C。

说明： XIC 指令检查数据位是否已设置。

算术状态标志： 不受影响

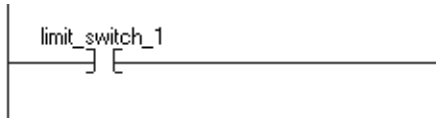
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	
后期扫描	梯级输出条件设置为 false。

示例 1： 如果设置了 *limit_switch_1*，这将启用下一条指令 （梯级输出条件为 true）。

梯形图

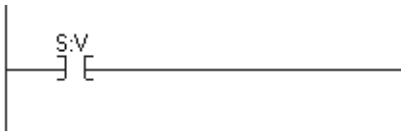


结构化文本

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

示例 2： 如果设置了 *S:V*（表示发生了溢出），这将启用下一条指令 （梯级输出条件为 true）。

梯形图



结构化文本

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

检查是否已打开 (Examine If Open, XIO)

XIO 指令检查数据位是否已清除。

操作数：

梯形图



操作数：	类型：	格式：	说明：
数据位	BOOL	标记	要测试的位

结构化文本

结构化文本没有 XIO 指令，但您可以使用 IF...THEN 结构实现相同的结果。

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

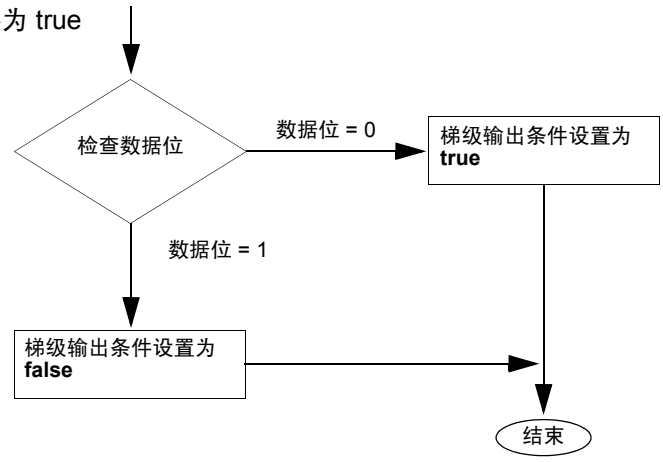
有关结构化文本内的结构语法的信息，请参见附录 C。

说明： XIO 指令检查数据位是否已清除。

算术状态标志： 不受影响

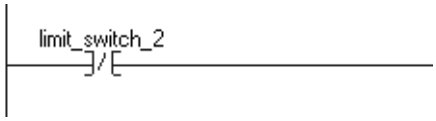
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	
后期扫描	梯级输出条件设置为 false。

示例 1： 如果清除了 *limit_switch_2*，这将启用下一条指令 （梯级输出条件为 true）。

梯形图

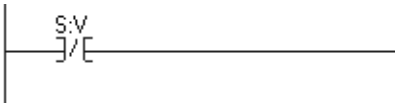


结构化文本

```
IF NOT limit_switch_2 THEN
    <statement>;
END_IF;
```

示例 2： 如果清除了 *S:V*（表示未发生溢出），这将启用下一条指令 （梯级输出条件为 true）。

梯形图



结构化文本

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

输出激励 (Output Energize, OTE)

OTE 指令设置或清除数据位。

操作数：

梯形图



操作数：	类型：	格式：	说明：
数据位	BOOL	标记	要设置或清除的位

结构化文本

结构化文本没有 OTE 指令，但您可以使用非保持赋值语句实现相同的结果。

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

有关结构化文本内的赋值语句和表达式的语法的信息，请参见附录 C。

说明： 当 OTE 指令启用时，控制器设置数据位。当 OTE 指令禁用时，控制器清除数据位。

算术状态标志： 不受影响

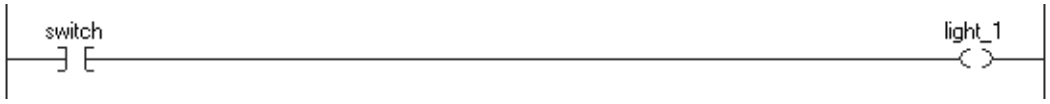
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	清除数据位。 梯级输出条件设置为 false。
梯级输入条件为 false	清除数据位。 梯级输出条件设置为 false。
梯级输入条件为 true	设置数据位。 梯级输出条件设置为 true。
后期扫描	清除数据位。 梯级输出条件设置为 false。

示例： 当设置了 *switch* 时， OTE 指令设置 （打开） *light_1*。当清除了 *switch* 时， OTE 指令清除 （关闭） *light_1*。

梯形图



结构化文本

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

输出锁存 (Output Latch, OTL)

OTL 指令设置 （锁存）数据位。

操作数：

梯形图



操作数：	类型：	格式：	说明：
数据位	BOOL	标记	要设置的位

结构化文本

结构化文本没有 OTL 指令，但您可以使用 IF...THEN 结构和赋值语句实现相同的结果。

```
IF BOOL_expression THEN
    data_bit := 1;
END_IF;
```

有关结构化文本内的结构、表达式和赋值语句的语法的信息，请参见附录 C。

说明： 启用时， OTL 指令设置数据位。该数据位保持设置状态，直到被清除。清除操作通常由 OTU 指令来执行。禁用时， OTL 指令不更改数据位的状态。

算术状态标志： 不受影响

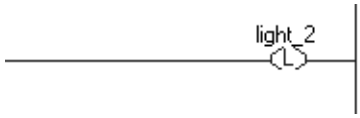
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	数据位未修改。 梯级输出条件设置为 false。
梯级输入条件为 false	数据位未修改。 梯级输出条件设置为 false。
梯级输入条件为 true	设置数据位。 梯级输出条件设置为 true。
后期扫描	数据位未修改。 梯级输出条件设置为 false。

示例： 启用时，OTL 指令设置 *light_2*。该数据位保持设置状态，直到被清除。清除操作通常由 OTU 指令来执行。

梯形图



结构化文本

```
IF BOOL_expression THEN
    light_2 := 1;
END_IF;
```

输出解锁 (Output Unlatch, OTU)

OTU 指令清除 （解锁存）数据位。

操作数：

梯形图



操作数：	类型：	格式：	说明：
数据位	BOOL	标记	要清除的位

结构化文本

结构化文本没有 OTU 指令，但您可以使用 IF...THEN 结构和赋值语句实现相同的结果。

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

有关结构化文本内的结构、表达式和赋值语句的语法的信息，请参见附录 C。

说明： 启用时， OTU 指令清除数据位。禁用时， OTU 指令不更改数据位的状态。

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	数据位未修改。 梯级输出条件设置为 false。
梯级输入条件为 false	数据位未修改。 梯级输出条件设置为 false。
梯级输入条件为 true	清除数据位。 梯级输出条件设置为 true。
后期扫描	数据位未修改。 梯级输出条件设置为 false。

示例： 启用时， OTU 指令清除 *light_2*。

梯形图



结构化文本

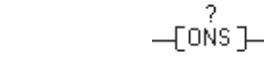
```
IF BOOL_expression THEN
    light_2 := 0;
END_IF;
```

单次触发 (One Shot, ONS)

ONS 指令根据存储位的状态启用或禁用梯级的其余部分。

操作数：

梯形图



操作数：	类型：	格式：	说明：
存储位	BOOL	标记	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，但您可以使用 IF...THEN 结构实现相同的结果。

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

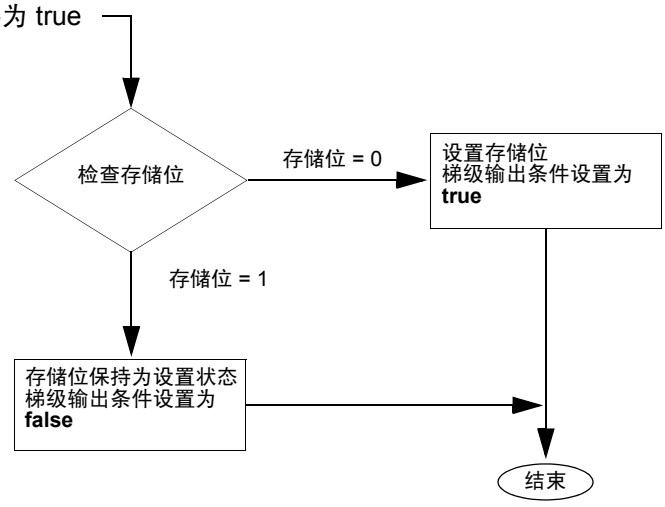
有关结构化文本内的结构、表达式和赋值语句的语法的信息，请参见附录 C。

说明： 启用时，如果清除了存储位，ONS 指令将启用梯级的其余部分。当禁用时或者当设置了存储位时， ONS 指令将禁用梯级的其余部分。

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	设置存储位以防止第一次扫描期间出现无效触发 梯级输出条件设置为 false。
梯级输入条件为 false	清除存储位。 梯级输出条件设置为 false。
梯级输入条件为 true 	
后期扫描	清除存储位。 梯级输出条件设置为 false。

示例： 通常需要在 ONS 指令前加上输入指令，因为您会在启用和禁用 ONS 指令时对它进行扫描，以使它正常工作。一旦启用了 ONS 指令，必须清除梯级输入条件或存储位， ONS 指令才能再次启用。

在清除了 *limit_switch_1* 或设置了 *storage_1* 的任何扫描上，此梯级没有任何作用。在设置了 *limit_switch_1* 并清除了 *storage_1* 的任何扫描上，ONS 指令设置 *storage_1*，并且 ADD 指令将 *sum* 加 1。只要 *limit_switch_1* 保持设置状态，*sum* 就保持相同的值不变。*limit_switch_1* 必须再次由清除状态变为设置状态， *sum* 才能再次递增。

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;
```

上升沿单次触发 (One Shot Rising, OSR)

OSR 指令根据存储位的状态设置或清除输出位。

该指令在结构化文本和功能块中可以作为 OSRI 使用，请参见第 第 1 章 -19 页。

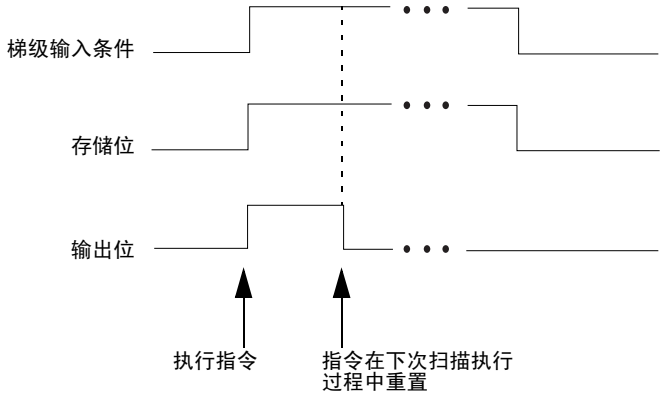
操作数：

梯形图



操作数：	类型：	格式：	说明：
存储位	BOOL	标记	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	标记	要设置的位

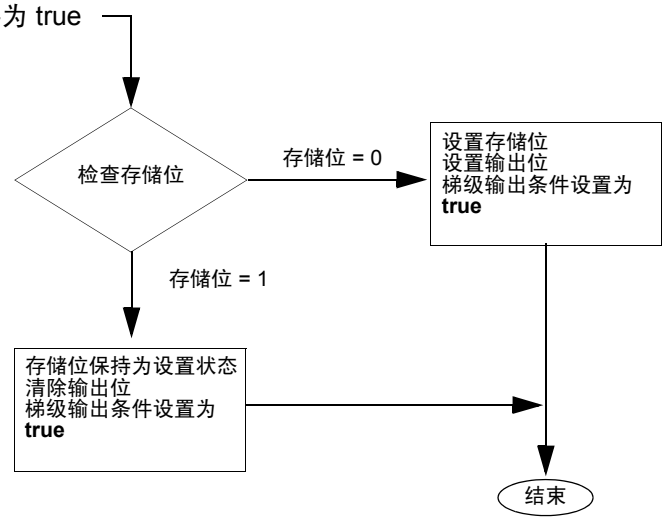
说明： 启用时，如果清除了存储位， OSR 指令将设置输出位。当已启用但存储位已设置时或者当已禁用时， OSR 指令清除输出位



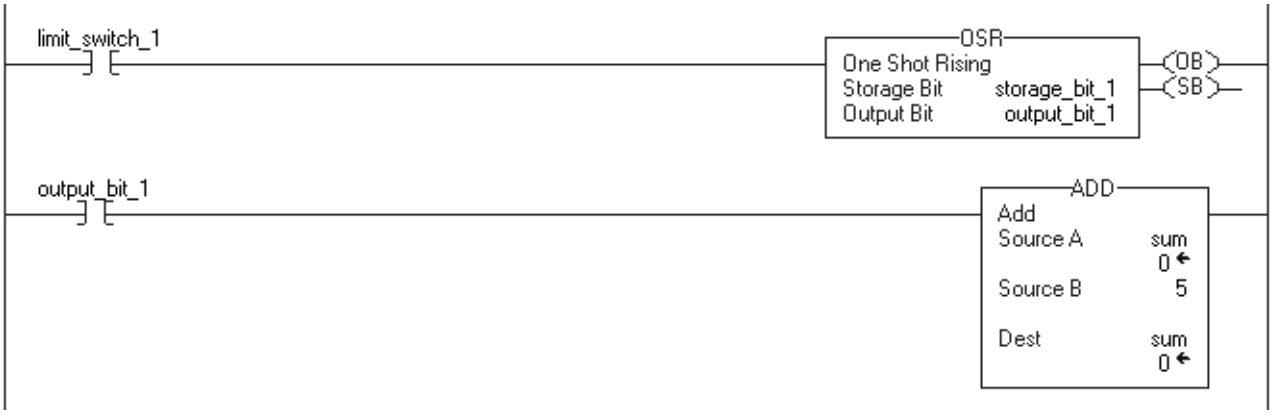
算术状态标志： 不受影响

错误条件： 无

执行：

条件：	梯形图操作：
预扫描	设置存储位以防止第一次扫描期间出现无效触发。 清除输出位。 梯级输出条件设置为 false。
梯级输入条件为 false	清除存储位。 输出位未修改。 梯级输出条件设置为 false。
梯级输入条件为 true	
后期扫描	清除存储位。 输出位未修改。 梯级输出条件设置为 false。

示例： 每当 *limit_switch_1* 从清除状态变为设置状态时， OSR 指令都设置 *output_bit_1*，并且 ADD 指令将 *sum* 加 5。只要 *limit_switch_1* 保持设置状态， *sum* 就保持相同的值不变。 *limit_switch_1* 必须再次由清除状态变为设置状态， *sum* 才能再次递增。可以在多个梯级上使用 *output_bit_1* 以触发其他操作



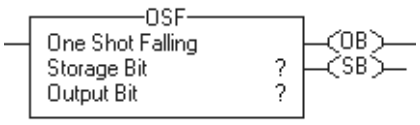
下降沿单次触发 (One Shot Falling, OSF)

OSF 指令根据存储位的状态设置或清除输出位。

该指令在结构化文本和功能块中可以作为 OSFI 使用，请参见第 第 1 章 -22 页。

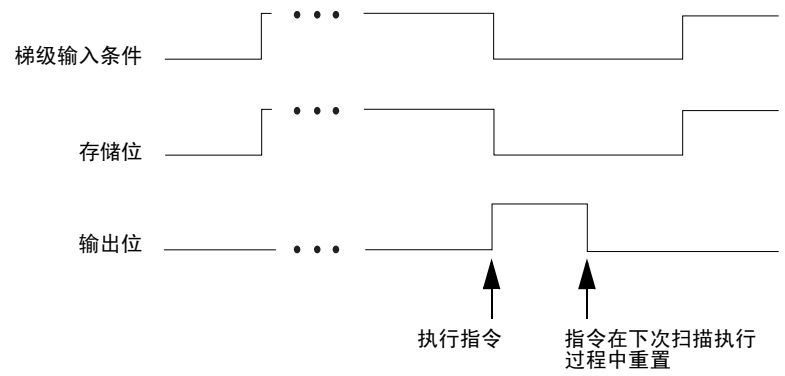
操作数：

梯形图操作数



操作数：	类型：	格式：	说明：
存储位	BOOL	标记	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	标记	要设置的位

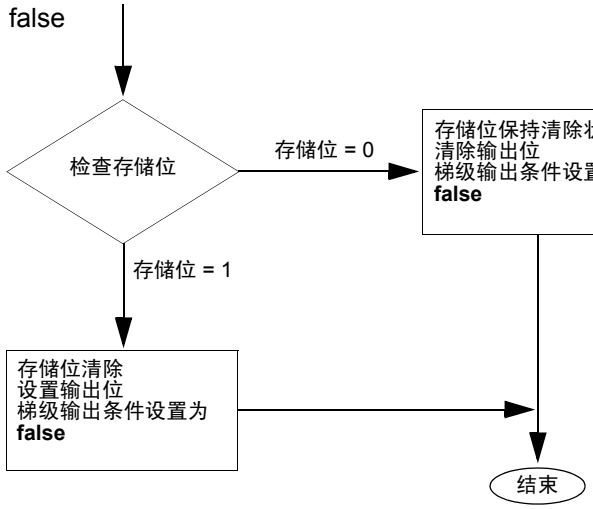
说明： 禁用时，如果设置了存储位， OSF 指令将设置输出位。当已禁用但存储位已清除时，或者当已启用时， OSF 指令将清除输出位。



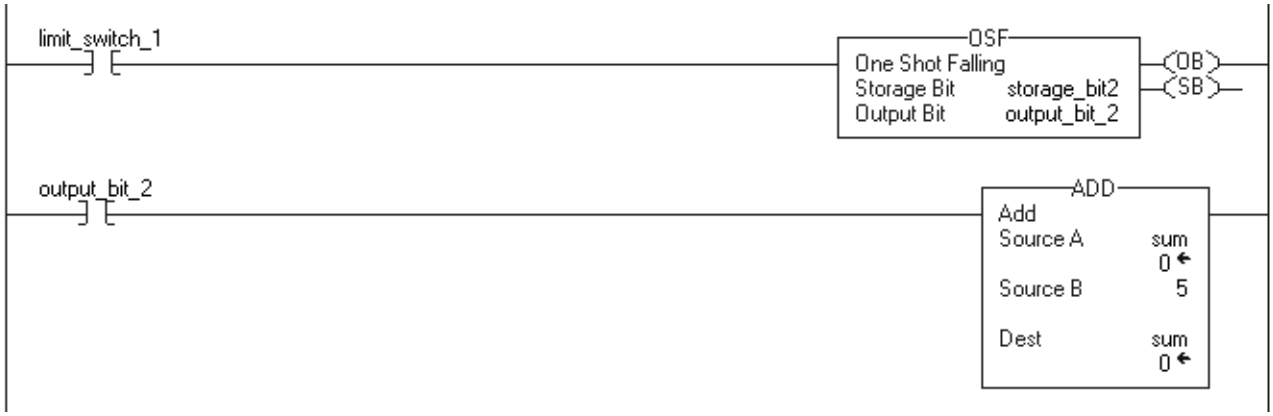
算术状态标志： 不受影响

错误条件： 无

执行：

条件：	梯形图操作：
预扫描	清除存储位以防止在第一次扫描期间出现无效触发。 清除输出位。 梯级输出条件设置为 false。
梯级输入条件为 false	
梯级输入条件为 true	设置存储位。 清除输出位。 梯级输出条件设置为 true。
后期扫描	请参见上面梯级输入条件为 false 的情况。

示例： 每当 *limit_switch_1* 从设置状态变为清除状态时， OSF 指令都设置 *output_bit_2*，并且 ADD 指令将 *sum* 加 5。只要 *limit_switch_1* 保持清除状态， *sum* 就保持相同的值不变。 *limit_switch_1* 必须再次从设置状态变为清除状态， *sum* 才能再次递增。可以在多个梯级上使用 *output_bit_2* 以触发其他操作。



带输入的上升沿单次触发 (One Shot Rising with Input, OSRI)

当输入位从清除状态切换为设置状态时， OSRI 指令为一个执行周期设置输出位。

该指令在梯形图中可以作为 OSR 使用，请参见第 第 1 章 -15 页。

操作数：



```
OSRI(OSRI_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
OSRI 标记	FBD_ONESHOT	结构	OSRI 结构





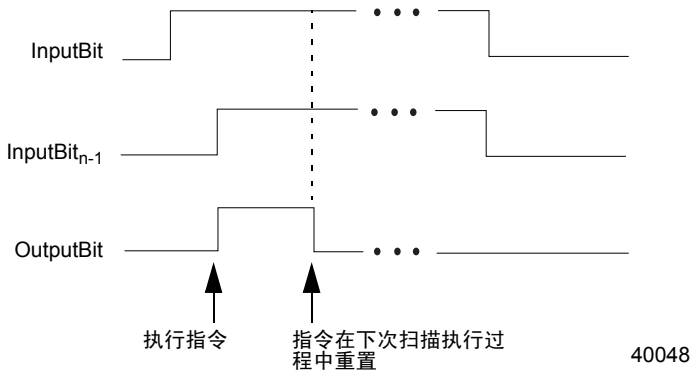
功能块

操作数：	类型：	格式：	说明：
OSRI 标记	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：** 如果为清除状态，则不执行指令，也不更新输出。 如果为设置状态，则执行指令。 默认为设置状态。 **结构化文本：** 无影响。指令执行。
InputBit	BOOL	输入位。这等效于梯形图 OSR 指令的梯级条件。 默认为清除状态。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
OutputBit	BOOL	输出位

说明： 当设置了 InputBit 但清除了 $InputBit_{n-1}$ 时，OSRI 指令设置 OutputBit。当设置了 $InputBit_{n-1}$ 或者当清除了 InputBit 时， OSRI 指令清除 OutputBit。



算术状态标志： 不受影响

错误条件： 无

执行：

条件：	功能块操作：	结构化文本操作：
预扫描	不采取任何操作。	不采取任何操作。
指令第一次扫描	设置 InputBit $_{n-1}$。	设置 InputBit $_{n-1}$。
指令第一次运行	设置 InputBit $_{n-1}$。	设置 InputBit $_{n-1}$。
EnableIn 处于清除状态	清除 EnableOut，指令不执行任何操作，也不更新输出。	不可用
EnableIn 处于设置状态	InputBit 由清除状态转换为设置状态时，该指令设置 InputBit $_{n-1}$。 指令执行。 设置 EnableOut。	InputBit 由清除状态转换为设置状态时，该指令设置 InputBit $_{n-1}$。 EnableIn 始终为设置状态。 指令执行。
后期扫描	不采取任何操作。	不采取任何操作。

示例： 当 *limit_switch1* 从清除状态变为设置状态时， OSRI 指令为一个扫描设置 OutputBit。

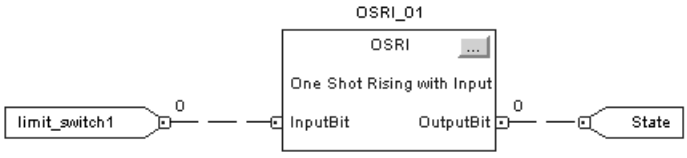
结构化文本

```
OSRI_01.InputBit := limit_switch1;

OSRI(OSRI_01);

State := OSRI_01.OutputBit;
```

功能块



带输入的下降沿单次触发 (One Shot Falling with Input, OSFI)

当 InputBit 从设置状态变为清除状态时， OSFI 指令为一个执行周期设置 OutputBit。

该指令在梯形图中可以作为 OSF 使用，请参见第 第 1 章 -17 页。

操作数：

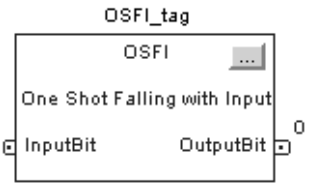


```
OSFI(OSFI_tag);
```

结构化文本

操作数：	类型：	格式：	说明：
OSFI 标记	FBD_ONESHOT	结构	OSFI 结构





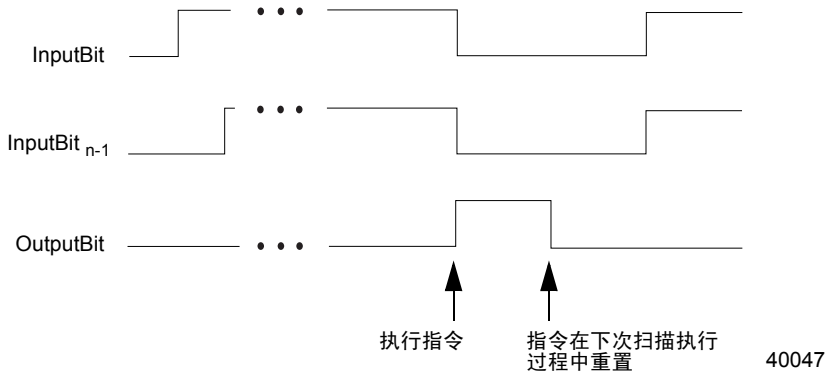
功能块

操作数：	类型：	格式：	说明：
OSFI 标记	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：** 如果为清除状态，则不执行指令，也不更新输出。 如果为设置状态，则执行指令。 默认为设置状态。 **结构化文本：** 无影响。指令执行。
InputBit	BOOL	输入位。这等效于梯形图 OSF 指令的梯级条件。 默认为清除状态。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
OutputBit	BOOL	输出位

说明： 当清除了 InputBit 但设置了 InputBit $_{n-1}$ 时，OSFI 指令设置 OutputBit。当清除了 InputBit $_{n-1}$ 或者设置了 InputBit 时， OSFI 指令清除 OutputBit。



算术状态标志： 不受影响

错误条件： 无

执行：

条件：	功能块操作：	结构化文本操作：
预扫描	不采取任何操作。	不采取任何操作。
指令第一次扫描	清除 InputBit $_{n-1}$。	清除 InputBit $_{n-1}$。
指令第一次运行	清除 InputBit $_{n-1}$。	清除 InputBit $_{n-1}$。
EnableIn 处于清除状态	清除 EnableOut，指令不执行任何操作，也不更新输出。	不可用
EnableIn 处于设置状态	InputBit 由清除状态转换为设置状态时，该指令清除 InputBit $_{n-1}$。 指令执行。 设置 EnableOut。	InputBit 由清除状态转换为设置状态时，该指令清除 InputBit $_{n-1}$。 EnableIn 始终为设置状态。 指令执行。
后期扫描	不采取任何操作。	不采取任何操作。

示例： 当 *limit_switch1* 从设置状态变为清除状态时，OSFI 指令为一个扫描设置 OutputBit。

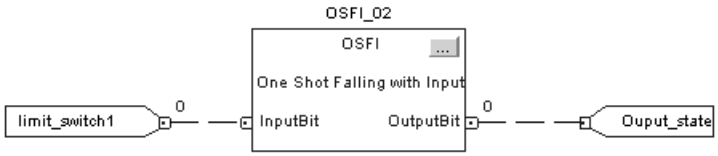
结构化文本

```
OSFI_01.InputBit := limit_switch1;

OSFI(OSFI_01);

Output_state := OSFI_01.OutputBit;
```

功能块



第 2 章

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事件的时间或数目控制操作。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
计时器启用后的时间长度	TON	梯形图	第 2 章 -2
计时器禁用后的时间长度	TOF	梯形图	第 2 章 -6
累计时间	RTO	梯形图	第 2 章 -10
功能块中带有内置复位的计时器启用后的时间长度	TONR	结构化文本 功能块	第 2 章 -15
功能块中带有内置复位的计时器禁用后的时间长度	TOFR	结构化文本 功能块	第 2 章 -19
功能块中带有内置复位的累计时间	RTOR	结构化文本 功能块	第 2 章 -23
递增	CTU	梯形图	第 2 章 -28
递减	CTD	梯形图	第 2 章 -32
功能块中的递增和递减	CTUD	结构化文本 功能块	第 2 章 -36
复位计时器或计数器	RES	梯形图	第 2 章 -40

所有计时器的时基为 1 毫秒。

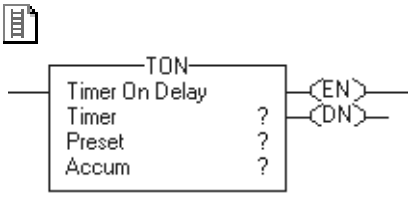
计时器打开延迟 (Timer On Delay, TON)

TON 指令是非保持计时器，它在启用指令 （梯级输入条件为 true）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，请参见第 第 2 章 -15 页。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Timer	TIMER	标记	计时器结构
Preset	DINT	立即数	延迟时间 （累计时间）
Accum	DINT	立即数	计时器已计算的总毫秒数 初始值通常为 0

TIMER 结构

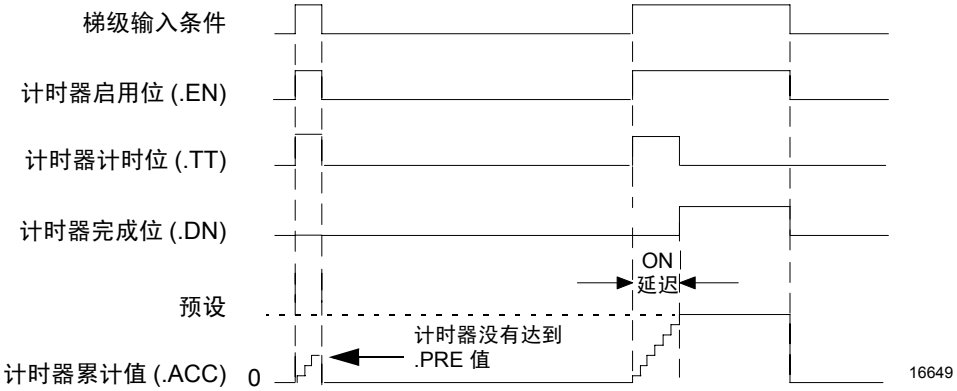
助记符：	数据类型：	说明：
.EN	BOOL	启用位指示已启用 TON 指令。
.TT	BOOL	计时位指示计时操作正在进行中。
.DN	BOOL	当 .ACC ≥ .PRE 时设置完成位。
.PRE	DINT	预设值指定累计值在该指令设置 .DN 位之前所必须达到的值 （以毫秒为单位）。
.ACC	DINT	累计值指定自从启用 TON 指令以来经过的毫秒数。

说明： TON 指令一直累计时间，直到：

- 禁用 TON 指令
- .ACC ≥ .PRE

时基始终为 1 毫秒。例如，对于 2 秒的计时器，应输入 2000 作为 .PRE 值。

当禁用 TON 指令时， .ACC 值被清除。



计时器的运行方式

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：

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

在更新 ACC 之后，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 值的最大范围限制是 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。若发生此情况，ACC 值将不正确。

当计时器运行时，如果您将其放置于以下位置，则请在 69 分钟内对其进行扫描：

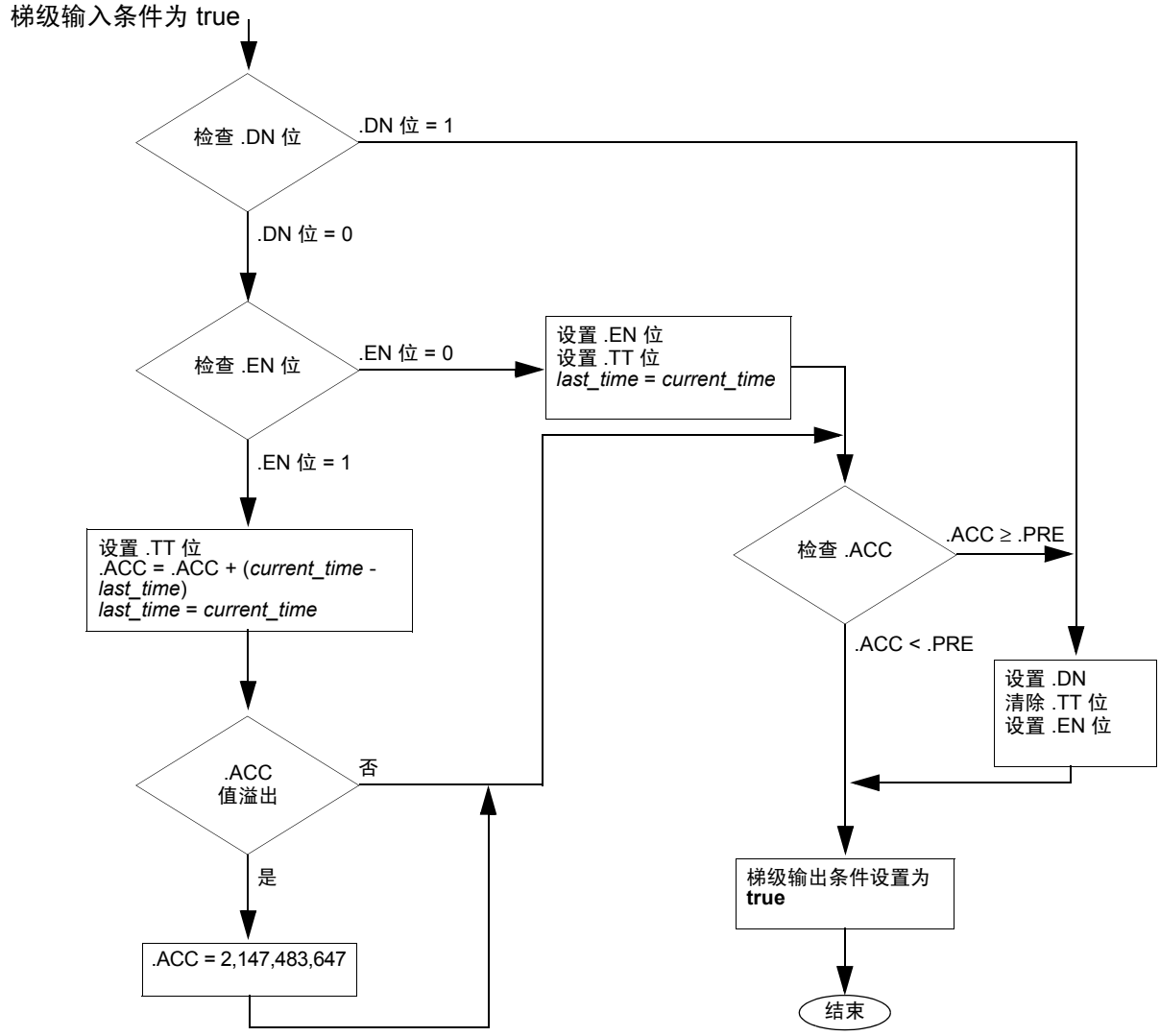
- 子例程中
- 位于 JMP 和 LBL 指令之间的代码段中
- 顺序流程图 (SFC) 中
- 事件或周期性任务中
- 阶段的状态例程中

算术状态标志： 不受影响

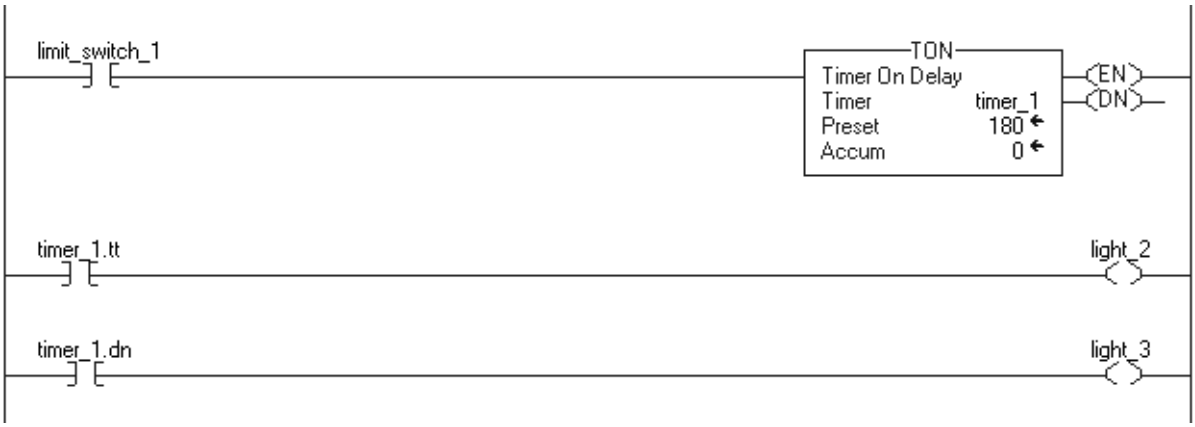
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
.PRE < 0	4	34
.ACC < 0	4	34

执行：

条件：	梯形图操作：
预扫描	清除 .EN、 .TT 和 .DN 位。 清除 .ACC 值。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .EN、 .TT 和 .DN 位。 清除 .ACC 值。 梯级输出条件设置为 false。
梯级输入条件为 true	
后扫描	梯级输出条件设置为 false。

示例： 当设置 *limit_switch_1* 时， *light_2* 打开 180 毫秒 （*timer_1* 正在计时）。当 *timer_1.acc* 达到 180 时， *light_2* 关闭， *light_3* 打开。*Light_3* 一直保持打开，直到禁用 TON 指令为止。如果在 *timer_1* 正在计时时清除了 *limit_switch_1*，则 *light_2* 关闭。



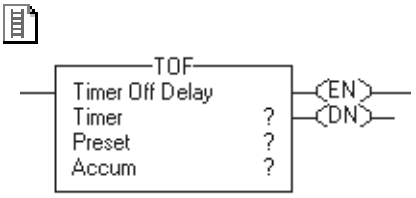
计时器关闭延迟 (Timer Off Delay, TOF)

TOF 指令是非保持计时器，它在启用指令 （梯级输入条件为 false）时累计时间。

该指令在结构化文本和功模块中以 TOFR 的形式提供，请参见第 第 2 章 -19 页。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Timer	TIMER	标记	计时器结构
Preset	DINT	立即数	延迟时间 （累计时间）
Accum	DINT	立即数	计时器已计算的总毫秒数 初始值通常为 0

TIMER 结构

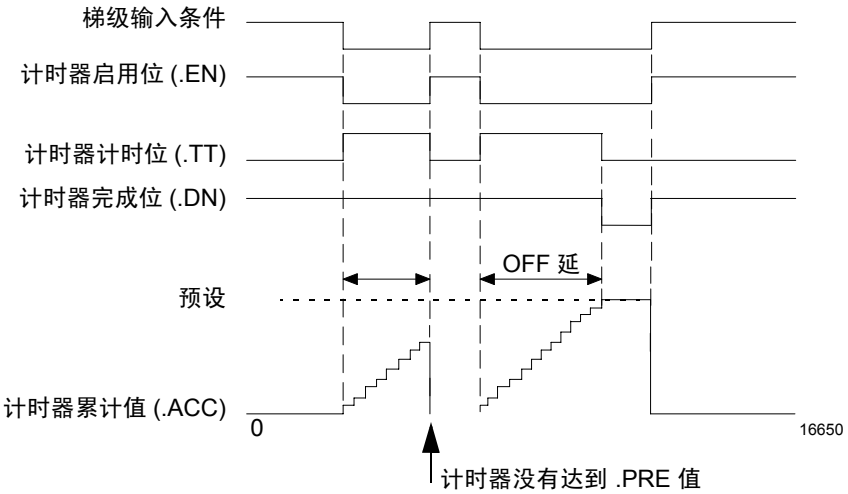
助记符：	数据类型：	说明：
.EN	BOOL	启用位指示已启用 TOF 指令。
.TT	BOOL	计时位指示计时操作正在进行中
.DN	BOOL	当 .ACC ≥ .PRE 时清除完成位。
.PRE	DINT	预设值指定累计值在该指令清除 .DN 位之前所必须达到的值 （以毫秒为单位）。
.ACC	DINT	累计值指定自从启用 TOF 指令以来经过的毫秒数。

说明： TOF 指令一直累计时间，直到：

- 禁用 TOF 指令
- .ACC ≥ .PRE

时基始终为 1 毫秒。例如，对于 2 秒的计时器，应输入 2000 作为 .PRE 值。

当禁用 TOF 指令时， .ACC 值被清除。



计时器的运行方式

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*)

在更新 ACC 之后，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 值的最大范围限制是 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。若发生此情况，ACC 值将不正确。

当计时器运行时，如果您将其放置于以下位置，则请在 69 分钟内对其进行扫描：

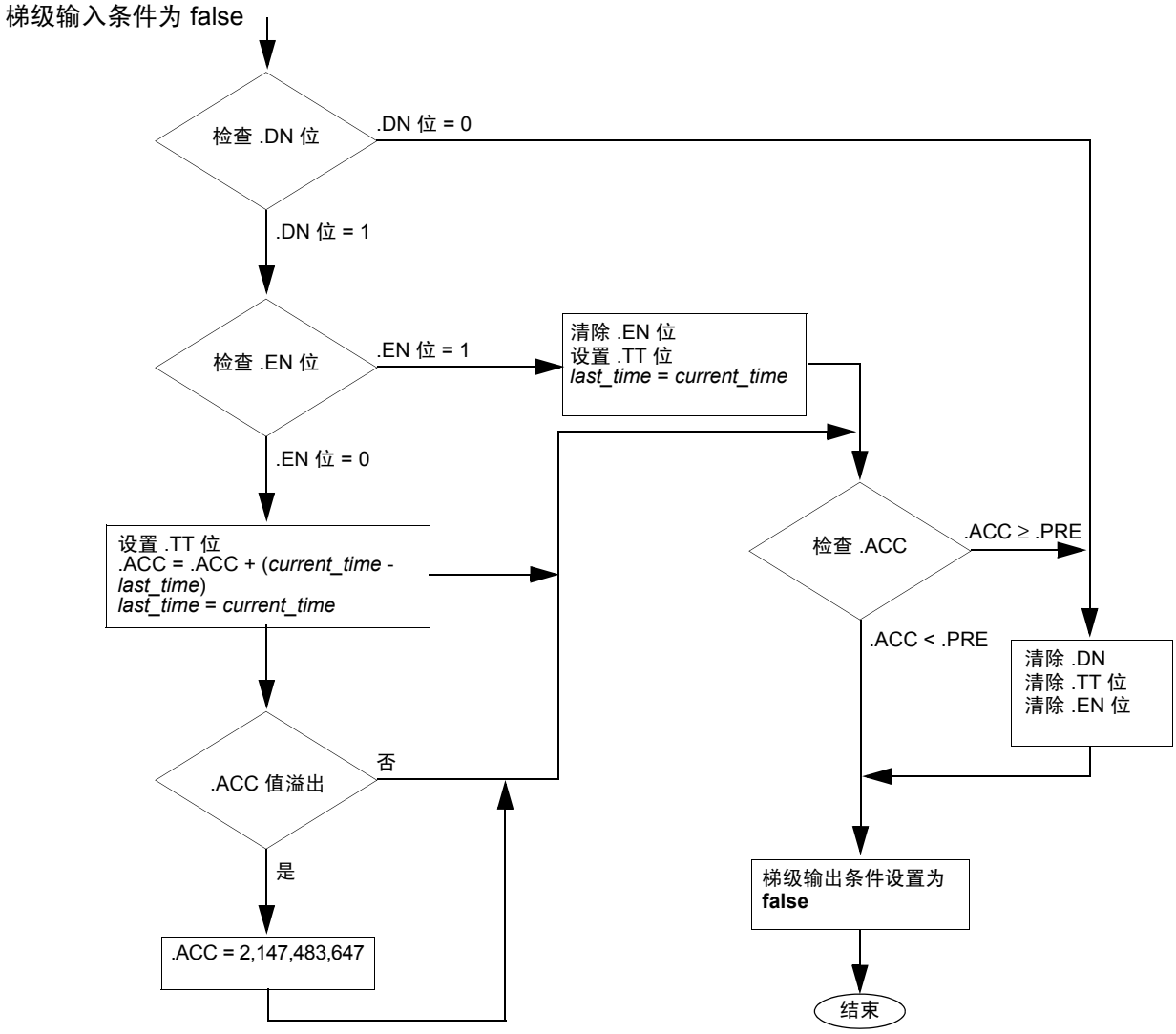
- 子例程中
- 位于 JMP 和 LBL 指令之间的代码段中
- 顺序流程图 (SFC) 中
- 事件或周期性任务中
- 阶段的状态例程中

算术状态标志： 不受影响

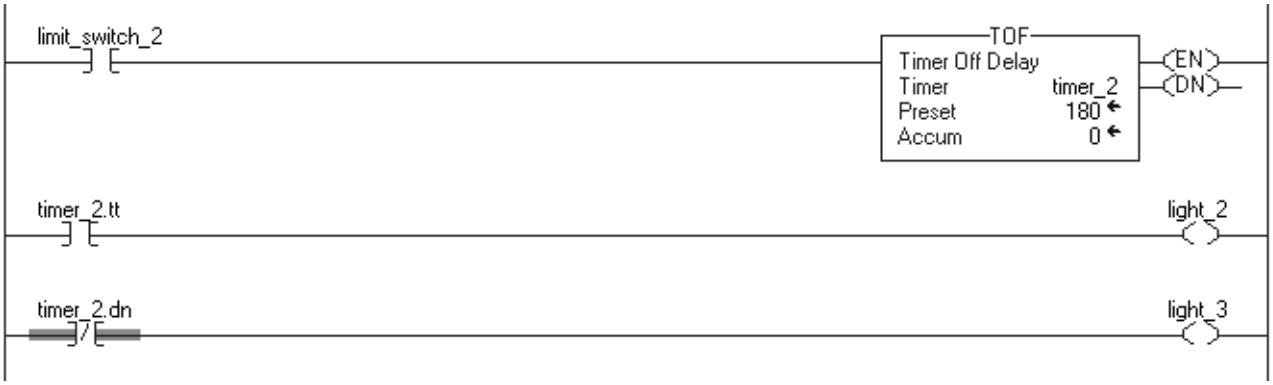
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
.PRE < 0	4	34
.ACC < 0	4	34

执行：

条件：	梯形图操作：
预扫描	清除 .EN、 .TT 和 .DN 位。 .ACC 值设置为等于 .PRE 值。 梯级输出条件设置为 false。
梯级输入条件为 false	
梯级输入条件为 true	设置 .EN、 .TT 和 .DN 位。 清除 .ACC 值。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

示例： 当清除 *limit_switch_2* 时， *light_2* 打开了 180 秒 （*timer_2* 正在计时）。当 *timer_2.acc* 达到 180 时， *light_2* 关闭， *light_3* 打开。 *Light_3* 一直保持打开，直到启用 TOF 指令为止。如果在 *timer_2* 正在计时时设置了 *limit_switch_2*，则 *light_2* 关闭。



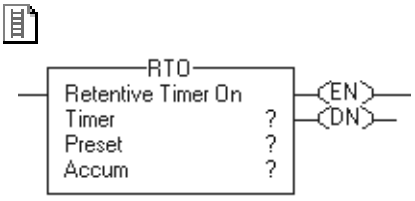
保持计时器打开 (Retentive Timer On, RTO)

RTO 指令是保持计时器，它在启用指令时累计时间。

该指令在结构化文本和功能块中以 RTOR 的形式提供，请参见第 第 2 章 -23 页。

操作数：

梯形图



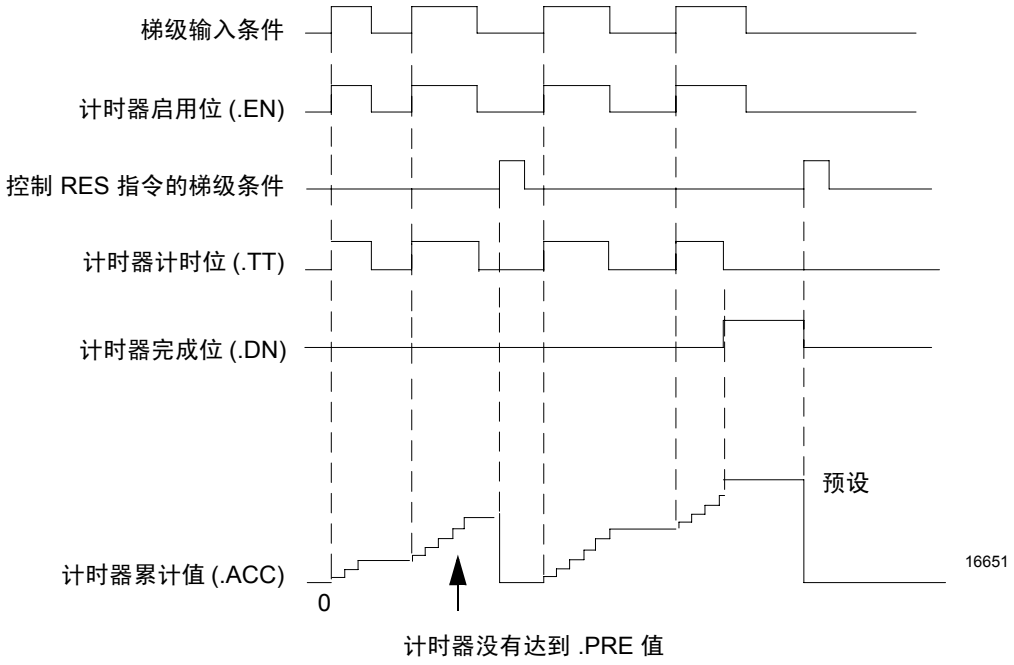
操作数：	类型：	格式：	说明：
Timer	TIMER	标记	计时器结构
Preset	DINT	立即数	延迟时间 （累计时间）
Accum	DINT	立即数	计时器已计算的毫秒数 初始值通常为 0

TIMER 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示已启用 RTO 指令。
.TT	BOOL	计时位指示计时操作正在进行中。
.DN	BOOL	完成位指示 .ACC ≥ .PRE。
.PRE	DINT	预设值指定累计值在该指令设置 .DN 位之前所必须达到的值 （以毫秒为单位）。
.ACC	DINT	累计值指定自从启用 RTO 指令以来经过的毫秒数。

说明： RTO 指令一直累计时间，直到它被禁用为止。当 RTO 指令被禁用时，它保持其 .ACC 值。您必须清除 .ACC 值，通常使用引用同一 TIMER 结构的 RES 指令。

时基始终为 1 毫秒。例如，对于 2 秒的计时器，应输入 2000 作为 .PRE 值。



计时器的运行方式

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：

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

在更新 ACC 之后，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 值的最大范围限制是 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。若发生此情况，ACC 值将不正确。

当计时器运行时，如果您将其放置于以下位置，则请在 69 分钟内对其进行扫描：

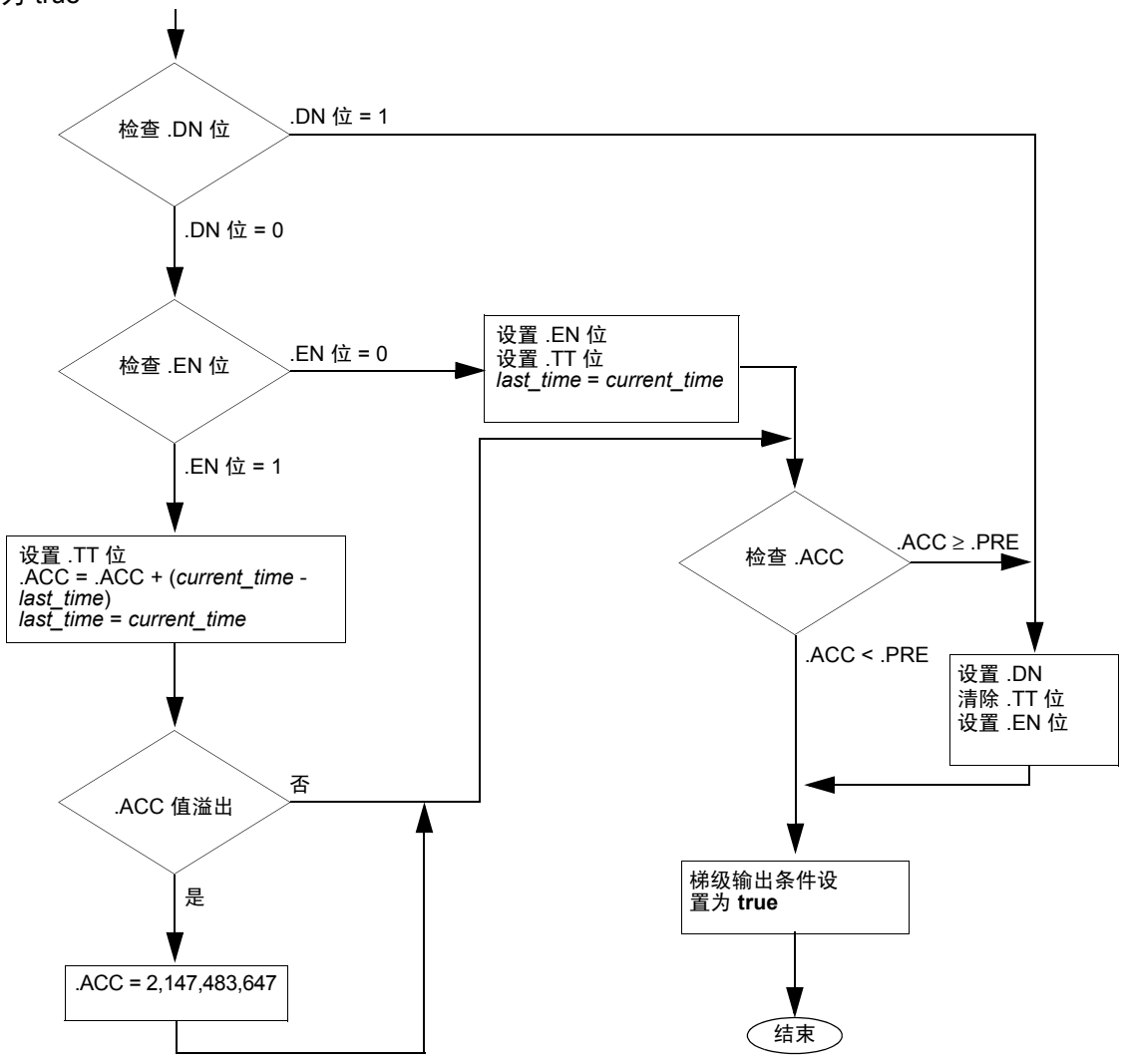
- 子例程中
- 位于 JMP 和 LBL 指令之间的代码段中
- 顺序流程图 (SFC) 中
- 事件或周期性任务中
- 阶段的状态例程中

算术状态标志： 不受影响

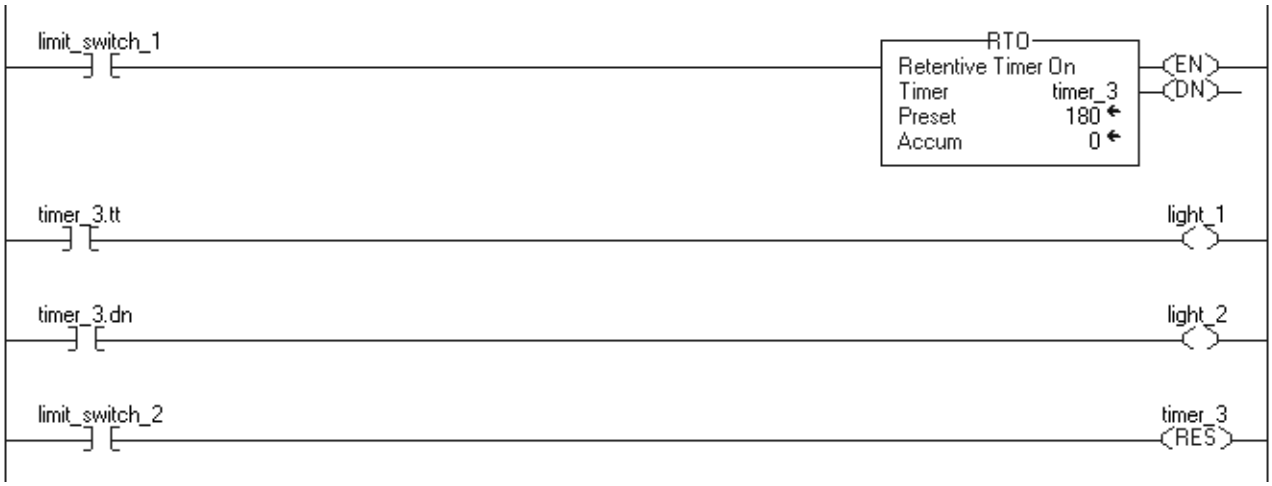
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
.PRE < 0	4	34
.ACC < 0	4	34

执行：

条件：	梯形图操作：
预扫描	清除 .EN、 .TT 和 .DN 位。 .ACC 值未修改。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .EN 和 .TT 位。 .DN 位未修改。 .ACC 值未修改。 梯级输出条件设置为 false。
梯级输入条件为 true	
后扫描	梯级输出条件设置为 false。

示例： 当设置 *limit_switch_1* 时， *light_1* 打开 180 毫秒（*timer_2* 正在计时）。当 *timer_3.acc* 达到 180 时， *light_1* 关闭， *light_2* 打开。 *Light_2* 一直保持，直到复位 *timer_3* 为止。如果在 *timer_3* 正在计时时清除了 *limit_switch_2*，则 *light_1* 保持打开。当设置 *limit_switch_2* 时，RES 指令复位 *timer_3* （清零状态位和 .ACC 值）。



带有复位的计时器打开延迟 (Timer On Delay with Reset, TONR)

TONR 指令是非保持计时器，它在设置 TimerEnable 时累计时间。

此指令在梯形图中以两个独立指令提供：TON （请参见第 第 2 章 -2 页）和 RES （请参见第 2-40 页）。

操作数：

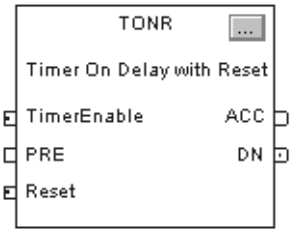


```
TONR(TONR_tag);
```

结构化文本

变量：	类型：	格式：	说明：
TONR 标记	FBD_TIMER	结构	TONR 结构





功能块

操作数：	类型：	格式：	说明：
TONR 标记	FBD_TIMER	结构	TONR 结构

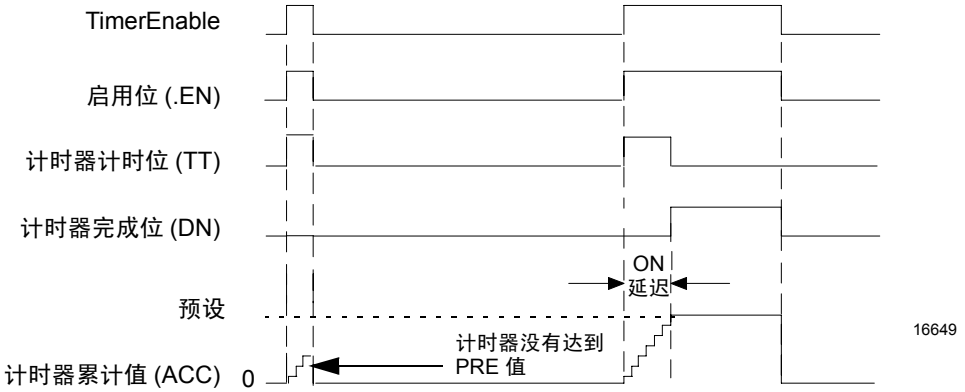
FBD_TIMER 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果为清零状态，则不执行指令，也不更新输出。 如果为置位状态，则执行指令。 默认为置位状态。 **结构化文本：** 无影响。指令执行。
TimerEnable	BOOL	如果设置，将启用计时器，使之运行并累计时间。 默认为清零状态。
PRE	DINT	计时器预设值。这是在计时完成前 ACC 必须达到的值 （以毫秒为单位）。 如果无效，该指令将设置 Status 中相应位，计时器不执行。 Valid = 0 表示设置为最大的正整数
Reset	BOOL	请求复位计时器。设置时，计时器复位。 默认为清零状态。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
ACC	BOOL	以毫秒表示的累计时间。
EN	BOOL	启用计时器的输出。指示计时器指令已启用。
TT	BOOL	计时器计时输出。设置时，表示计时器操作正在进行。
DN	BOOL	计时器完成输出。指示累计值何时大于或等于预设值。
状态	DINT	功能块状态。
InstructFault (Status.0)	BOOL	该指令检测到以下执行错误之一。这不是轻微或严重的控制器错误。检查其余状态位以确定发生的情况。
PresetInv (Status.1)	BOOL	预设值无效。

说明： TONR 指令一直累计时间，直到：

- 禁用 TONR 指令
- ACC ≥ PRE

时基始终为 1 毫秒。例如，对于 2 秒的计时器，应输入 2000 作为 PRE 值。



设置 Reset 输入参数可复位该指令。如果在设置 Reset 时设置了 TimerEnable，则 TONR 指令在清除 Reset 时再次开始计时。

计时器的运行方式

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：

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

在更新 ACC 之后，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 值的最大范围限制是 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。若发生此情况，ACC 值将不正确。

当计时器运行时，如果您将其放置于以下位置，则请在 69 分钟内对其进行扫描：

- 子例程中
- 位于 JMP 和 LBL 指令之间的代码段中
- 顺序流程图 (SFC) 中
- 事件或周期性任务中
- 阶段的状态例程中

算术状态标志： 不受影响

错误条件： 无

执行：

条件：	功能块操作：	结构化文本操作：
预扫描	不采取任何操作。	不采取任何操作。
指令第一次扫描	清除 EN、 TT 和 DN。 将 ACC 值设置为 0。	清除 EN、 TT 和 DN。 将 ACC 值设置为 0。
指令第一次运行	清除 EN、 TT 和 DN。 将 ACC 值设置为 0。	清除 EN、 TT 和 DN。 将 ACC 值设置为 0。
EnableIn 处于清零状态	清除 EnableOut，指令不执行任何操作， 也不更新输出。	不可用
EnableIn 处于置位状态	当 EnableIn 从清除转为设置时，该指令按 指令初次扫描的方式进行初始化。 指令执行。 设置 EnableOut。	EnableIn 始终为置位状态。 指令执行。
复位	设置 Reset 输入参数时，该指令清除 EN、 TT 和 DN 并将 ACC 设置为零。	设置 Reset 输入参数时，该指令清除 EN、 TT 和 DN 并将 ACC 设置为零。
后扫描	不采取任何操作。	不采取任何操作。

示例： 在设置了 *limit_switch1* 的每次扫描中， TONR 指令按经过的时间递增 ACC 值，直到 ACC 值达到 PRE 值。当 ACC ≥ PRE 时，设置 DN 参数，并设置 *timer_state*。

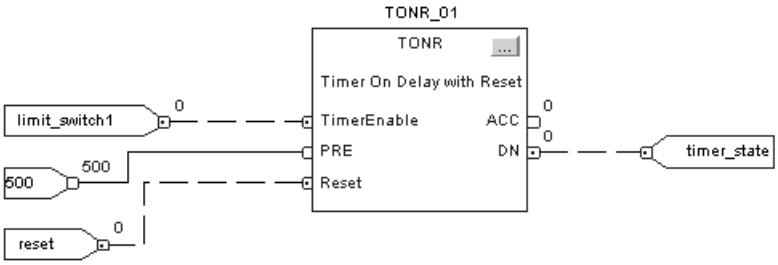
结构化文本

```
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


TONR_01
TONR
...
Timer On Delay with Reset
limit_switch1
0
TimerEnable
ACC
0
PRE
DN
0
timer_state
500
500
Reset
reset
0

带有复位的计时器关闭延迟 (Timer Off Delay with Reset, TOFR)

TOFR 指令是非保持计时器，它在清除 TimerEnable 时累计时间。

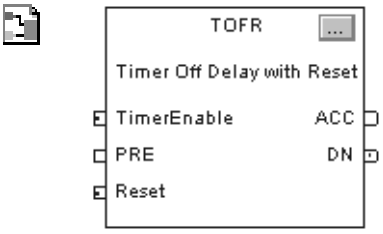
此指令在梯形图中以两个独立指令提供：TOF （请参见第 第 2 章 -6 页）和 RES （请参见 2-40 页）。

操作数：

```
TOFR(TOFR_tag);
```

结构化文本

变量：	类型：	格式：	说明：
TOFR 标记	FBD_TIMER	结构	TOFR 结构



功能块操作数

操作数：	类型：	格式：	说明：
TOFR 标记	FBD_TIMER	结构	TOFR 结构

FBD_TIMER 结构

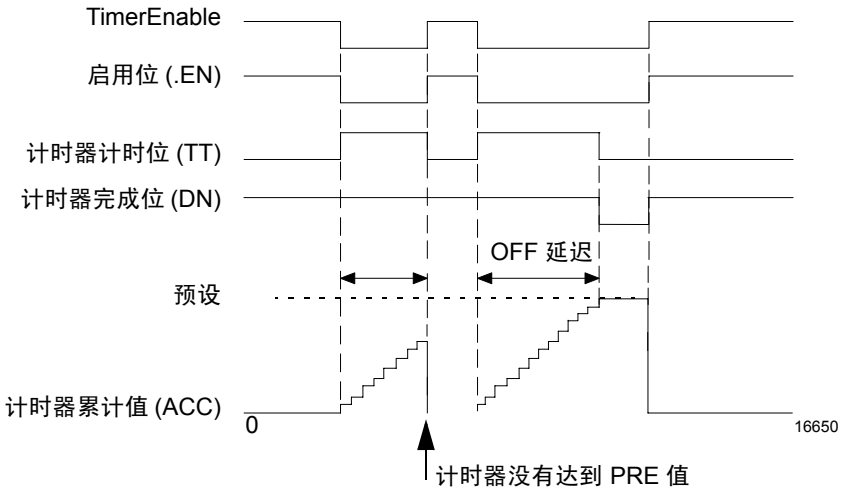
输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果为清零状态，则不执行指令，也不更新输出。 如果为置位状态，则执行指令。 默认为置位状态。 **结构化文本：** 无影响。指令执行。
TimerEnable	BOOL	如果为清零状态，则会使计时器运行并累计时间 默认为清零状态。
PRE	DINT	计时器预设值。这是在计时完成前 ACC 必须达到的值 （以毫秒为单位）。 如果该值无效，则指令在 Status 中设置相应的位，计时器不执行。 Valid = 0 表示设置为最大的正整数
Reset	BOOL	请求复位计时器。设置时，计时器复位。 默认为清零状态。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
ACC	BOOL	以毫秒表示的累计时间。
EN	BOOL	启用计时器的输出。指示计时器指令已启用。
TT	BOOL	计时器计时输出。设置时，表示计时器操作正在进行。
DN	BOOL	计时器完成输出。指示累计值何时大于或等于预设值。

输入参数：	数据类型：	说明：
状态	DINT	功能块状态。
InstructFault (Status.0)	BOOL	该指令检测到以下执行错误之一。这不是轻微或严重的控制器错误。检查其余状态位以确定发生的情况。
PresetInv (Status.1)	BOOL	预设值无效。

说明： TOFR 指令一直累计时间，直到：

- 禁用 TOFR 指令
- ACC ≥ PRE

时基始终为 1 毫秒。例如，对于 2 秒的计时器，应输入 2000 作为 PRE 值。



设置 Reset 输入参数可复位该指令。如果在设置 Reset 时清除了 TimerEnable，则 TOFR 指令在清除 Reset 时不再次开始计时。

计时器的运行方式

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：

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

在更新 ACC 之后，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 值的最大范围限制是 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。若发生此情况，ACC 值将不正确。

当计时器运行时，如果您将其放置于以下位置，则请在 69 分钟内对其进行扫描：

- 子例程中
- 位于 JMP 和 LBL 指令之间的代码段中
- 顺序流程图 (SFC) 中
- 事件或周期性任务中
- 阶段的状态例程中

算术状态标志： 不受影响

错误条件： 无

执行：

条件：	功能块操作：	结构化文本操作：
预扫描	不采取任何操作。	不采取任何操作。
指令第一次扫描	清除 EN、 TT 和 DN。 将 ACC 值设置为 PRE。	清除 EN、 TT 和 DN。 将 ACC 值设置为 PRE。
指令第一次运行	清除 EN、 TT 和 DN。 将 ACC 值设置为 PRE。	清除 EN、 TT 和 DN。 将 ACC 值设置为 PRE。
EnableIn 处于清零状态	清除 EnableOut，指令不执行任何操作，也不更新输出。	不可用
EnableIn 处于置位状态	当 EnableIn 从清除转为设置时，该指令按指令初次扫描的方式进行初始化。 指令执行。 设置 EnableOut。	EnableIn 始终为置位状态。 指令执行。
复位	设置 Reset 输入参数时，该指令清除 EN、TT 和 DN 并将 ACC 设置为 PRE。注意，这与对 TOF 指令使用 RES 指令不同。	设置 Reset 输入参数时，该指令清除 EN、TT 和 DN 并将 ACC 设置为 PRE。注意，这与对 TOF 指令使用 RES 指令不同。
后扫描	不采取任何操作。	不采取任何操作。

示例： 在清除 *limit_switch1* 后的每次扫描中， TOFR 指令按经过的时间递增 ACC 值，直到 ACC 值达到 PRE 值。当 ACC ≥ PRE 时，清除 DN 参数，并设置 *timer_state2*。

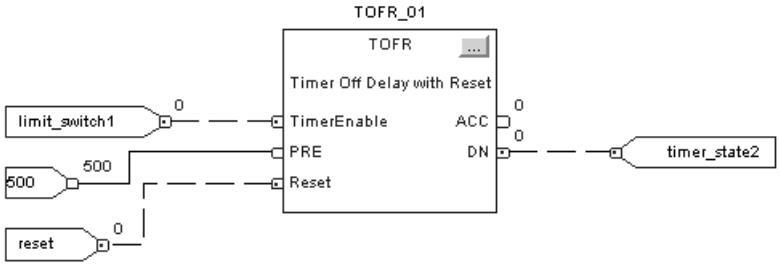
结构化文本

```
TOFR_01.Preset := 500
TOFR_01.Reset := reset;
TOFR_O1.TimerEnable := limit_switch1;

TOFR(TOFR_01);

timer_state2 := TOFR_01.DN;
```

功能块



带有复位的保持计时器打开 (Retentive Timer On with Reset, RTOR)

RTOR 指令是保持计时器，它在设置 TimerEnable 时累计时间。

此指令在梯形图中以两个独立指令提供：RTO （请参见第 第 2 章 -10 页）和 RES （请参见第 2-40 页）。

操作数：

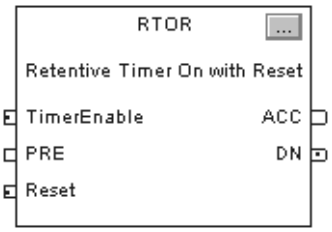


```
RTOR(RTOR_tag);
```

结构化文本

变量：	类型：	格式：	说明：
RTOR 标记	FBD_TIMER	结构	RTOR 结构





功能块操作数

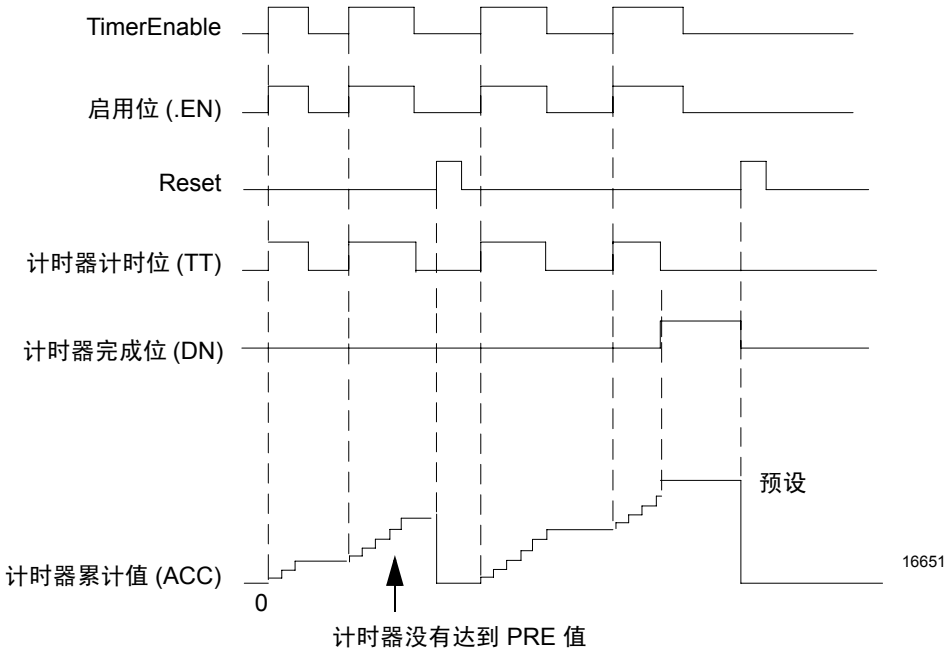
操作数：	类型：	格式：	说明：
RTOR 标记	FBD_TIMER	结构	RTOR 结构

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，则不执行指令，也不更新输出。 如果为置位状态，则执行指令。 默认为置位状态。 **结构化文本：** 无影响。指令执行。
TimerEnable	BOOL	如果设置，将启用计时器，使之运行并累计时间。 默认为清零状态。
PRE	DINT	计时器预设值。这是在计时完成前 ACC 必须达到的值 （以毫秒为单位）。 如果无效，该指令将设置 Status 中相应位，计时器不执行。 Valid = 0 表示设置为最大的正整数
Reset	BOOL	请求复位计时器。设置时，计时器复位。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
ACC	DINT	以毫秒表示的累计时间。即使在清除了 TimerEnable 输入的情况下，该值也保持。这导致了该块的行为不同于 TONR 块。
EN	BOOL	启用计时器的输出。指示计时器指令已启用。
TT	BOOL	计时器计时输出。设置时，表示计时器操作正在进行。
DN	BOOL	计时器完成输出。指示累计值何时大于或等于预设值。
状态	DINT	功能块状态。
InstructFault (Status.0)	BOOL	该指令检测到以下执行错误之一。这不是轻微或严重的控制器错误。检查其余状态位以确定发生的情况。
PresetInv (Status.1)	BOOL	预设值无效。

说明： RTOR 指令一直累计时间，直到它被禁用为止。当 RTOR 指令被禁用时，它保持其 ACC 值。您必须使用 Reset 输入清除 .ACC 值。

时基始终为 1 毫秒。例如，对于 2 秒的计时器，应输入 2000 作为 PRE 值。



设置 Reset 输入参数可复位该指令。如果在设置 Reset 时设置了 TimerEnable，则 RTOR 指令在清除 Reset 时再次开始计时。

计时器的运行方式

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：

$$ACC = ACC + (current\_time - last\_time\_scanned)$$

在更新 ACC 之后，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 值的最大范围限制是 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。若发生此情况，ACC 值将不正确。

当计时器运行时，如果您将其放置于以下位置，则请在 69 分钟内对其进行扫描：

- 子例程中
- 位于 JMP 和 LBL 指令之间的代码段中
- 顺序流程图 (SFC) 中
- 事件或周期性任务中
- 阶段的状态例程中

算术状态标志： 不受影响

错误条件： 无

执行：

条件：	功能块操作：	结构化文本操作：
预扫描	不采取任何操作。	不采取任何操作。
指令第一次扫描	清除 EN、 TT 和 DN ACC 值未修改	清除 EN、 TT 和 DN ACC 值未修改
指令第一次运行	清除 EN、 TT 和 DN ACC 值未修改	清除 EN、 TT 和 DN ACC 值未修改
EnableIn 处于清零状态	清除 EnableOut，指令不执行任何操作， 也不更新输出。	不可用
EnableIn 处于置位状态	**功能块：** 当 EnableIn 从清除转为置位时，该指令按指令初次扫描的方式进行初始化。 指令执行。 设置 EnableOut。	EnableIn 始终为置位状态。 指令执行。
复位	设置 Reset 输入参数时，该指令清除 EN、TT 和 DN 并将 ACC 设置为零。	设置 Reset 输入参数时，该指令清除 EN、TT 和 DN 并将 ACC 设置为零。
后扫描	不采取任何操作。	不采取任何操作。

示例： 在设置了 *limit_switch1* 的每次扫描中， RTOR 指令按经过的时间递增 ACC 值，直到 ACC 值达到 PRE 值。当 ACC ≥ PRE 时，设置 DN 参数，并设置 *timer_state3*。

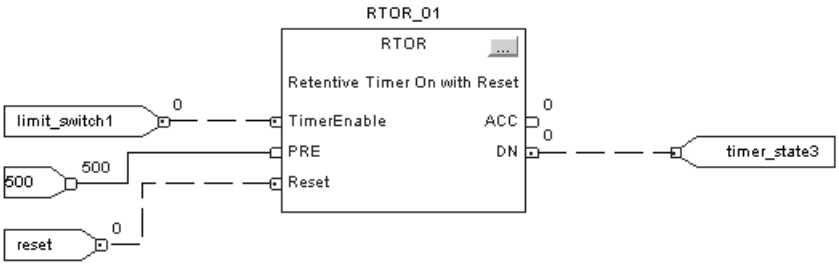
结构化文本

```
RTOR_01.Preset := 500
RTOR_01.Reset := reset;
RTOR_O1.TimerEnable := limit_switch1;

RTOR(RTOR_01);

timer_state3 := RTOR_01.DN;
```

功能块



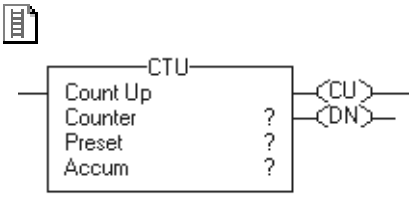
递增 (Count Up, CTU)

CTU 指令执行递增操作。

该指令在结构化文本和功能块中以 CTUD 的形式提供，请参见第 第 2 章 -36 页。

操作数：

梯形图

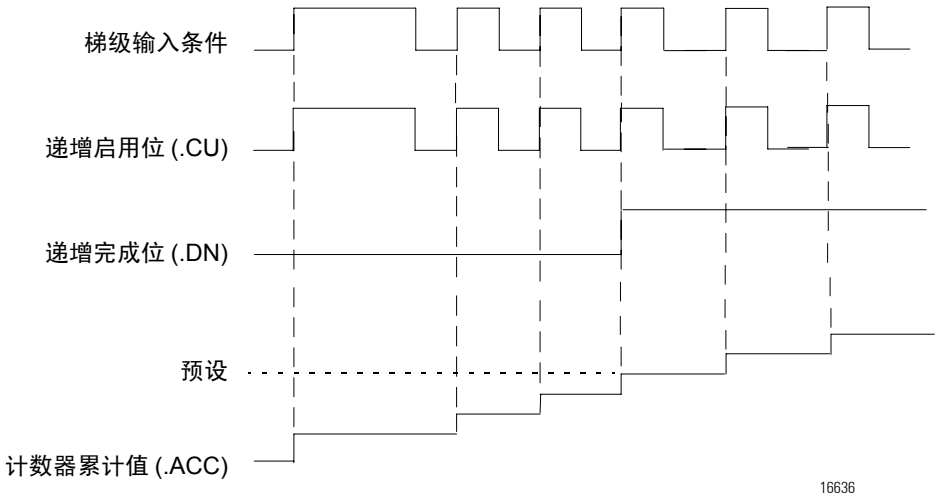


操作数：	类型：	格式：	说明：
Counter	COUNTER	标记	计数器结构
Preset	DINT	立即数	计数的上限
Accum	DINT	立即数	计数器已计算的次数 初始值通常为 0

COUNTER 结构

助记符：	数据类型：	说明：
.CU	BOOL	递增启用位指示 CTU 指令已启用。
.DN	BOOL	完成位指示 .ACC ≥ .PRE。
.OV	BOOL	上溢出位指示计数器已超出上限 2,147,483,647。之后，计数器溢出为 –2,147,483,648 并再次开始递增。
.UN	BOOL	下溢出位指示计数器已超出下限 –2,147,483,648。之后，计数器溢出为 2,147,483,647 并再次开始递减。
.PRE	DINT	预设值指定累计值在该指令设置 .DN 位之前所必须达到的值。
.ACC	DINT	累计值指定该指令已经计算的转换数。

说明： 当启用 CTU 指令并清除 .CU 位时， CTU 指令按 1 递增计数器。当启用 CTU 指令并设置 .CU 位时，或者当禁用 CTU 指令时，CTU 指令保持其 .ACC 值。



即使在设置 .DN 位后，累计值也继续递增。要清除累计值，请使用引用计数器结构的 RES 指令，或者将 0 写入累计值。

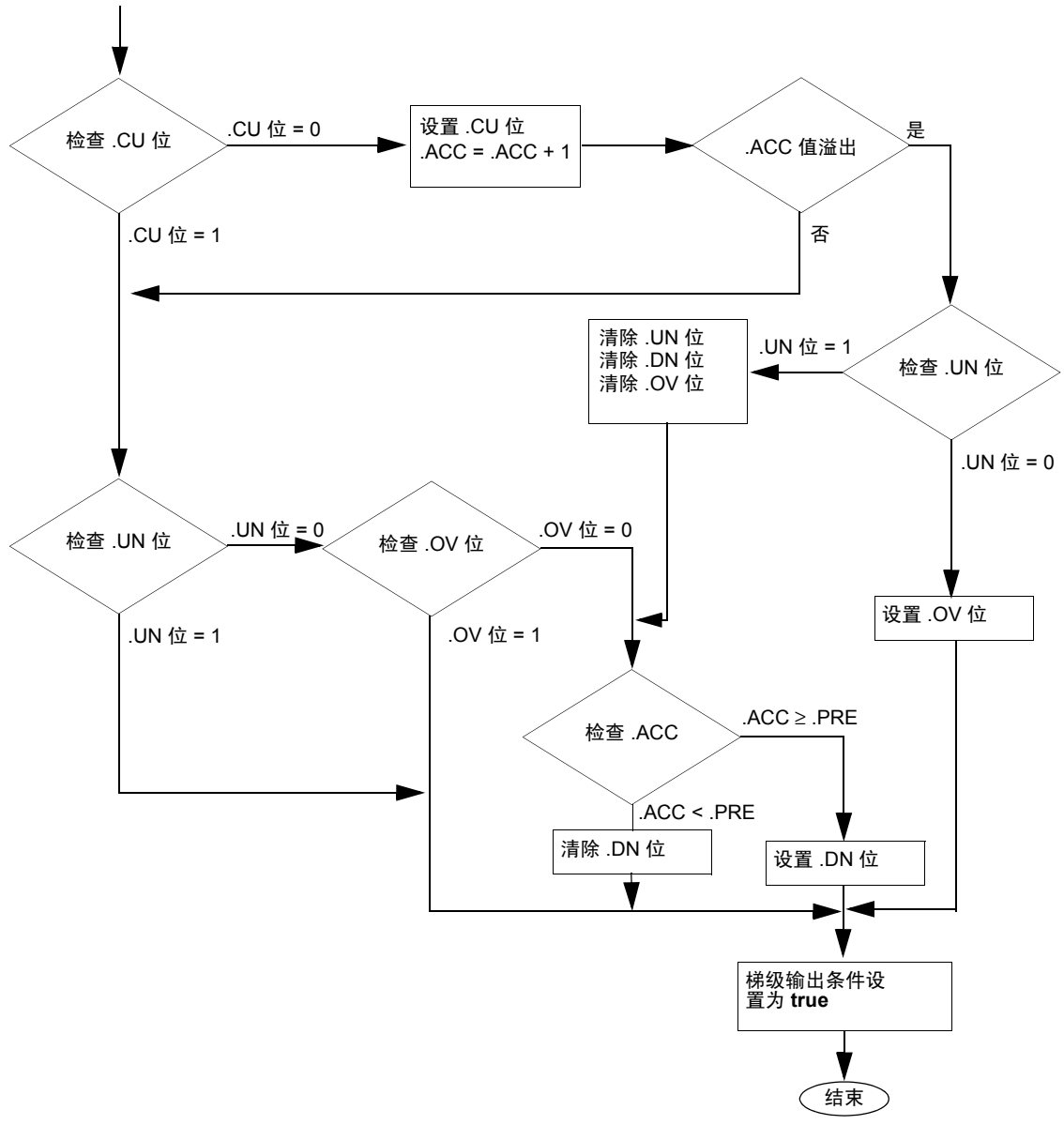
算术状态标志： 不受影响

错误条件： 无

执行：

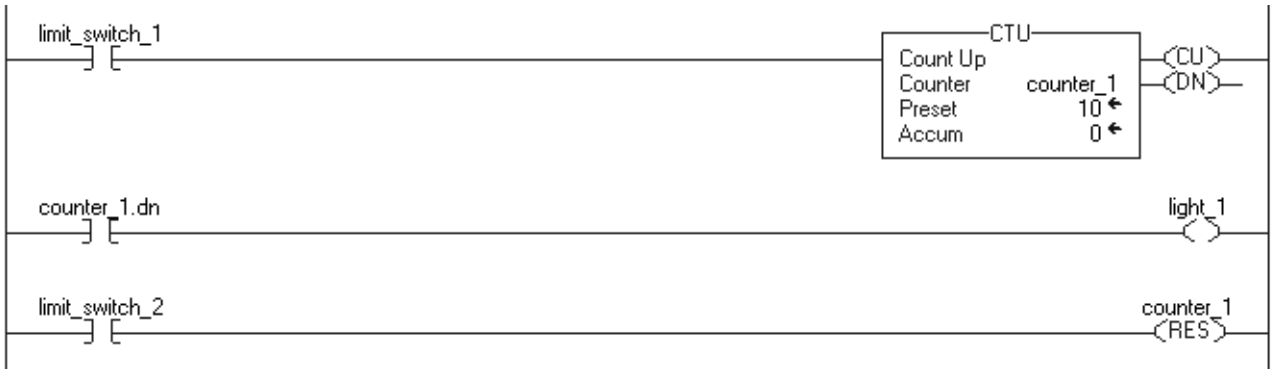
条件：	梯形图操作：
预扫描	设置 .CU 位，防止第一次程序扫描期间的无效递增。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .CU 位。 梯级输出条件设置为 false。

梯级输入条件为 true



后扫描	梯级输出条件设置为 false。

示例： *limit_switch_1* 从禁用状态转为启用状态 10 次后，设置 .DN 位，且 *light_1* 打开。如果 *limit_switch_1* 继续从禁用状态转为启用状态， *counter_1* 将继续递增其计数， .DN 位保持置位状态。当启用 limit_switch_2 时，RES 指令复位 *counter_1* （清零状态位和 .ACC 值）， *light_1* 关闭。



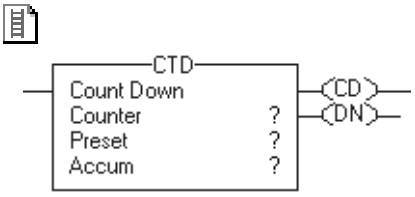
递减 (Count Down, CTD)

CTD 指令执行递减操作。

该指令在结构化文本和功能块中以 CTUD 的形式提供，请参见第 第 2 章 -36 页。

操作数：

梯形图



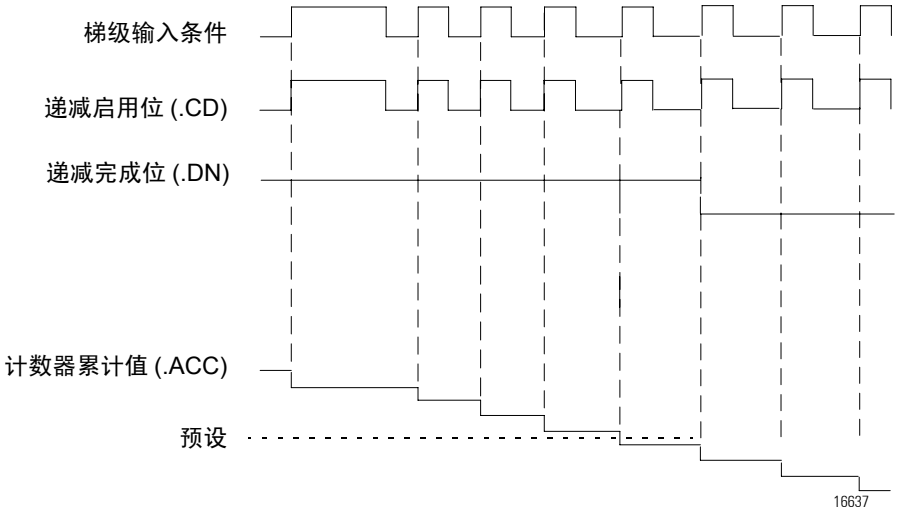
操作数：	类型：	格式：	说明：
Counter	COUNTER	标记	计数器结构
Preset	DINT	立即数	计数的下限
Accum	DINT	立即数	计数器已计算的次数 初始值通常为 0

COUNTER 结构

助记符：	数据类型：	说明：
.CD	BOOL	递减启用位指示 CTD 指令已启用。
.DN	BOOL	完成位指示 .ACC ≥ .PRE。
.OV	BOOL	上溢出位指示计数器已超出上限 2,147,483,647。之后，计数器溢出为 –2,147,483,648 并再次开始递增。
.UN	BOOL	下溢出位指示计数器已超出下限 –2,147,483,648。之后，计数器溢出为 2,147,483,647 并再次开始递减。
.PRE	DINT	预设值指定累计值在该指令设置 .DN 位之前所必须达到的值。
.ACC	DINT	累计值指定该指令已经计算的转换数。

说明： CTD 指令通常与引用同一计数器结构的 CTU 指令一起使用。

当启用 CTD 指令并清除 .CD 位时， CTD 指令按 1 递减计数器。当启用 CTD 指令并设置 .CD 位时，或者当禁用 CTD 指令时，CTD 指令保持其 .ACC 值。



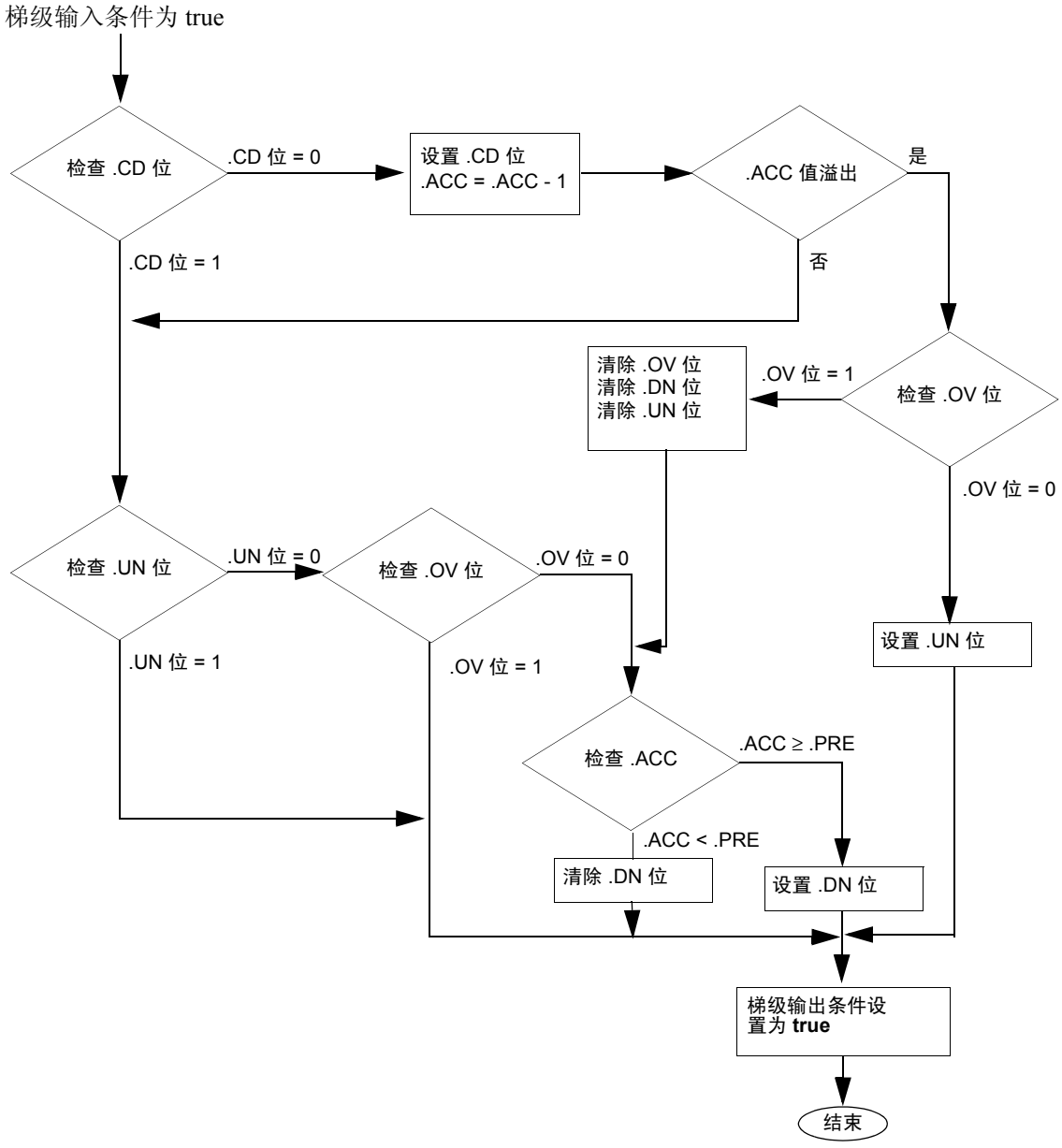
即使设置 .DN 位之后，累计值也继续递减。要清除累计值，请使用引用计数器结构的 RES 指令，或者将 0 写入累计值。

算术状态标志： 不受影响

错误条件： 无

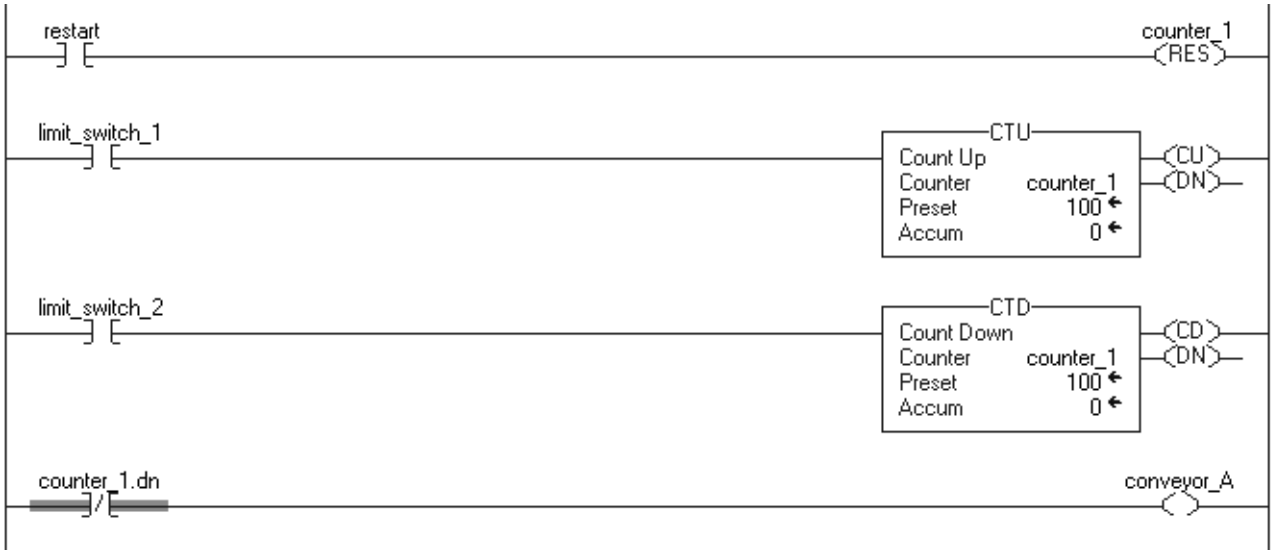
执行：

条件：	梯形图操作：
预扫描	设置 .CD 位，防止第一次程序扫描期间的无效递增。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .CD 位。 梯级输出条件设置为 false。



后扫描	梯级输出条件设置为 false。

示例： 传送带将零件传入缓冲区。每当有一个零件进入时，都启用 *limit_switch_1* 且 *counter_1* 递增 1。每当有一个零件离开时，都启用 *limit_switch_2* 且 *counter_1* 递减 1。如果在缓冲区中有 100 个零件（设置了 *counter_1.dn*），则 *conveyor_a* 打开，并阻止传送带传入更多零件，直到缓冲区有空间可以存放更多零件为止。



递增 / 递减 (CTUD)

CTUD 指令在 CUEnable 从清零状态转为置位状态时按 1 递增。该指令在 CDEnable 从清零状态转为置位状态时按 1 递减。

此指令在梯形图中以三个独立指令提供：CTU （请参见第 第 2 章 -28 页）、 CTD （请参见第 2-32 页）和 RES （请参见第 第 2 章 -40 页）。

操作数：

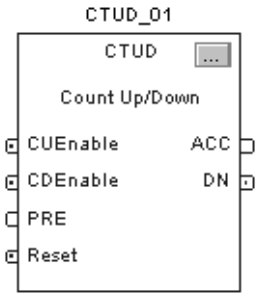


```
CTUD(CTUD_tag);
```

结构化文本

变量：	类型：	格式：	说明：
CTUD 标记	FBD_COUNTER	结构	CTUD 结构





功能块

操作数：	类型：	格式：	说明：
CTUD 标记	FBD_COUNTER	结构	CTUD 结构

FBD_COUNTER 结构

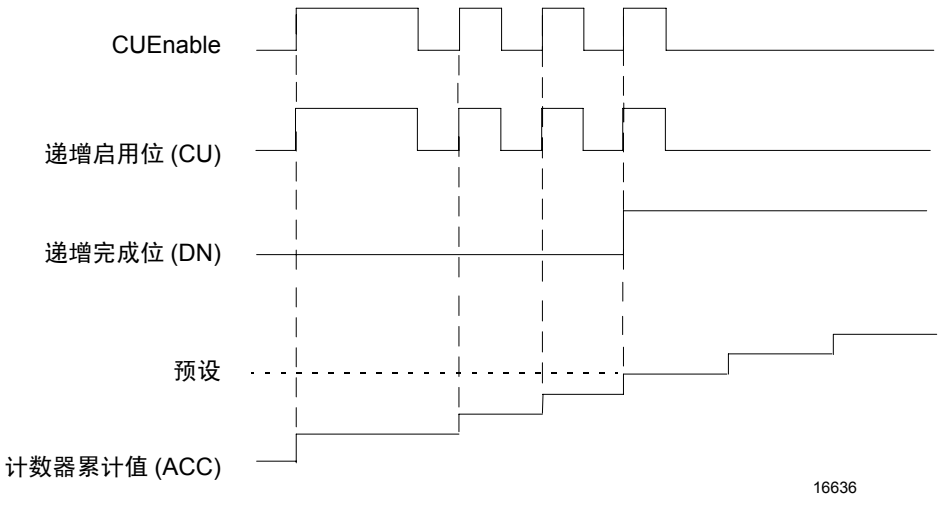
输入参数：	数据类型：	说明：
EnableIn	BOOL	**功能块：** 如果为清零状态，则不执行指令，也不更新输出。 如果为置位状态，则执行指令。 默认为置位状态。 **结构化文本：** 无影响。指令执行。
CUEnable	BOOL	启用递增。当输入从清零状态切换为置位状态时，累加器按 1 递增。 默认为清零状态。
CDEnable	BOOL	启用递减。当输入从清零状态切换为置位状态时，累加器按 1 递减。 默认为清零状态。
PRE	DINT	计数器预设值。这是设置 DN 之前累计值必须达到的值。 任何整数都有效 默认值为 0。
Reset	BOOL	请求复位计时器。当设置时，计数器复位。 默认为清零状态。

输出参数：	数据类型：	说明：
EnableOut	BOOL	指令产生有效结果。
ACC	DINT	累计值。
CU	BOOL	启用递增。
CD	BOOL	启用递减。
DN	BOOL	计数完成。当累计值大于或等于预设值时设置。
OV	BOOL	计数器上溢出。指示计数器已超出上限 2,147,483,647。 之后，计数器溢出为 –2,147,483,648 并再次开始递增。
UN	BOOL	计数器下溢出。指示计数器已超出下限 –2,147,483,648。 之后，计数器溢出为 2,147,483,647 并再次开始递减。

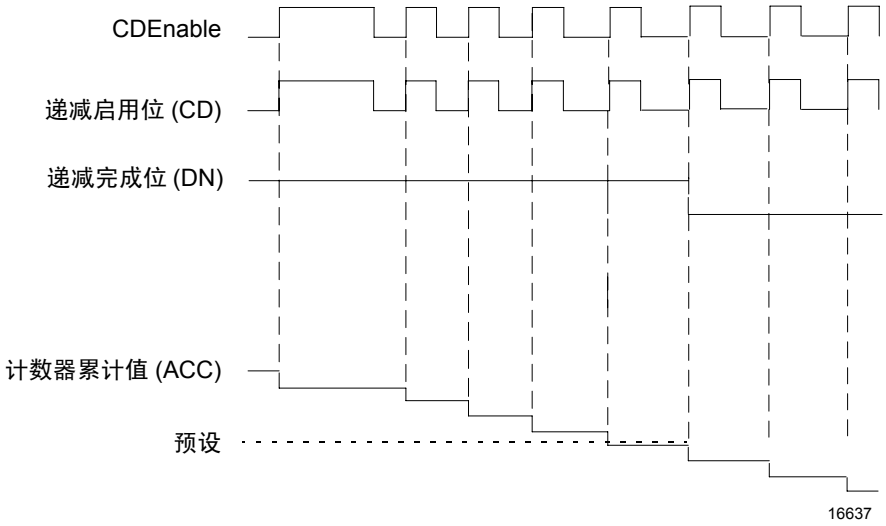
说明 当启用 CTUD 指令并设置 CUEnable 时， CTUD 指令按 1 递增计数器。
当启用 CTUD 指令并设置 CDEnable 时， CTUD 指令按 1 递减计数器。

在同一扫描期间， CUEnable 和 CDEnable 输入参数都可以切换。该指令在执行递减之前执行递增。

《递增》



《递减》



当禁用 CTUD 指令时， CTUD 指令保持其累计值。设置 FBD_COUNTER 结构的 Reset 输入参数以复位该指令。

算术状态标志： 不受影响

错误条件： 无

执行：

条件：	功能块操作：	结构化文本操作：
预扫描	无需初始化。	无需初始化。
指令第一次扫描	设置 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。	设置 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。
指令第一次运行	设置 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。	设置 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。
EnableIn 处于清零状态	清除 EnableOut，指令不执行任何操作，也不更新输出。	不可用
EnableIn 处于置位状态	该指令设置 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。 在 EnableIn 从清零状态转为置位状态时： • 指令执行。 • 设置 EnableOut。	该指令设置 $CUEnable_{n-1}$ 和 $CDEnable_{n-1}$。 EnableIn 始终为置位状态。 指令执行。
复位	设置时，该指令清除 $CUEnable_{n-1}$、$CDEnable_{n-1}$、CU、CD、DN、OV 和 UN，并将 ACC 设置为零。	设置时，该指令清除 $CUEnable_{n-1}$、$CDEnable_{n-1}$、CU、CD、DN、OV 和 UN，并将 ACC 设置为零。
后扫描	不采取任何操作。	不采取任何操作。

示例： 当 *limit_switch1* 从清零状态转为置位状态时，为一次扫描设置 CUEnable，且 CTUD 指令按 1 递增 ACC 值。当 ACC ≥ PRE 时，设置 DN 参数，这将在 CTUD 指令之后启用功能块指令。

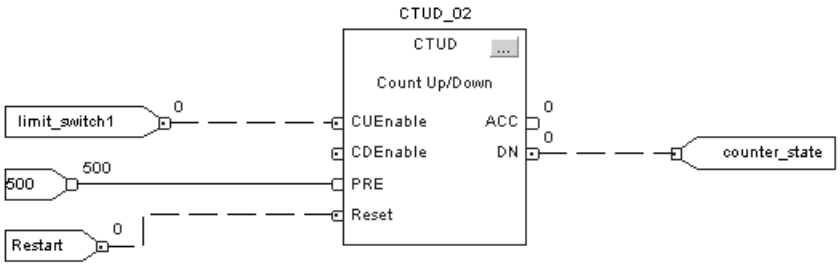
结构化文本

```
CTUD_01.Preset := 500;
CTUD_01.Reset := Restart;
CTUD_O1.CUEnable := limit_switch1;

CTUD(CTUD_01);

counter_state := CTUD_01.DN;
```

功能块



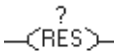
复位 (Reset, RES)

RES 指令复位 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	标记	要复位的结构

说明： 当启用时， RES 指令清除下列元素：

当 RES 指令用于：	该指令清除：
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 指令清除 .ACC 值、.DN 位和 .TT 位，不要使用 RES 指令复位 TOF 计时器。

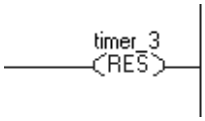
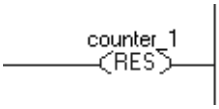
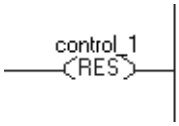
算术状态标志： 不受影响

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	RES 指令复位指定的结构。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

示例：

示例：	说明：
	当启用时，复位 *timer_3*。
	当启用时，复位 *counter_1*。
	当启用时，复位 *control_1*。

说明：

第 3 章

输入 / 输出指令
（MSG、 GSV、 SSV、 IOT）

简介

输入 / 输出指令从 / 向控制器读取 / 写入数据，或者从 / 向另一个网络中的另一个模块读取 / 写入数据块。

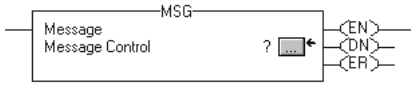
操作目的：	操作指令：	适用语言：	参考页面：
从 / 向另一个模块传送数据	MSG	梯形图 结构化文本	第 3 章 -2
获取控制器状态信息	GSV	梯形图 结构化文本	第 3 章 -33
设置控制器状态信息	SSV	梯形图 结构化文本	第 3 章 -33
• 将输出值发送到位于逻辑中的特定点的 I/O 模块或使用控制器 • 在另一个控制器中触发事件任务	IOT	梯形图 结构化文本	第 3 章 -57

消息 (Message, MSG)

MSG 指令以异步方式从 / 向网络中的另一个模块读取 / 写入数据块。

操作数：







```
MSG(MessageControl);
```

梯形图

操作数：	类型：	格式：	说明：
消息控制	MESSAGE	标记	MESSAGE 结构

结构化文本

操作数与梯形图 MSG 指令的数目相同。

MESSAGE 结构

注意



如果您多次检查状态位

控制器将更改 DN、ER、EW 和 ST 位，而这些位与逻辑扫描不同步。如果在逻辑中的多个位置检查这些位，请使用它们的副本。否则，这些位可能在扫描过程中更改，这样您的逻辑运算就不会像预期的那样工作。

生成副本的一种方法是使用 FLAGS 字。将 FLAGS 字复制到另一个标记中，然后检查副本中的这些状态位。

重要事项

请勿更改 MSG 指令的下列状态位：

- 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。否则，控制器可能发生不可恢复的错误。控制器发生不可恢复的错误时，会从其内存中清除项目。

助记符：	数据类型：	说明：
.FLAGS	INT	FLAGS 成员以一个 16 位字提供对状态成员 （位）的访问。 **位：** **对应成员：** 2 .EW 4 .ER 5 .DN 6 .ST 7 .EN 8 .TO 9 .EN_CC **重要事项：**请勿更改 FLAGS 成员的 EW、ER、DN 或 ST 位。例如，不要清除整个 FLAGS 字。控制器会忽略所做的更改，并使用在内部存储的位值。
.ERR	INT	如果设置了 .ER 位，则错误代码字将标识 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	请求的长度指定消息指令尝试传输的字数。
.DN_LEN	INT	完成的长度标识实际传输的字数。
.EW	BOOL	启用等待位是在控制器检测到有消息请求进入队列时设置的。设置 .ST 位时，控制器重置 .EW 位。 **重要事项：**不要更改 EW 位。控制器会忽略所做的更改，并使用在内部存储的位值。
.ER	BOOL	错误位是在控制器检测到传输失败时设置的。在梯级输入条件再次从 false 变为 true 时重置 .ER 位。 **重要事项：**不要更改 ER 位。
.DN	BOOL	完成位是在成功传输消息的最后一个包时设置的。在梯级输入条件再一次从 false 变为 true 时重置 .DN 位。 **重要事项：**不要更改 DN 位。
.ST	BOOL	启动位是在控制器开始执行 MSG 指令时设置的。在设置 .DN 位或 .ER 位时重置 .ST 位。 **重要事项：**不要更改 ST 位。控制器会忽略所做的更改，并使用在内部存储的位值。
.EN	BOOL	启用位是在梯级输入条件变为 true 时设置的，并一直保持设置状态，直到设置 .DN 位或 .ER 位且梯级输入条件为 false。如果梯级输入条件变为 false，但是清除了 .DN 位和 .ER 位，则 .EN 位将保持设置状态。 **重要事项：**不要更改 EN 位。
.TO	BOOL	如果您手动设置 .TO 位，控制器将停止处理消息，并设置 .ER 位。
.EN_CC	BOOL	启用缓存位确定如何管理 MSG 连接。请参考第 3-30 页的 " 选择缓存选项 "。即使 .EN_CC 位已设置，也不会缓存通过串行端口传出的 MSG 指令的连接。
.ERR_SRC	SINT	被 RSLogix 5000 软件用来在 Message Configuration （消息配置）对话框中显示错误路径。
.DestinationLink	INT	要更改 DH+ 或 CIP with Source ID （带有源 ID 的 CIP）消息的 Destination Link （目标链接），请将此成员设置为所需的值。

助记符：	数据类型：	说明：
.DestinationNode	INT	要更改 DH+ 或 CIP with Source ID （带有源 ID 的 CIP）消息的 Destination Node （目标节点），请将此成员设置为所需的值。
.SourceLink	INT	要更改 DH+ 或 CIP with Source ID （带有源 ID 的 CIP）消息的 Source Link （源链接），请将此成员设置为所需的值。
.Class	INT	要更改 CIP Generic （CIP 通用）消息的 Class 参数，请将此成员设置为所需的值。
.Attribute	INT	要更改 CIP Generic （CIP 通用）消息的 Attribute 参数，请将此成员设置为所需的值。
.Instance	DINT	要更改 CIP Generic （CIP 通用）消息的 Instance 参数，请将此成员设置为所需的值。
.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，请将此成员设置为所需的值。 **如果消息：** **则本地数组为：** 读取数据 Destination 元素 写入数据 Source 元素
.Channel	SINT	要将消息从 1756-DHRIO 模块的另一个信道发出，请将此成员设置为所需的值。使用 ASCII 字符 A 或 B。
.Rack	SINT	要更改块传输消息的框架编号，请将此成员设置为所需的框架编号 （八进制）。
.Group	SINT	要更改块传输消息的组编号，请将此成员设置为所需的组编号 （八进制）。
.Slot	SINT	要更改块传输消息的插槽编号，请将此成员设置为所需的插槽编号。 **如果消息通过此网络传输：** **则通过以下方式指定插槽编号：** 通用远程 I/O 八进制 ControlNet 十进制 (0-15)
.Path	STRING	要将消息发送到另一个控制器，请将此成员设置为新路径。 • 以十六进制值输入路径。 • 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，并自动创建新字符串的一部分或整个字符串，请右键单击字符串标记，然后选择 *Go to Message Path Editor* （转至消息路径编辑器）。
.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，请将此成员设置为所需的值。 **如果消息：** **则远程数组为：** 读取数据 Source 元素 写入数据 Destination 元素
.RemoteElement	STRING	要指定消息发送到的控制器中的另一个标记或地址，请将此成员设置为所需的值。以 ASCII 字符输入标记或地址。 **如果消息：** **则远程数组为：** 读取数据 Source 元素 写入数据 Destination 元素

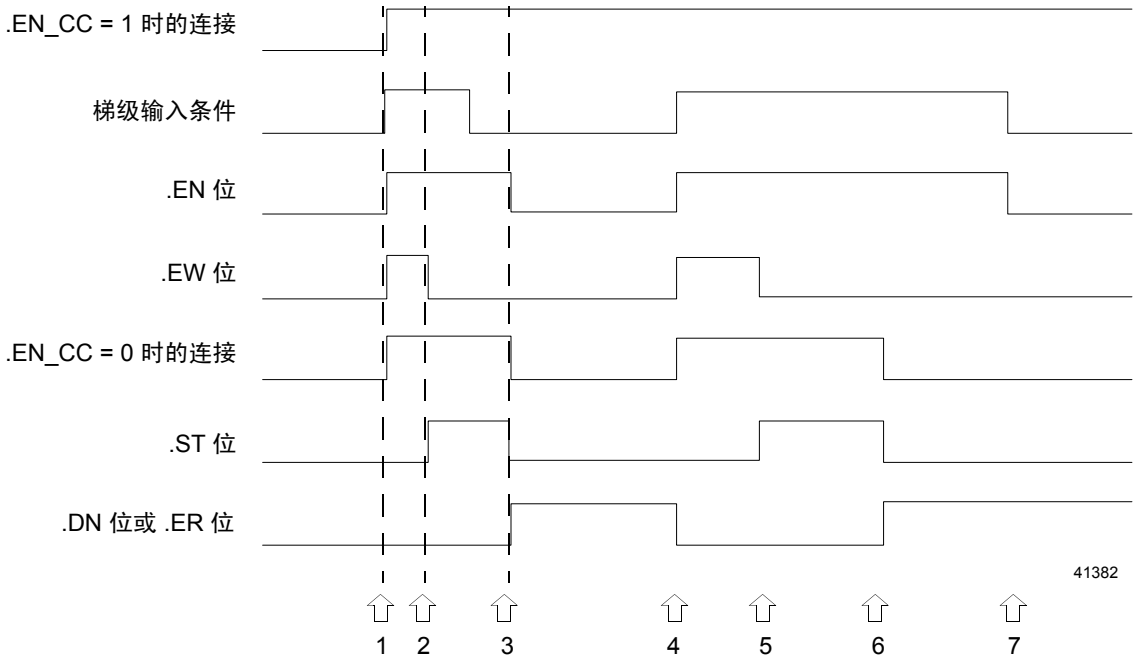
<table>
<tr><th>助记符：</th><th>数据类型：</th><th>说明：</th></tr>
<tr><td>.Unconnnected Timeout</td><td>DINT</td><td>unconnected （未连接）消息的超时时间限制，或建立连接时的超时时间限制。默认值为 30 秒。
<table><tr><th>如果消息为</th><th>则</th></tr><tr><td>unconnected （未连接）</td><td>如果控制器在 UnconnectedTimeout 时间内未得到响应，ER 位将打开。</td></tr><tr><td>connected （已连接）</td><td>如果控制器在 UnconnectedTimeout 时间内未得到关于建立连接的响应， ER 位将打开。</td></tr></table></td></tr>
<tr><td>.ConnectionRate</td><td>DINT</td><td rowspan="2">连接后 connected （已连接）消息的超时时间限制。此超时时间限制适用于来自另一个设备的关于发送数据的响应。
• 只有在建立连接后，此超时时间限制才适用。
• 超时时间限制 = ConnectionRate x TimeoutMultiplier。
• 默认 ConnectionRate 为 7.5 秒。
• 默认 TimeoutMultiplier 为 0 （表示倍乘系数为 4）。
• connected （已连接）消息的默认超时时间限制为 30 秒 （7.5 秒 x 4 = 30 秒）。
• 要更改超时时间限制，请更改 ConnectionRate，而将 TimeoutMultiplier 保留为默认值。</td></tr>
<tr><td>.Timeout Multiplier</td><td>SINT</td></tr>
</table>

说明 MSG 指令传输数据的元素。

这是一个转换指令：

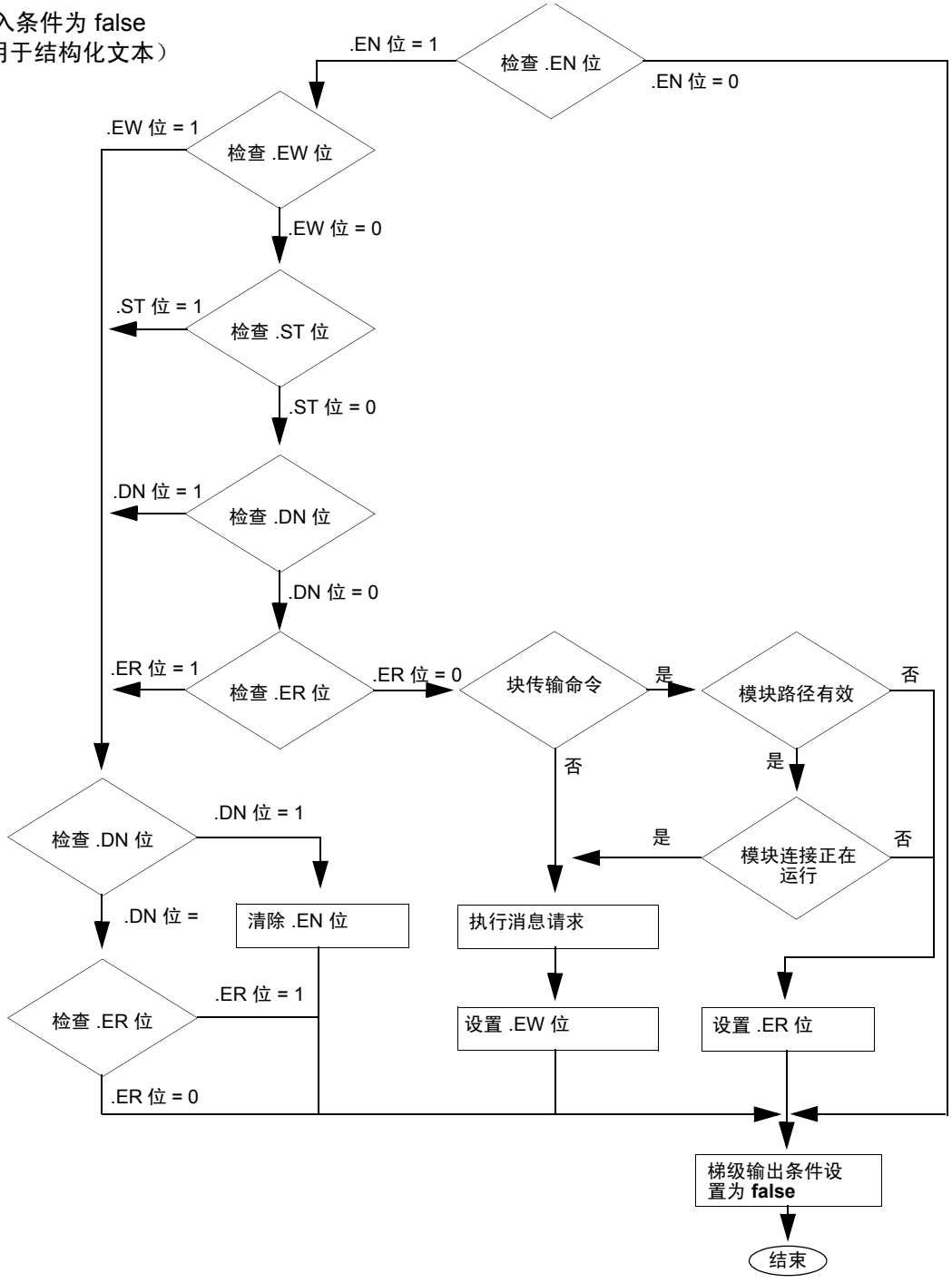
- 在梯形图中，每次应执行指令时都将梯级输入条件由清除状态切换为设置状态。
- 在结构化文本中，限制指令只对转换执行。请参见附录 C。

每个元素的大小取决于您指定的数据类型和使用的消息命令的类型。



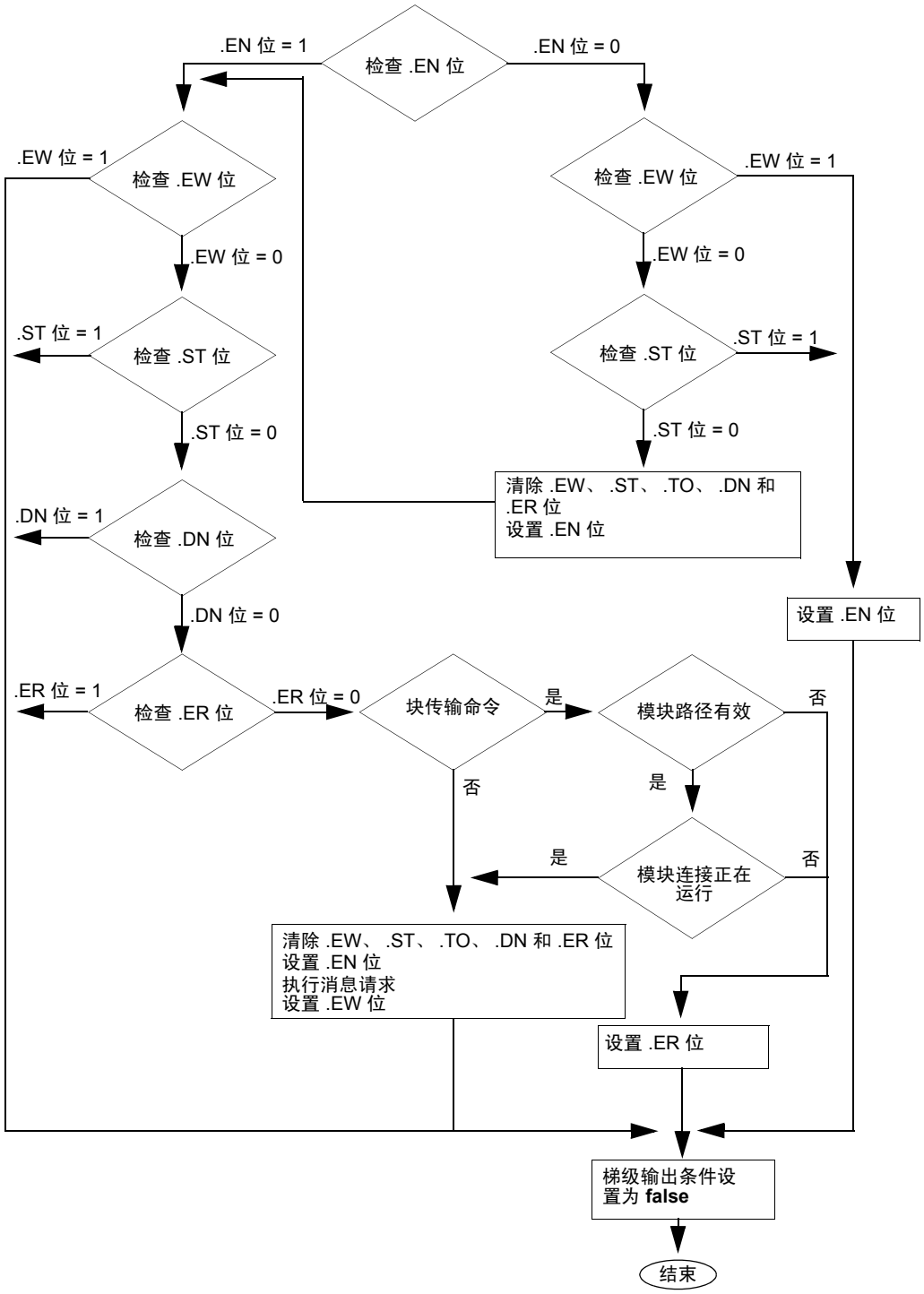
数字：	说明：	数字：	说明：
1	梯级输入条件为 true 设置 .EN 设置 .EW 打开连接 *	5	发送消息 设置 .ST 清除 .EW
2	发送消息 设置 .ST 清除 .EW	6	消息已完成或发生错误 梯级输入条件仍为 true 设置 .DN 或 .ER 清除 .ST 关闭连接 （如果 .EN_CC = 0）
3	消息已完成或发生错误 梯级输入条件为 false 设置 .DN 或 .ER 清除 .ST 关闭连接 （如果 .EN_CC = 0） 清除 .EN （梯级输入条件为 false）	7	梯级输入条件变为 false 并设置 .DN 或 .ER 清除 .EN
4	梯级输入条件为 true 以前设置了 .DN 或 .ER 设置 .EN 设置 .EW 打开连接 * 清除 .DN 或 .ER		

执行：

条件：	梯形图操作：	结构化文本操作：
扫描前	梯级输出条件设置为 false。	不执行任何操作。
梯级输入条件为 false （不适用于结构化文本）		
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	无

条件：	梯形图操作：	结构化文本操作
EnableIn 已设置	无	EnableIn 始终为设置状态。 指令执行。

指令执行



扫描后	梯级输出条件设置为 false。	不执行任何操作。

算术状态标志： 不受影响

错误条件： 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 软件并不总是显示完整的说明。

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

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	与说明相同
001F	与连接有关的故障 （参见扩展错误代码）	与说明相同
0022	收到无效的回复	与说明相同
0025	关键段错误	与说明相同
0026	无效 IOI 错误	与说明相同
0027	列表中存在意外的属性	与说明相同
0028	DeviceNet 错误 - 成员 ID 无效	与说明相同
0029	DeviceNet 错误 - 成员不可设置	与说明相同
00D1	模块未处于运行状态	未知错误
00FB	消息端口不受支持	未知错误
00FC	消息数据类型不受支持	未知错误
00FD	消息未初始化	未知错误
00FE	消息超时	未知错误
00FF	一般错误 （参见扩展错误代码）	未知错误

扩展错误代码

RSLogix 5000 软件不为扩展错误代码显示文本。

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
0103	传送不受支持
0106	所有权冲突
0107	找不到连接
0108	连接类型无效
0109	连接大小无效
0110	模块未配置
0111	EPR 不受支持
0114	错误的模块
0115	错误的设备类型
0116	错误的版本
0118	无效配置格式
011A	应用程序已用完连接

扩展错误代码（十六进制）：	说明：
0203	连接超时
0204	未连接消息超时
0205	未连接发送参数错误
0206	消息太长
0301	无缓冲区内存
0302	带宽不可用
0303	无可用筛选器
0305	签名相符
0311	端口不可用
0312	链接地址不可用
0315	无效段类型
0317	连接未计划

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	扩展状态已用完内存
0001	扩展状态已用完实例

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	大小太小
201C	无效大小
2100	权限无效
2101	按键开关位置无效
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

扩展错误代码（十六进制）：	说明：
2107	类型无效或不受支持
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	控制器处于错误的模式
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 消息类型 （PCCC 消息）关联的错误提供了新的错误代码。

- 这一更改使 RSLogix 5000 软件可以为其中的许多错误显示更有意义的说明。以前，该软件不为与 00F0 错误代码关联的错误提供说明。
- 该更改还使错误代码与其他控制器 （如 PLC-5 控制器）返回的错误更一致。

下表对照显示了 R9.x 及以前版本中的错误代码与 R10.x 及以后版本中的错误代码。更改之后， .ERR 成员为每个 PCCC 错误返回一个唯一值。这些错误不再需要 .EXERR。

表 3.1 PLC 和 SLC 错误代码 （十六进制）

R9.x 及以前的版本		R10.x 及以后的版本		说明：
.ERR	.EXERR	.ERR	.EXERR	
0010		1000		来自本地处理器的命令或格式不合法
0020		2000		通信模块不工作
0030		3000		远程节点丢失、连接断开或关闭
0040		4000		处理器已连接但发生故障 （硬件）
0050		5000		站编号不正确
0060		6000		请求的功能不可用
0070		7000		处理器处于程序模式
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，因此无法访问它
00F0	0001	F001		处理器对地址的转换不正确
00F0	0002	F002		地址不完整
00F0	0003	F003		地址不正确
00F0	0004	F004		非法的地址格式 - 找不到符号
00F0	0005	F005		非法的地址格式 - 符号不包含字符或所包含字符的个数超过设备支持的最大数目
00F0	0006	F006		目标处理器中没有地址文件
00F0	0007	F007		对于请求的字数而言，目标文件太小
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		数据或文件太大 内存不够
00F0	000A	F00A		目标处理器无法将请求的信息放入包中

表 3.1 PLC 和 SLC 错误代码 （十六进制）（续）

R9.x 及以前的版本		R10.x 及以后的版本		说明：
.ERR	.EXERR	.ERR	.EXERR	
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		无法执行命令
00F0	000F	F00F		溢出；柱状图溢出
00F0	0010	F010		无访问权
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		存在对已删除区域的地址引用
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		无法使用扫描程序与 1771 框架适配器进行通信
00F0	0017	F017		无法使用适配器与模块进行通信
00F0	0018	F018		1771 模块响应无效
00F0	0019	F019		标签重复
00F0	001A	F01A		文件所有者处于活动状态 - 文件正在使用中
00F0	001B	F01B		程序所有者处于活动状态 - 有人正在下载或在线编辑程序
00F0	001C	F01C		磁盘文件有写保护，或由于其他原因不可访问 （仅限脱机的情况）
00F0	001D	F01D		磁盘文件被另一个应用程序使用 未执行更新 （仅限脱机的情况）

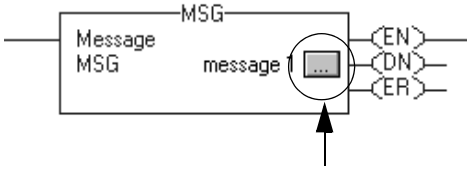
块传输错误代码

下面是 Logix5000 块传输特有的错误代码。

错误代码（十六进制）：	说明：	软件中的显示内容：
00D0	在发出请求后的 3.5 秒内，扫描程序没有收到来自块传输模块的块传输响应	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	未知错误
00D2	扫描程序请求了读取或写入操作，但块传输模块响应以相反的操作	未知错误
00D3	扫描程序请求了某个长度，而块传输模块却响应了另一个长度	未知错误
00D6	扫描程序收到来自块传输模块的响应，该响应指出写入请求失败	未知错误
00EA	扫描程序未被配置与包含此块传输模块的框架进行通信	未知错误
00EB	指定的逻辑插槽对于给定的框架大小不可用	未知错误
00EC	当前有一个块传输请求正在进行，必须提供一个响应，另一个请求才能开始	未知错误
00ED	块传输大小请求与有效块传输大小请求不一致	未知错误
00EE	块传输类型请求与所需的 BT_READ 或 BT_WRITE 不一致	未知错误
00EF	扫描程序在块传输表中找不到可用插槽来满足块传输请求	未知错误
00F0	在块传输未完成的情况下，扫描程序收到重置远程 I/O 信道的请求	未知错误
00F3	远程块传输的队列已满	未知错误
00F5	没有为请求的框架或插槽配置通信信道	未知错误
00F6	没有为远程 I/O 配置通信信道	未知错误
00F7	块传输在完成之前超过指令中设置的块传输超时时间限制	未知错误
00F8	块传输协议中出现错误 - 存在不期而至的块传输	未知错误
00F9	由于通信信道不正确，块传输数据丢失	未知错误
00FA	块传输模块所请求的长度与关联的块传输指令的长度不同	未知错误
00FB	块传输读取数据的校验和错误	未知错误
00FC	在适配器与块传输模块之间进行的块传输写入数据的传输无效	未知错误
00FD	块传输大小与块传输数据表中的索引大小之和大于块传输数据表文件的大小	未知错误

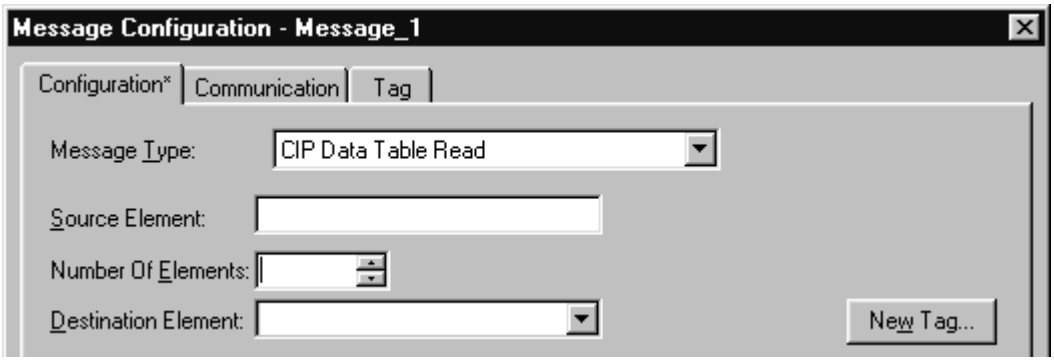
指定配置详细信息

输入 MSG 指令并指定 MESSAGE 结构之后，使用 Message Configuration （消息配置）对话框指定消息的详细信息。



单击此处配置 MSG 指令

您配置的详细信息取决于所选择的消息类型。



如果目标设备为：	选择以下消息类型之一：	参考页面：
Logix5000 控制器	CIP Data Table Read （CIP 数据表读取）	第 3 章 -17
	CIP Data Table Write （CIP 数据表写入）	
使用 RSLogix 5000 软件配置的 I/O 模块	Module Reconfigure （模块重新配置）	第 3 章 -18
	CIP Generic （CIP 通用）	第 3 章 -19
PLC-5 控制器	PLC5 Typed Read （PLC5 类型化读取）	第 3 章 -20
	PLC5 Typed Write （PLC5 类型化写入）	
	PLC5 Word Range Read （PLC5 字范围读取）	
	PLC5 Word Range Write （PLC5 字范围写入）	
SLC 控制器 MicroLogix 控制器	SLC Typed Read （SLC 类型化读取）	第 3 章 -22
	SLC Typed Write （SLC 类型化写入）	
块传输模块	Block-Transfer Read （块传输读取）	第 3 章 -22
	Block-Transfer Write （块传输写入）	
PLC-3 处理器	PLC3 Typed Read （PLC3 类型化读取）	第 3 章 -23
	PLC3 Typed Write （PLC3 类型化写入）	
	PLC3 Word Range Read （PLC3 字范围读取）	
	PLC3 Word Range Write （PLC3 字范围写入）	
PLC-2 处理器	PLC2 Unprotected Read （PLC2 不受保护的读取）	第 3 章 -24
	PLC2 Unprotected Write （PLC2 不受保护的写入）	

必须指定以下配置信息：

对于属性：	指定：
Source Element （源元素）	• 如果选择读取消息类型，则 Source Element （源元素）是您要在目标设备中读取的数据的地址。使用目标设备的寻址语法。 • 如果选择写入消息类型，则 Source Tag （源标记）是您要发送到目标设备的标记的第一个元素。
Number of Elements （元素数目）	读取 / 写入元素的数目取决于所用的数据类型。元素指的是一 “大块”相关数据。例如，*timer1* 标记是包含计时器控制结构的元素。
Destination Element （目标元素）	• 如果选择读取消息类型，则 Destination Element （目标元素）是 Logix5000 控制器中要用来存储从目标设备读取的数据的标记的第一个元素。 • 如果选择写入消息类型，则 Destination Element （目标元素）是目标设备中要写入数据的位置的地址。

指定 CIP Data Table Read（CIP 数据表读取）和 CIP Data Table Write （CIP 数据表写入）消息

CIP Data Table Read （CIP 数据表读取）和 CIP Data Table Write （CIP 数据表写入）消息类型在 Logix5000 控制器之间传输数据。

选择命令：	操作目的：
CIP Data Table Read （CIP 数据表读取）	从另一个控制器中读取数据。 Source 和 Destination 类型必须相符。
CIP Data Table Write （CIP 数据表写入）	将数据写入另一个控制器。 Source 和 Destination 类型必须相符。

重新配置 I/O 模块

使用 Module Reconfigure （模块重新配置）消息可以将新的配置信息发送到 I/O 模块。在重新配置过程中：

- 输入模块继续将输入数据发送到控制器。
- 输出模块继续控制它们的输出设备。

Module Reconfigure （模块重新配置）消息需要以下配置属性：

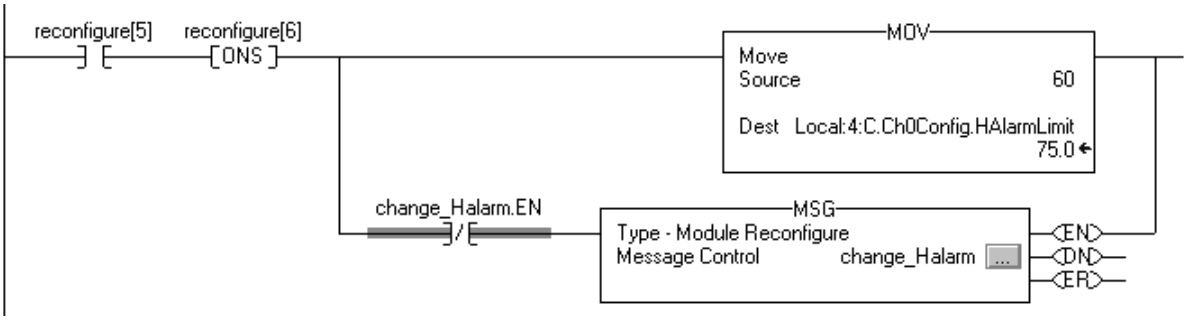
在此属性中：	选择：
Message Type （消息类型）	Module Reconfigure （模块重新配置）

示例： 要重新配置 I/O 模块，请执行下列操作：

1. 将模块的配置标记的必需成员设置为新值。
2. 将 Module Reconfigure （模块重新配置）消息发送到模块。

当设置 *reconfigure[5]* 时，对于插槽 4 中的本地模块，将高限警报设置为 60。然后 Module Reconfigure （模块重新配置）消息会将新的警报值发送到模块。单触发指令可以防止梯级在 *reconfigure[5]* 打开的情况下向模块发送多个消息。

继电器梯形图



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 Generic （CIP 通用）消息

CIP Generic （CIP 通用）消息对 I/O 模块执行特定操作。

操作目的	在此属性中：	键入或选择：	
对数字输出模块执行脉冲测试	Message Type （消息类型）	CIP Generic （CIP 通用）	
	Service Type （服务类型）	Pulse Test （脉冲测试）	
	Source （源）	类型 INT [5] 的 *tag_name*	
		此数组包含：	
		tag_name[0]	要测试的点的位掩码 （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 µ，通常为 2000 微秒）
		tag_name[3]	ControlLogix I/O 的零交叉延迟 （数百微秒 µ，通常为 4000 微秒）
		tag_name[4]	验证延迟
	Destination （目标）	留空	
在数字输出模块上重置电子保险丝	Message Type （消息类型）	CIP Generic （CIP 通用）	
	Service Type （服务类型）	Reset Electronic Fuse （重置电子保险丝）	
	Source （源）	类型 DINT 的 *tag name* 此标记表示要重置保险丝的点的位掩码。	
	Destination （目标）	留空	
在数字输入模块上重置锁存诊断	Message Type （消息类型）	CIP Generic （CIP 通用）	
	Service Type （服务类型）	Reset Latched Diagnostics (I) （重置锁存诊断 (I)）	
	Source （源）	类型 DINT 的 *tag_name* 此标记表示要重置诊断的点的位掩码。	
在数字输出模块上重置锁存诊断	Message Type （消息类型）	CIP Generic （CIP 通用）	
	Service Type （服务类型）	Reset Latched Diagnostics (O) （重置锁存诊断 (O)）	
	Source （源）	类型 DINT 的 *tag_name* 此标记表示要重置诊断的点的位掩码。	

操作目的	在此属性中：	键入或选择：
发出模拟输入模块的警报	Message Type （消息类型）	CIP Generic （CIP 通用）
	Service Type （服务类型）	选择要发出的警报： • Unlatch All Alarms (I) （发出所有警报 (I)） • Unlatch Analog High Alarm (I) （发出模拟高警报 (I)） • Unlatch Analog High High Alarm (I) （发出模拟高高警报 (I)） • Unlatch Analog Low Alarm (I) （发出模拟低警报 (I)） • Unlatch Analog Low Low Alarm (I) （发出模拟低低警报 (I)） • Unlatch Rate Alarm (I) （发出速率警报 (I)）
	Instance （实例）	要发出的警报的信道
发出模拟输出模块的警报	Message Type （消息类型）	CIP Generic （CIP 通用）
	Service Type （服务类型）	选择要发出的警报： • Unlatch All Alarms (O) （发出所有警报 (O)） • Unlatch High Alarm (O) （发出高警报 (O)） • Unlatch Low Alarm (O) （发出低警报 (O)） • Unlatch Ramp Alarm (O) （发出斜率警报 (O)）
	Instance （实例）	要发出的警报的信道

指定 PLC-5 消息

PLC-5 消息类型用于与 PLC-5 控制器进行通信。

选择命令：	操作目的：
PLC5 Typed Read （PLC5 类型化读取）	读取 16 位整数、浮点数或字符串类型数据，并保持数据的完整性。请参见第 3-21 页的表 3.2 .
PLC5 Typed Write （PLC5 类型化写入）	写入 16 位整数、浮点数或字符串类型数据，并保持数据的完整性。请参见第 3-21 页的表 3.2
PLC5 Word Range Read （PLC5 字范围读取）	不考虑数据类型，读取 PLC-5 内存中某一连续范围的 16 位字。 此命令从指定为 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 Destination Tag 的地址处开始，存储来自 Source Element 的数据。
PLC5 Word Range Write （PLC5 字范围写入）	不考虑数据类型，将 Logix5000 内存中某一连续范围的 16 位字写入 PLC-5 内存。 此命令从指定为 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 处理器中被指定为 Destination Element （目标元素）的地址开始，存储来自 Source Tag 的数据。

下表显示了用于 PLC5 Typed Read （PLC5 类型化读取）和 PLC5 Typed Write （PLC5 类型化写入）消息的数据类型。

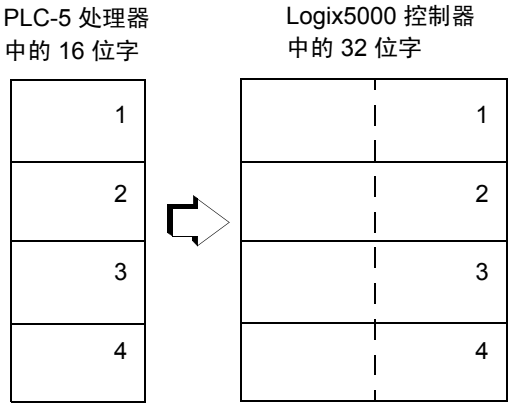
表 3.2 用于 PLC5 Typed Read（PLC5 类型化读取）和 PLC5 Typed Write（PLC5 类型化写入）消息的数据类型

对于此 PLC-5 数据类型：	使用此 Logix5000 数据类型：
B	INT
F	REAL
N	INT
	DINT （如果值 ≥ –32,768 且 ≤ 32,767，则只将 DINT 值写入 PLC-5 控制器。）
S	INT
ST	STRING

类型化读取 （Typed Read）和类型化写入 （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。

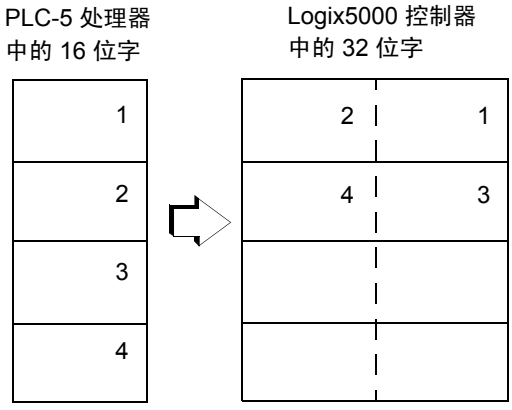
下图说明了类型化命令与字范围命令之间的差异。该示例使用读取命令将 PLC-5 处理器中的内容读入 Logix5000 控制器中。

类型化读取命令



类型化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记。数据结构和值随目标数据类型而变化。

指定 SLC 消息

使用 SLC 消息类型可以与 SLC 和 MicroLogix 控制器进行通信。下表显示了使用该指令可以访问的数据类型。该表还显示了对应的 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

指定块传输消息

块传输消息类型用于通过通用远程 I/O 网络与块传输模块进行通信。

操作目的：	选择命令：
从块传输模块读取数据。 此消息类型取代 BTR 指令。	Block-Transfer Read （块传输读取）
将数据写入块传输模块。 此消息类型取代 BTW 指令。	Block-Transfer Write （块传输写入）

配置块传输消息时请遵循以下原则：

- 除了 MESSAGE、 AXIS 和 MODULE 结构，源标记 （用于 BTW）和目标标记 （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。可以指定的整数数目为 0 到 64 之间的某个数。

如果希望：	则指定：
由块传输模块确定要发送的 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)。	

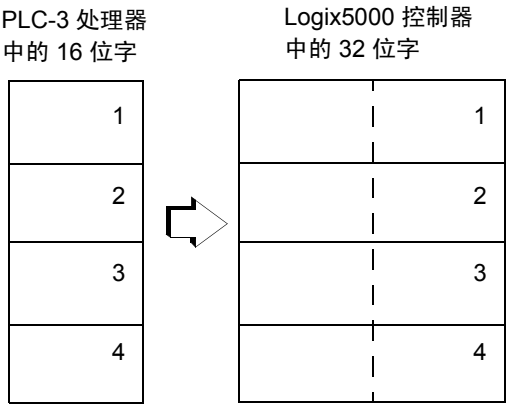
指定 PLC-3 消息

PLC-3 消息类型是为 PLC-3 处理器设计的。

选择命令：	操作目的：
PLC3 Typed Read （PLC3 类型化读取）	读取整数或 REAL 类型数据。 对于整数，此命令从 PLC-3 处理器读取 16 位整数，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。 此命令还从 PLC-3 读取浮点数据，并将它存储在 Logix5000 控制器中的 REAL 数据类型标记中。
PLC3 Typed Write （PLC3 类型化写入）	写入整数或 REAL 类型数据。 此命令将 SINT 或 INT 数据写入 PLC-3 整数文件，并保持数据完整性。只要 DINT 数据符合 INT 数据类型 （–32,768 ≥ 数据 ≤ 32,767），您就可以写入这种数据。 此命令还将 Logix5000 控制器中的 REAL 类型数据写入 PLC-3 浮点文件。
PLC3 Word Range Read （PLC3 字范围读取）	不考虑数据类型，读取 PLC-3 内存中某一连续范围的 16 位字。 此命令从指定为 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 Destination Tag 的地址处开始，存储来自 Source Element 的数据。
PLC3 Word Range Write （PLC3 字范围写入）	不考虑数据类型，将 Logix5000 内存中某一连续范围的 16 位字写入 PLC-3 内存。 此命令从指定为 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 处理器中被指定为 Destination Element （目标元素）的地址开始，存储来自 Source Tag 的数据。

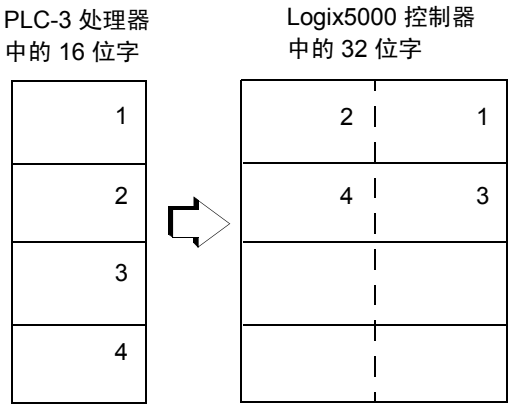
下图说明了类型化命令与字范围命令之间的差异。该示例使用读取命令将 PLC-3 处理器中的内容读入 Logix5000 控制器中。

类型化读取命令



类型化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记。数据结构和值随目标数据类型而变化。

指定 PLC-2 消息

PLC-2 消息类型是为 PLC-2 处理器设计的。

选择命令：	操作目的：
PLC2 Unprotected Read （PLC2 不受保护的读取）	从 PLC-2 数据表的任何区域或另一个处理器的 PLC-2 兼容性文件读取 16 位字。
PLC2 Unprotected Write （PLC2 不受保护的写入）	将 16 位字写入 PLC-2 数据表的任何区域或另一个处理器的 PLC-2 兼容性文件。

该消息传输使用 16 位字，因此请确保 Logix5000 标记正确地存储所传输的数据 （通常以 INT 数组的形式存储）。

MSG 配置示例

下面的示例显示了不同控制器组合的源和目标标记以及源和目标元素。

对于从 Logix5000 控制器发出并写入到另一个控制器的 MSG 指令：

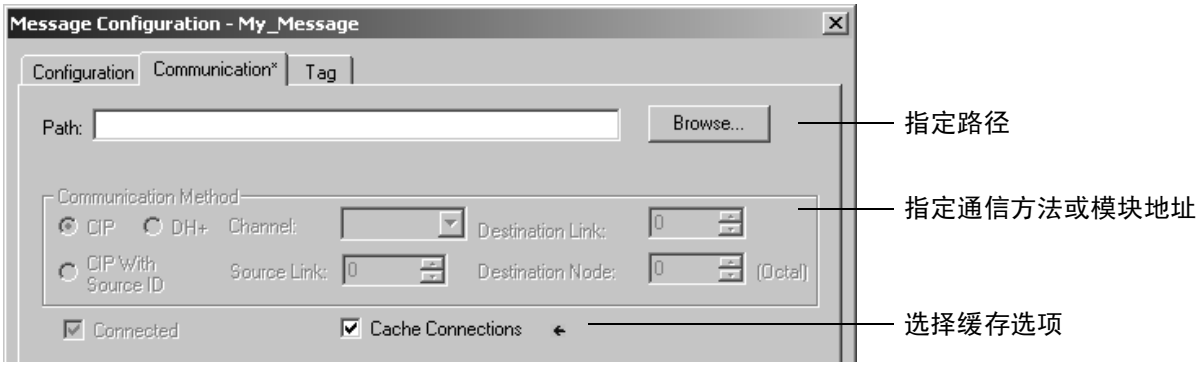
消息路径：	示例 源 和 目标：	
Logix5000 → Logix5000	源标记	*array_1[0]*
	目标标记	*array_2[0]*
	可以为源标记使用别名标记 （在 Logix5000 源控制器中）。 **不能**对目标标记使用别名。目标标记必须是基础标记。	
Logix5000 → PLC-5 Logix5000 → SLC	源标记	*array_1[0]*
	目标元素	*N7:10*
	可以为源标记使用别名标记 （在 Logix5000 源控制器中）。	
Logix5000 → PLC-2	源标记	*array_1[0]*
	目标元素	*010*

对于从 Logix5000 控制器发出并从另一个控制器读取的 MSG 指令：

消息路径：	示例 源和目标：	
Logix5000 → Logix5000	源标记	*array_1[0]*
	目标标记	*array_2[0]*
	不能对源标记使用别名标记。源标记必须是基础标记。 可以为目标标记使用别名标记 （在 Logix5000 源控制器中）。	
Logix5000 → PLC-5 Logix5000 → SLC	源元素	*N7:10*
	目标标记	*array_1[0]*
	可以为目标标记使用别名标记 （在 Logix5000 源控制器中）。	
Logix5000 → PLC-2	源元素	*010*
	目标标记	*array_1[0]*

指定通信详细信息

要配置 MSG 指令，请在 Communication （通信）选项卡上指定以下详细信息。



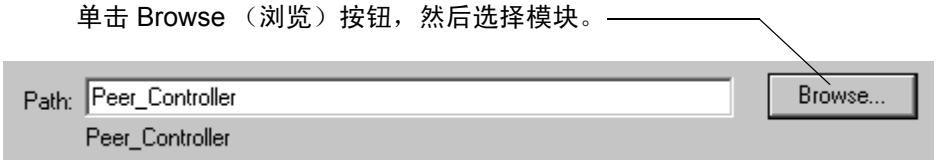
指定路径

路径显示了消息到达目的地所采用的路由。它使用控制器 I/O 配置中的名称、您键入的数字或者两者都使用：

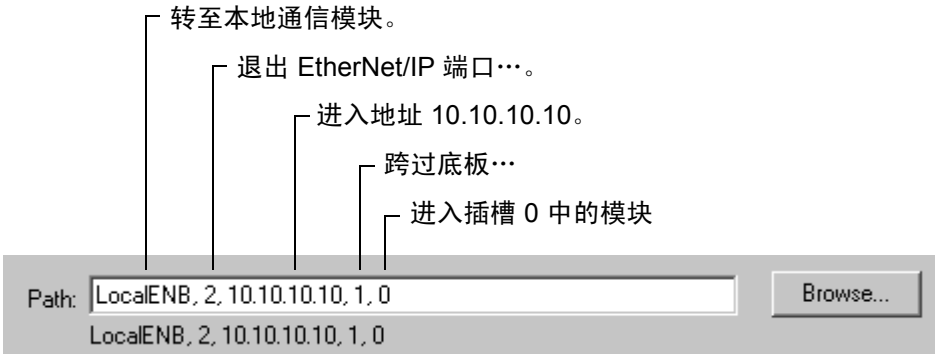
如果	则
控制器的 I/O 配置中包含获取消息的模块。	使用 *Browse* （浏览）按钮选择模块。
控制器的 I/O 配置中只有本地通信模块。	1. 使用 *Browse* （浏览）按钮选择本地通信模块。 2. 键入路径的其余部分。
控制器的 I/O 配置中不包含任何需要用于消息的模块。	键入路径。

示例

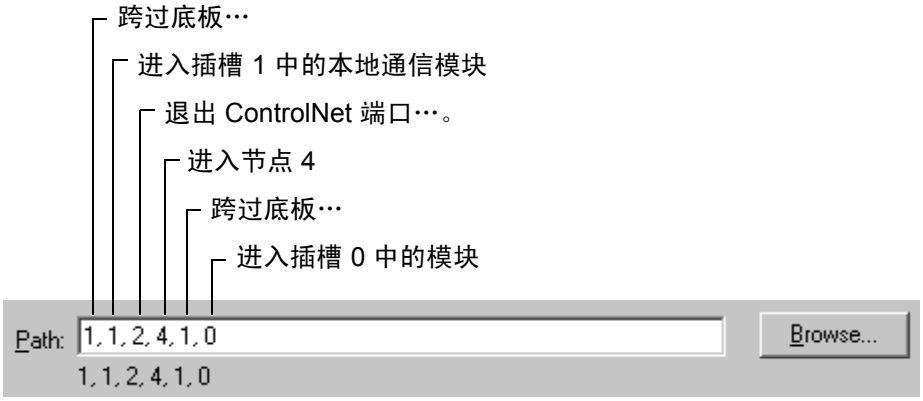
控制器的 I/O 配置中包含获取消息的模块。



控制器的 I/O 配置中只有本地通信模块。



控制器的 I/O 配置中不包含任何需要用于消息的模块。



键入路径时请使用如下格式：

port, next_address, port, next_address, …

其中	表示	
	对于此网络	**类型**
port	底板	1
	DF1 （串行、串行信道 0）	2
	ControlNet	
	EtherNet/IP	
	DH+ 信道 A	
	DH+ 信道 B	3
	DF1 信道 1 （串行信道 1）	
next_address	底板	模块的插槽编号
	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	可以按照以下任一种格式指定 EtherNet/IP 网络上的模块： IP 地址 （如 10.10.10.10） IP 地址 : 端口 （如 10.10.10.10:24） DNS 名称 （如 tanks） DNS 名称 : 端口 （如 tanks:24）

对于块传输

对于块传输消息，请将下列模块添加到控制器的 I/O 配置中：

对于以下网络中的块传输：	将下列模块添加到 I/O 配置中：
ControlNet	• 本地通信模块 （如 1756-CNB 模块） • 远程适配器模块 （如 1771-ACN 模块）
通用远程 I/O	• 本地通信模块 （如 1756-DHRIO 模块） • 为机架中的每个框架或某个框架的每部分添加一个远程适配器模块 （如 1771-ASB 模块） • 块传输模块 （可选）

指定通信方法或模块地址

使用下表来为消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**	不需要指定其他内容	
EtherNet/IP 网络上的 PLC-5 控制器			
ControlNet 网络上的 PLC-5 控制器			
SLC 5/05 控制器			
DH+ 网络上的 PLC-5 控制器	**DH+**	Channel （信道）：	连接到 DH+ 网络的 1756-DHRIO 模块的信道 A 或 B
DH+ 网络上的 SLC 控制器		Source Link （源链接）：	1756-DHRIO 模块的路由表中被分配给控制器底板的链接 ID。（路由表中的源节点自动变为控制器的插槽编号。）
PLC-3 处理器		Destination Link （目标链接）	目标设备所在的远程 DH+ 链接的 ID
PLC-2 处理器		Destination Node （目标节点）：	目标设备的站地址，以八进制表示
		如果只有一个 DH+ 链接，且您未使用 RSLinx 软件为远程链接配置 DH/RIO 模块，请为 Source Link （源链接）和 Destination Link （目标链接）均指定 0。	
工作站上的应用程序，该应用程序借助 RSLinx 接收通过 EtherNet/IP 或 ControlNet 网络路由的不期而至的消息	**CIP with Source ID （带有源 ID 的 CIP）** （这使应用程序可以接收来自控制器的数据。）	Source Link （源链接）：	RSLinx 软件中主题的远程 ID
		Destination Link （目标链接）：	RSLinx 中设置的虚拟链接 ID (0-65535)
		Destination Node （目标节点）：	应用程序向 RSLinx 提供的目标 ID （0-77 八进制）。有关 RSLinx 中的 DDE 主题，请使用 77。
		ControlLogix 控制器的插槽编号被用作 Source Node （源节点）。	
通用远程 I/O 网络上的块传输模块	RIO	Channel （信道）：	连接到 RIO 网络的 1756-DHRIO 模块的信道 A 或 B
		Rack （框架）	模块的框架编号 （八进制）
		Group （组）	模块的组编号
		Slot （插槽）	模块所在插槽的编号
ControlNet 网络上的块传输模块	ControlNet	Slot （插槽）	模块所在插槽的编号

选择缓存选项

它可能使用连接来发送或接收数据，具体取决于您配置 MSG 指令的方式。

消息类型：	通信方法：	使用连接：
CIP data table read （CIP 数据表读取）或 CIP data table write （CIP 数据表写入）	→	✓
PLC2、 PLC3、 PLC5 或 SLC （所有类型）	CIP CIP with Source ID（带有源 ID 的 CIP）	
	DH+	✓
CIP Generic （CIP 通用）	→	您的选项 (1)
block-transfer read （块传输读取）或 block-transfer write （块传输写入）	→	✓

(1) 您可以连接 CIP generic （CIP 通用）消息。但对于我们建议的大多数应用程序，可以让 CIP generic （CIP 通用）消息保留断开状态。

如果 MSG 指令使用连接，您可以选择保持连接为打开状态 （缓存）或在消息传输完毕时关闭连接。

如果：	则：
缓存连接	在 MSG 指令完成后连接保持打开。这样可以缩短执行时间。每次消息执行时都打开连接会延长执行时间。
不缓存连接	在 MSG 指令完成后连接关闭。这样会将该连接释放以作他用。

控制器对您可以缓存的连接数有以下限制：

如果软件和固件版本为：	则可以缓存：
11.x 或更早版本	• 最多 16 个连接的块传输消息 • 最多 16 个连接的其他类型的消息
12.x 或更高版本	最多 32 个连接

如果有几个消息发送到同一设备，则这些消息可能能够共用一个连接。

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。
同一设备	同时启用	每个 MSG 指令使用一个连接。
	未同时启用	MSG 指令共用一个连接。（即它们一起计为 1 个连接。）

示例 共用连接

如果控制器向同一模块交替发送 block-transfer read （块传输读取）消息和 block-transfer write （块传输写入）消息，则这两个消息一起计为 1 个连接。对这两个消息的缓存在缓存列表中计为 1 次缓存。

指导原则

当对 MSG 指令进行计划和编程时，请遵循以下原则：

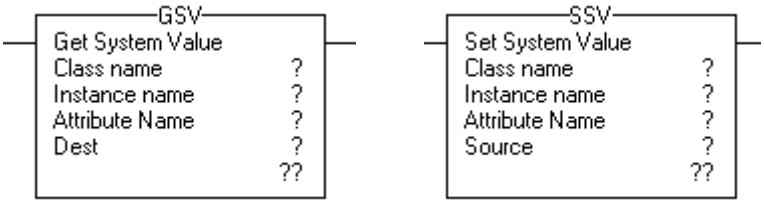
原则：	详细信息：
1. 为每个 MSG 指令各创建一个控制标记。	每个 MSG 指令都需要有自己的控制标记。 • 数据类型 = MESSAGE • 范围 = 控制器 • 标记 《不能》用作数组或用户定义数据类型的一部分。
2. 将源和 / 或目标数据保持在控制器范围内。	MSG 指令只能访问 Controller Tags 文件夹 （控制器范围）中的标记。
3. 如果 MSG 发送到使用 16 位整数的设备，则在 MSG 中使用 INT 的缓冲区，在整个项目中使用 DINT 的缓冲区。	如果消息发送到使用 16 位整数的设备 （如 PLC-5® 或 SLC 500™ 控制器），并且它传输的是整数 （而非 REAL），则在消息中使用 INT 的缓冲区，在整个项目中使用 DINT 的缓冲区。 这样可以提高项目效率，因为 Logix 控制器在处理 32 位整数 (DINT) 时可以更高效地执行操作且使用的内存较少。 要在 INT 与 DINT 之间转换，请参见 *Logix5000 Controllers Common Procedures* （Logix5000 控制器通用编程步骤），出版号 1756-PM001。
4. 缓存最常执行的已连接 MSG。	缓存最常执行的 MSG 指令的连接，指令数最多可达到您的控制器版本允许的最大数目。 这样可以缩短执行时间，因为控制器不必在每次消息执行时都打开连接。
5. 如果希望同时启用 16 个以上 MSG，请使用某些管理策略类型。	如果您同时启用 16 个以上 MSG，则一些 MSG 指令在进入队列时可能会延迟。为了保证每个消息都执行，请使用下列选项之一： • 按顺序启用每个消息。 • 按组启用消息。 • 对消息进行编程，以便与多个设备进行通信。有关更多信息，请参见 *Logix5000 Controllers Common Procedures* （Logix5000 控制器通用编程步骤），出版号 1756-PM001。 • 对逻辑进行编程，以协调消息的执行。有关更多信息，请参见 *Logix5000 Controllers Common Procedures* （Logix5000 控制器通用编程步骤），出版号 1756-PM001。
6. 确保未连接和未缓存的 MSG 的数目小于未连接的缓冲区的数目。	控制器可以有 10 - 40 个未连接的缓冲区。默认数目为 10。 • 如果在指令离开消息队列时所有未连接的缓冲区都在使用，则指令会出错并且不传输数据。 • 可以增加未连接缓冲区的数目 （最大为 40），但始终要遵守第 5 条原则。 • 要增加未连接缓冲区的数目，请参见 *Logix5000 Controllers Common Procedures* （Logix5000 控制器通用编程步骤），出版号 1756-PM001。

获取系统值（Get System Value， GSV）和设置系统值 （Set System Value， SSV）

GSV/SSV 指令获取和设置对象中存储的控制器系统数据。

操作数：

继电器梯形图



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

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。与在 PLC-5 处理器中相同，没有状态文件。

启用后， GSV 指令检索指定的信息，并将这些信息放入目标中。
启用后， SSV 指令使用源中的数据设置指定的属性。

当您输入 GSV/SSV 指令时，编程软件为每个指令显示有效的对象类、对象名称和属性名称。对于 GSV 指令，您可以获取所有可用属性的值。对于 SSV 指令，该软件只显示允许设置的属性 (SSV)。

注意

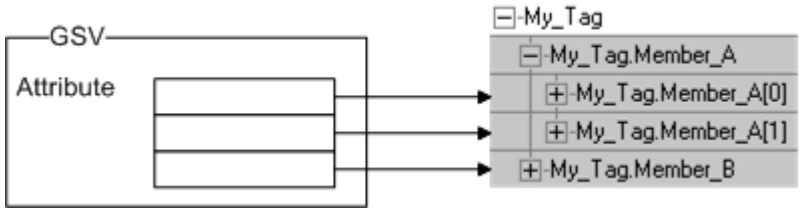


使用 GSV 和 SSV 指令时要小心。更改对象后可能导致出现意外的控制器操作或人身伤害。

您 **《必须》**测试并确认指令未更改您不希望更改的数据。

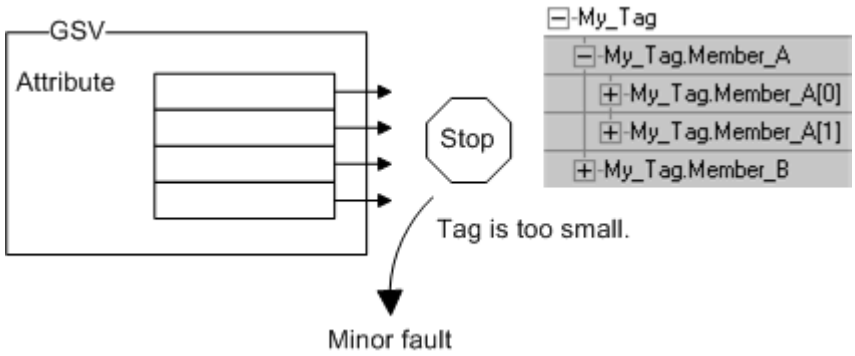
GSV 和 SSV 指令会超出标记的一个成员的范围而进入其他成员中读写。如果标记太小，则指令不写入或读取数据，而改为记录一个小错误。

示例 1



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

示例 2



对于该属性而言， My_Tag 太小。因此 GSV 指令停止，并记录一个小错误。

GSV/SSV 《对象》一节显示了每个对象的属性及其关联的数据类型。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

错误条件：

出现小错误的条件：	错误类型：	错误代码：
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 目标不够大，无法存放请求的数据	4	7

执行：

条件：	梯形图操作：	结构化文本操作：
扫描前	梯级输出条件设置为 false。	不执行任何操作
梯级输入条件为 false	梯级输出条件设置为 false。	无
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	无
EnableIn 已设置	无	EnableIn 始终为设置状态。 指令执行。
指令执行	获取或设置指定值。	获取或设置指定值。
扫描后	梯级输出条件设置为 false。	不执行任何操作。

GSV/SSV 对象

输入 GSV/SSV 指令时，您指定了要访问的对象及其属性。在有些情况下，同一类型的对象有多个实例，因此您可能还必须指定对象名称。例如，您的应用程序中可能有几个任务。每个任务都有自己的 TASK 对象，您可以通过任务名称访问该对象。

注意



对于 GSV 指令，只将指定大小的数据复制到目标。例如，如果属性被指定为 SINT，而目标为 DINT，则只更新 DINT 目标的低 8 位，其余 24 位保持不变。

您可以访问下列对象：

有关此对象的信息：	请参见此页面或出版物：
AXIS	*ControlLogix Motion Module Setup and Configuration Manual* （ControlLogix 运动模块安装和配置手册），出版号 1756-UM006
CONTROLLER	第 3 章 -37
CONTROLLERDEVICE	第 3 章 -37
CST	第 3 章 -39
DF1	第 3 章 -40
FAULTLOG	第 3 章 -43
MESSAGE	第 3 章 -44
MODULE	第 3 章 -46
MOTIONGROUP	第 3 章 -47
PROGRAM	第 3 章 -48
ROUTINE	第 3 章 -49
SERIALPORT	第 3 章 -49
TASK	第 3 章 -51
WALLCLOCKTIME	第 3 章 -53

访问 CONTROLLER 对象

CONTROLLER 对象提供控制器的执行状态信息。

属性：	数据类型：	指令：	说明：
TimeSlice	INT	GSV SSV	分配给通信活动的可用 CPU 百分比。 有效值为 10-90。当控制器按键开关处于运行位置时，不能更改该值。

访问 CONTROLLERDEVICE 对象

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	用于标识控制器和内存板的目录编号的 ASCII 字符串。 第一个字节包含数组字符串中返回的 ASCII 字符数的计数。
ProductCode	INT	GSV	标识控制器的类型
ProductRev	INT	GSV	标识当前产品版本。显示内容应该为十六进制值。 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。 序列号是在建造设备时分配的。

Logix 控制器：	产品代码：
CompactLogix5320	43
CompactLogix5330	44
CompactLogix5335E	65
ControlLogix5550	3
ControlLogix5553	50
ControlLogix5555	51
ControlLogix5561	54
ControlLogix5562	55
ControlLogix5563	56
DriveLogix5720	48
FlexLogix5433	41
FlexLogix5434	42
SoftLogix5860	15

属性：	数据类型：	指令：	说明：
Status	INT	GSV	位标识状态： 3-0 位已保留 **设备状态位** **7-4 位：含义：** 0000 保留 0001 正在进行闪存更新 0010 保留 0011 保留 0100 闪存错误 0101 发生错误 0110 运行 0111 程序 **错误状态位** **11-8 位：含义：** 0001 可恢复的小错误 0010 不可恢复的小错误 0100 可恢复的严重错误 1000 不可恢复的严重错误 **Logix5000 特定状态位** **13-12 位：含义：** 01 按键开关处于运行位置 10 按键开关处于程序位置 11 按键开关处于远程位置 **15-14 位 含义** 01 控制器正在更改模式 10 调试模式 （如果控制器处于运行模式）
类型	INT	GSV	将设备标识为控制器。 控制器 = 14
Vendor	INT	GSV	标识设备供应商。 Allen-Bradley = 0001

访问 CST 对象

CST （协调系统时间）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。位标识：	
			位：	含义
			0	计时器硬件发生错误：设备的内部计时器硬件处于出错状态
			1	启用斜升：计时器低 16+ 位的当前值斜升至请求的值，而不是向较低的值靠齐。这些位受网络特定时钟同步方法操纵。
			2	系统时间主控：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来同步
			4	本地网络主控：CST 对象是本地网络主时间源
			5	处于接替模式：CST 对象在时间接替模式下工作
			6	检测到重复的主节点：检测到重复的本地网络时间主节点。对于与时间有关的节点，该位始终为 0。
			7	未使用
			8-9	00 = 与时间有关的节点 01 = 时间主节点 10 = 时间接替节点 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。 计时器源调整为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 属性的报告，该调整操作或者为斜升到请求值，或者为立即设置为请求值。	

访问 DF1 对象

DF1 对象提供您可以为串行端口配置的 DF1 通信驱动程序的接口。

属性：	数据类型：	指令：	说明：
ACKTimeout	DINT	GSV	等待消息传输回应的时间 （仅限点对点传输和主传输）。有效值为 0-32,767。以 20 毫秒为一个时段所计算的延迟值。默认值为 50 （1 秒）。
DiagnosticCounters	INT[19]	GSV	DF1 通信驱动程序的诊断计数器数组。

字偏移	DF1 点对点	DF1 从节点	主节点
0	签名 (0x0043)	签名 (0x0042)	签名 (0x0044)
1	调制解调器位	调制解调器位	调制解调器位
2	发送的包数	发送的包数	发送的包数
3	收到的包数	收到的包数	收到的包数
4	未提交的包数	未提交的包数	未提交的包数
5	未使用	检索到的消息数	重试的消息数
6	收到的 NAK 数	收到的 NAK 数	未使用
7	收到查询数	收到的轮询包数	未使用
8	NAK 的坏包数	未确认的坏包数	未确认的坏包数
9	无内存发送 NAK	无内存未确认	未使用
10	收到的重复包数	收到的重复包数	收到的重复包数
11	收到的错误字符数	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	最大优先扫描时间
15	未使用	未使用	上次优先扫描时间
16	未使用	未使用	最大普通扫描时间
17	未使用	未使用	上次普通扫描时间
18	发送的查询数	未使用	未使用

DuplicateDetection	SINT	GSV	启用重复消息检测。 **值：** **含义：** 0 禁用重复消息检测 非零 禁用重复消息检测
EmbeddedResponseEnable	SINT	GSV	启用嵌入响应功能 （仅限点对点）。 **值：** **含义：** 0 仅在收到一个响应后启动 （默认设置） 1 无条件启用
ENQTransmitLimit	SINT	GSV	确认超时后要发送的查询 (ENQ) 的数目 （仅限点对点）。有效值 0-127。默认设置为 3。
EOTSuppression	SINT	GSV	在对轮询包的响应中启用 EOT 传输抑制 （仅限从节点）。 **值：** **含义：** 0 禁用 EOT 抑制 （禁用） 非零 启用 EOT 抑制
ErrorDetection	SINT	GSV	指定错误检测方案。 **值：** **含义：** 0 BCC （默认） 1 CRC

属性：	数据类型：	指令：	说明：
MasterMessageTransmit	SINT	GSV	主消息传输的当前值 （仅限主节点）。 **值：** **含义：** 0 在站轮询之间 1 按轮询顺序 （而不是按主节点的站编号） 默认值为 0。
NAKReceiveLimit	SINT	GSV	停止传输之前在消息响应中收到的 NAK 数 （仅限点对点通信）。 有效值 0-127。默认值为 3。
NormalPollGroupSize	INT	GSV	在轮询优先轮询节点数组中的所有站之后，要在普通轮询节点数组中轮询的站数 （仅限主节点）。 有效值 0-255。默认值为 0。
PollingMode	SINT	GSV	当前轮询模式 （仅限主节点）。 **值：** **含义：** 0 基于消息，但是不允许从节点启动消息 1 基于消息，但允许从节点启动消息 （默认） 2 标准，每次节点扫描传输一则消息 3 标准，每次节点扫描传输多则消息 默认设置为 1。
ReplyMessageWait	DINT	GSV	收到确认之后、轮询从节点以获取响应之前等待的时间（充当主控时间）（仅限主节点）。 有效值 0-65,535。以 20 毫秒为一个时段所计算的延迟。 默认值为 5 个时段 （100 毫秒）。
StationAddress	INT	GSV	串行端口的当前站地址。 有效值 0-254。默认值为 0。
SlavePollTimeout	DINT	GSV	在从节点声明由于主节点不活动而无法传输之前，从节点等待主节点轮询的时间 （以毫秒为单位）（仅限从节点）。 有效值 0-32,767。延迟以 20 毫秒为一个时段计算。默认值为 3000 个时段 （1 分钟）。
TransmitRetries	SINT	GSV	在未得到确认的情况下重新发送消息的次数 （仅限主节点和从节点）。 有效值 0-127。默认值为 3。
PendingACKTimeout	DINT	SSV	ACKTimeout 属性的未决值。
PendingDuplicateDetection	SINT	SSV	DuplicateDetection 属性的未决值。
PendingEmbeddedResponse Enable	SINT	SSV	EmbeddedResponse 属性的未决值。
PendingENQTransmitLimit	SINT	SSV	ENQTransmitLimit 属性的未决值。
PendingEOTSuppression	SINT	SSV	EOTSuppression 属性的未决值。
PendingErrorDetection	SINT	SSV	ErrorDetection 属性的未决值。
PendingNormalPollGroup Size	INT	SSV	NormalPollGroupSize 属性的未决值。
PendingMasterMessage Transmit	SINT	SSV	MasterMessageTransmit 属性的未决值。
PendingNAKReceiveLimit	SINT	SSV	NAKReceiveLimit 属性的未决值。

属性：	数据类型：	指令：	说明：
PendingPollingMode	SINT	SSV	PollingMode 属性的未决值。
PendingReplyMessageWait	DINT	SSV	ReplyMessageWait 属性的未决值。
PendingStationAddress	INT	SSV	StationAddress 属性的未决值。
PendingSlavePollTimeout	DINT	SSV	SlavePollTimeout 属性的未决值。
PendingTransmitRetries	SINT	SSV	TransmitRetries 属性的未决值。

要对任意 DF1 未决属性应用值，请执行下列操作：

1. 使用 SSV 指令设置未决属性的值。

   可以通过对每个未决属性使用 SSV 指令来设置任意数目的未决属性。

2. 使用 MSG 指令应用属性值。 MSG 指令应用您设置的每个未决属性。可按如下方式配置 MSG 指令：

MSG Configuration （MSG 配置）选项卡：	字段：	值：
Configuration （配置）	Message Type （消息类型）	CIP Generic （CIP 通用）
	Service Code （服务代码）	0d hex
	Object Type （对象类型）	a2
	Object ID （对象 ID）	1
	Object Attribute （对象属性）	留空
	Source （源）	留空
	Number of Elements （元素数目）	0
	Destination （目标）	留空
Communication （通信）	Path （路径）	自身的通信路径 （1,*s* 其中 *s* = 控制器的插槽编号）

访问 FAULTLOG 对象

FAULTLOG 对象提供有关控制器的错误信息。

属性：	数据类型：	指令：	说明：
MajorEvents	INT	GSV SSV	自上次重置此计数器以来发生的严重错误的数目。
MinorEvents	INT	GSV SSV	自上次重置此计数器以来发生的小错误的数目。
MajorFaultBits	DINT	GSV SSV	指明当前严重错误的原因的各个位。 **位： 含义：** 1 断电 3 I/O 4 指令执行 （程序） 5 错误处理程序 6 Watchdog 7 堆栈 8 模式更改 11 运动
MinorFaultBits	DINT	GSV SSV	指明当前小错误的原因的各个位。 **位： 含义：** 4 指令执行 （程序） 6 Watchdog 9 串行端口 10 电池

访问 MESSAGE 对象

可以通过 GSV/SSV 指令访问 MESSAGE 对象。指定消息标记名称以确定所需的 MESSAGE 对象。 MESSAGE 对象为设置和触发对等通信提供一个接口。此对象取代 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	连接的请求包速率。
MessageType	SINT	GSV SSV	指定消息类型。 **值：** **含义：** 0 不初始化
Port	SINT	GSV SSV	表示发送消息时应当使用的端口。 **值：** **含义：** 1 底板 2 串行端口
TimeoutMultiplier	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） 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
UnconnectedTimeout	DINT	GSV SSV	所有未连接消息的超时时段 （以毫秒为单位）。默认值为 30,000,000 毫秒 （30 秒）。

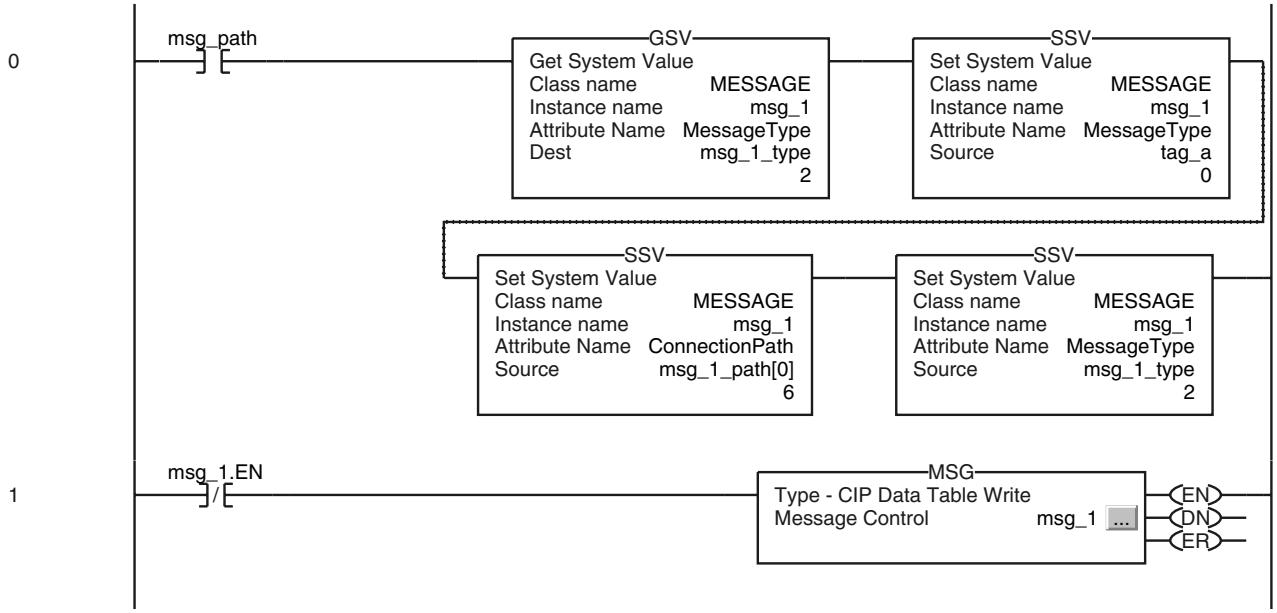
要更改 MESSAGE 属性，请按下列步骤操作：

1. 使用 GSV 指令获取 MessageType 属性并将其保存在标记中。
2. 使用 SSV 指令将 MessageType 设置为 0。
3. 使用 SSV 指令设置要更改的 MESSAGE 属性。
4. 使用 SSV 指令将 MessageType 属性重新设置为您在步骤 1 中获得的初始值。

示例： 下面的示例更改 ConnectionPath 属性，以便消息可以进入另一个控制器。当 *msg_path* 启用时，将 *msg_1* 消息的路径设置为 *msg_1_path* 的值。这会将消息发送到另一个控制器。

其中：	表示：
msg_1	您要更改其属性的消息
msg_1_type	存储 MessageType 属性值的标记
tag_a	存储 0 的标记。
msg_1_path	存储消息新连接路径的数组标记

继电器梯形图



结构化文本

```
IF msg_path THEN
     GSV(MESSAGE,msg_1,MessageType,msg_1_type);
     SSV(MESSAGE,msg_1,MessageType,tag_a);
     SSV(MESSAGE,msg_1,ConnectionPath,msg_1_path[0]);
     SSV(MESSAGE,msg_1,MessageType,msg_1_type);
END_IF;
IF NOT msg_1.EN THEN
     MSG(msg_1);
END_IF;
```

访问 MODULE 对象

MODULE 对象提供模块的状态信息。要选择特定 MODULE 对象，请将 GSV/SSV 指令的 Object Name （对象名称）操作数设置为模块名称。指定的模块必须出现在控制器组织程序的 I/O Configuration （I/O 配置）部分，且必须具有设备名称。

属性：	数据类型：	指令：	说明：
EntryStatus	INT	GSV	指定先前指定的映射条目的当前状态。执行比较操作时应当遮蔽低 12 位。只有 12-15 位有效。 **值：** **含义：** 16#0000 **待机：**控制器已通电。 16#1000 **发生错误：**MODULE 对象与关联模块的所有连接都失败。不应当使用该值来确定模块是否失败，因为 MODULE 对象在尝试重新连接到模块时周期性地保持此状态。应当改为测试运行状态 (16#4000)。检查 FaultCode 是否等于 0 可确定模块是否出错。发生错误后，如果不纠正错误情况，FaultCode 和 FaultInfo 属性将一直保持有效。 16#2000 **正在验证：**在与模块建立连接之前，MODULE 对象正在验证 MODULE 对象的完整性。 16#3000 **正在连接：**MODULE 对象正在启动与模块的连接。 16#4000 **正在运行**已建立与模块的所有连接，数据正在顺利传输。 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 （设置了 Mode 属性中的禁止位）。 16#7000 **正在等待：**此 MODULE 对象依赖的父 MODULE 对象未运行。
FaultCode	INT	GSV	标识模块错误 （如有错误）的数字。
FaultInfo	DINT	GSV	提供有关 MODULE 对象错误代码的特定信息。
ForceStatus	INT	GSV	指定强制状态。 **位：** **含义：** 0 已安装强制 （1= 是， 0= 否） 1 已启用强制 （1= 是， 0= 否） 2-15 未使用

属性：	数据类型：	指令：	说明：
Instance	DINT	GSV	提供此 MODULE 对象的实例编号。
LEDStatus	INT	GSV	指定控制器前面 I/O LED 的当前状态。 **值：** **含义：** 0 **LED 关闭：**没有为控制器配置 MODULE 对象 （在控制器组织程序的 I/O Configuration （I/O 配置）部分没有模块）。 1 红灯闪烁：没有 MODULE 对象运行。 2 绿灯闪烁：至少有一个 MODULE 对象未运行。 3 绿灯常亮：所有 Module 对象都正在运行。 **说明：**您不必输入具有此属性的对象名称，因为此属性适用于整个模块集合。
Mode （模式）	INT	GSV SSV	指定 MODULE 对象的当前模式。 **位：** **含义：** 0 设置后，如果在控制器处于运行模式时所有 MODULE 对象连接都发生错误，将导致生成严重错误。 2 如果设置，会使 MODULE 对象在关闭与模块的所有连接后进入 Inhibited （禁止）状态。

访问 MOTIONGROUP 对象

MOTIONGROUP 对象提供有关伺服模块的一组轴的状态信息。指定运动组标记名称以确定所需的 MOTIONGROUP 对象。

属性：	数据类型：	指令：	说明：
Instance （实例）	DINT	GSV	提供此 MOTION_GROUP 对象的实例编号。

访问 PROGRAM 对象

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：	指令：	说明：
DisableFlag	SINT	GSV SSV	控制此程序的执行。 **值：** **含义：** 0 启用执行 1 禁用执行
Instance （实例）	DINT	GSV	提供此 PROGRAM 对象的实例编号。
LastScanTime	DINT	GSV SSV	上次执行此程序所耗费的时间。时间以毫秒为单位。
MajorFaultRecord	DINT[11]	GSV SSV	为此程序记录严重错误 建议您创建用户定义的结构以简化对 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	故障的唯一代码 （取决于故障类型）
Info	DINT[8]	十六进制	故障特定的信息 （取决于故障类型和代码）
MaxScanTime	DINT	GSV SSV	为此程序记录的最长执行时间。时间以毫秒为单位。
MinorFaultRecord	DINT[11]	GSV SSV	为此程序记录小错误 建议您创建用户定义的结构以简化对 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	故障的唯一代码 （取决于故障类型）
Info	DINT[8]	十六进制	故障特定的信息 （取决于故障类型和代码）
SFCRestart	INT	GSV SSV	未使用 - 留待将来使用

访问 ROUTINE 对象

ROUTINE 对象提供有关例程的状态信息。指定例程名称以确定所需的 ROUTINE 对象。

属性：	数据类型：	指令：	说明：
Instance （实例）	DINT	GSV	提供此 ROUTINE 对象的例程编号。 有效值为 0-65,535。

访问 SERIALPORT 对象

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）。
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）
Parity	SINT	GSV	指定奇偶校验。 **值： 含义：** 0 不进行奇偶校验 （无默认值） 1 奇校验 （仅限 ASCII） 2 偶校验
RTSOffDelay	INT	GSV	传输完最后一个字符后延迟关闭 RTS 行的时间。 有效值 0-32,767。以 20 毫秒为一个时段所计算的延迟。 默认值为 0 毫秒。
RTSSendDelay	INT	GSV	打开 RTS 行后延迟传输消息的第一个字符的时间。 有效值 0-32,767。以 20 毫秒为一个时段所计算的延迟。 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 属性的未决值。
PendingDataBits	SINT	SSV	DataBits 属性的未决值。
PendingParity	SINT	SSV	Parity 属性的未决值。

属性：	数据类型：	指令：	说明：
PendingRTSOffDelay	INT	SSV	RTSOffDelay 属性的未决值。
PendingRTSSendDelay	INT	SSV	RTSSendDelay 属性的未决值。
PendingStopBits	SINT	SSV	StopBits 属性的未决值。

要对任意 SERIALPORT 未决属性应用值，请执行下列操作：

1. 使用 SSV 指令设置未决属性的值。

   您可以通过对每个未决属性使用 SSV 指令来设置任意数目的未决属性。

2. 使用 MSG 指令应用属性值。 MSG 指令应用您设置的每个未决属性。按如下方式配置 MSG 指令：

MSG Configuration （MSG 配置）选项卡：	字段：	值：
Configuration （配置）	Message Type （消息类型）	CIP Generic （CIP 通用）
	Service Code （服务代码）	0d （十六进制）
	Object Type （对象类型）	6f （十六进制）
	Object ID （对象 ID）	1
	Object Attribute （对象属性）	留空
	Source （源）	留空
	Number of Elements （元素数目）	0
	Destination （目标）	留空
Communication （通信）	Path （路径）	自身的通信路径 （1,*s* 其中 *s* = 控制器的插槽编号）

访问 TASK 对象

TASK 对象提供有关任务的状态信息。指定任务名称以确定所需的 TASK 对象。

属性：	数据类型：	指令：	说明：	
DisableUpdateOutputs	DINT	GSV SSV	在任务结束时启用或禁用对输出的处理	
			操作目的：	**将该属性设置为：**
			在任务结束时启用对输出的处理	0
			在任务结束时禁用对输出的处理	1 （或任意非零值）
EnableTimeOut	DINT	GSV SSV	启用或禁用事件任务的超时功能。	
			操作目的：	**将该属性设置为：**
			禁用超时功能	0
			启用超时功能	1 （或任意非零值）
InhibitTask	DINT	GSV SSV	阻止任务执行。如果禁止某个任务，控制器在从程序模式转换为运行模式或测试模式时仍预先扫描该任务。	
			操作目的：	**将该属性设置为：**
			启用任务	0 （默认值）
			禁止 （禁用）任务	1 （或任意非零值）
Instance （实例）	DINT	GSV	提供此 TASK 对象的实例编号。 有效值为 0-31。	
LastScanTime	DINT	GSV SSV	上次执行此任务所耗费的时间。时间以毫秒为单位。	
MaxInterval	DINT[2]	GSV SSV	任务后续执行之间的最大时间间隔。 DINT[0] 存放该值的低 32 位； DINT[1] 存放该值的高 32 位。 值为 0 指示该任务执行了一次或更少次。	
MaxScanTime	DINT	GSV SSV	为此程序记录的最长执行时间。时间以毫秒为单位。	
MinInterval	DINT[2]	GSV SSV	任务后续执行之间的最小时间间隔。 DINT[0] 存放该值的低 32 位； DINT[1] 存放该值的高 32 位。 值为 0 表示该任务执行了一次或更少次。	
OverlapCount	DINT	GSV SSV	在任务仍在执行时触发该任务的次数。对事件或定期任务有效。 要清除计数，请将该属性设置为 0。	
Priority （优先级）	INT	GSV SSV	此任务与其他任务比较后的相对优先级。 有效值为 0-15。	

属性：	数据类型：	指令：	说明：	
Rate （比率）	DINT	GSV SSV	**如果任务类型为：**	**则 Rate （比率）属性指定：**
			定期	任务的时段。时间以毫秒为单位。
			EVENT	任务的超时值。时间以毫秒为单位。
StartTime	DINT[2]	GSV SSV	上次启动任务时 WALLCLOCKTIME 的值。 DINT[0] 存放该值的低 32 位； DINT[1] 存放该值的高 32 位。	
状态	DINT	GSV SSV	提供任务的状态信息。一旦控制器设置了这些位之一，您必须手动清除该位。	
			要确定是否：	**检查位：**
			任务由 EVNT 指令触发（仅限事件任务）。	0
			任务由超时时间限制触发（仅限事件任务）。	1
			该任务发生重叠。	2
Watchdog	DINT	GSV SSV	执行与此任务相关的所有程序的时间限制。时间以毫秒为单位。 如果输入 0，将分配以下值：	
			时间：	**任务类型：**
			0.5 秒	定期或事件
			5.0 秒	连续

访问 WALLCLOCKTIME 对象

WALLCLOCKTIME 对象提供有关控制器可以用于计划的时间戳。

属性：	数据类型：	指令：	说明：
CSTOffset	DINT[2]	GSV SSV	距离 CST 对象的 CurrentValue 的正偏移 （协调系统时，参见第 第 3 章 -39 页）。DINT[0] 存放该值的低 32 位； DINT[1] 存放该值的高 32 位。 值以微秒为单位 m。默认值为 0。
CurrentValue	DINT[2]	GSV SSV	时钟时间的当前值。 DINT[0] 存放该值的低 32 位；DINT[1] 存放该值的高 32 位。 该值是自 1972 年 1 月 1 日 0000 时以来经过的毫秒数。 CST 和 WALLCLOCKTIME 对象在控制器中数学相关。例如，如果添加 CST CurrentValue 和 WALLCLOCKTIME CTSOffset，则结果为 WALLCLOCKTIME CurrentValue。
DateTime	DINT[7]	GSV SSV	可读格式的日期和时间。 DINT[0] 年 DINT[1] 月的整数表示 (1-12) DINT[2] 日期的整数表示 (1-31) DINT[3] 小时 (0-23) DINT[4] 分钟 (0-59) DINT[5] 秒 (0-59) DINT[6] 微秒 (0-999,999)

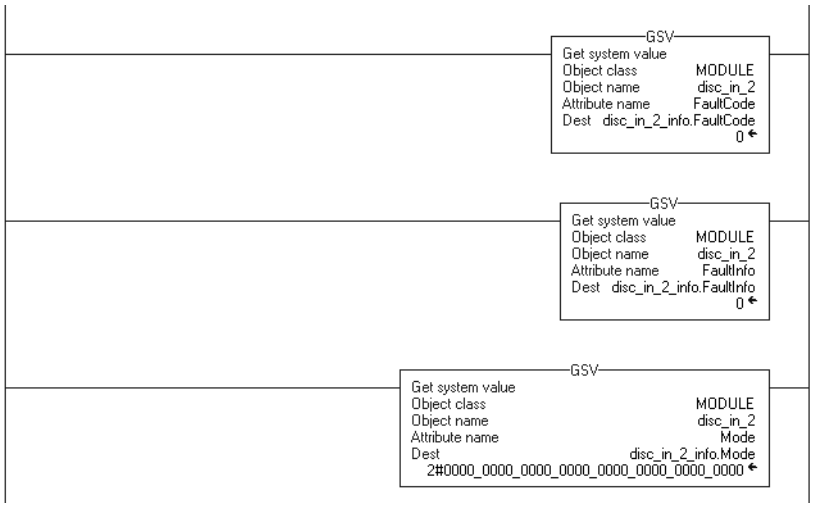
GSV/SSV 编程示例

获取错误信息

下面的示例使用 GSV 指令获取错误信息。

示例 1： 此示例从 I/O 模块 *disc_in_2* 获取错误信息，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

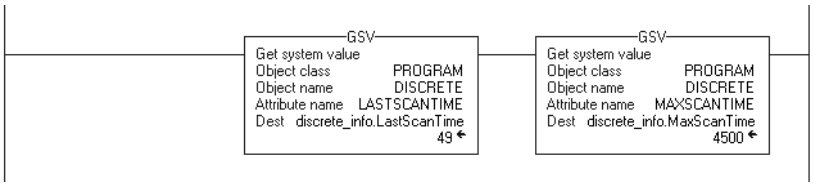
```
GSV(MODULE,disc_in_2,FaultCode,disc_in_2_info.FaultCode);

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。

继电器梯形图



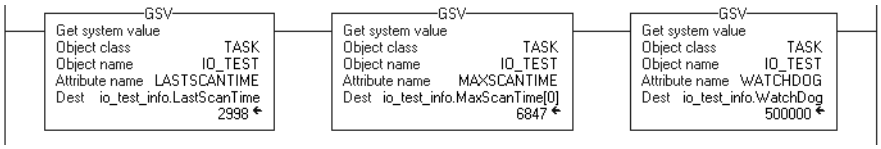
结构化文本

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

GSV(PROGRAM,DISCRETE,MAXSCANTIME,discrete_info.MaxScanTime);
```

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。

继电器梯形图



结构化文本

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

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

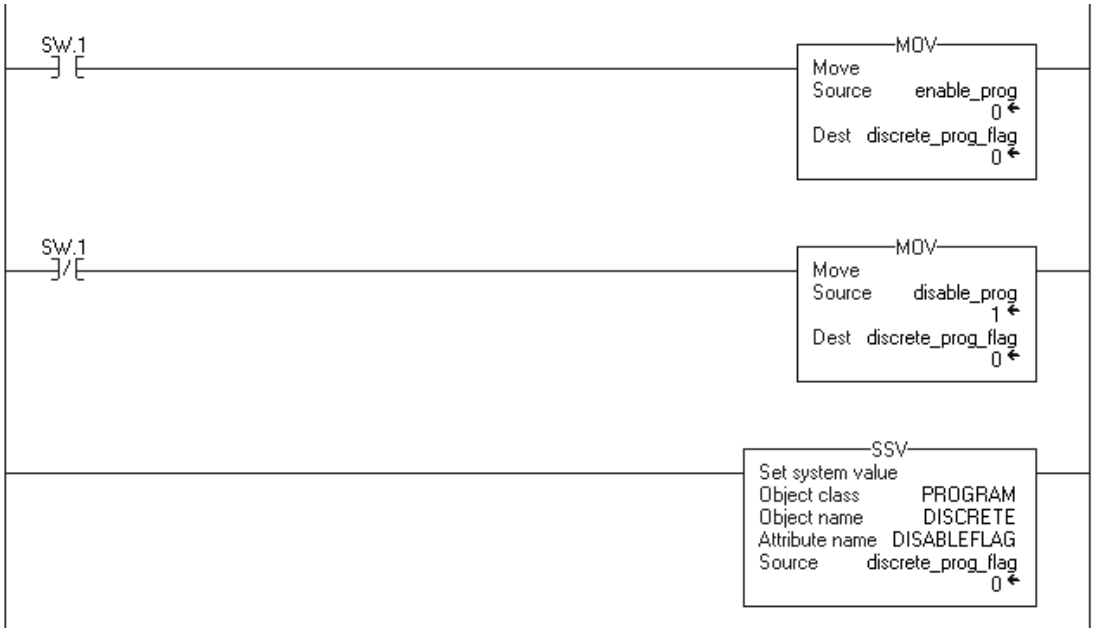
GSV(TASK,IO_TEST,WATCHDOG,io_test_info.WatchDog);
```

设置启用和禁用标记

下面的示例使用 SSV 指令启用或禁用程序。您还可以使用此方法启用或禁用 I/O 模块，这与对 PLC-5 处理器使用禁止位类似。

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

继电器梯形图



结构化文本

```
IF SW.1 THEN

     discrete_prog_flag := enable_prog;

ELSE

     discrete_prog_flag := disable_prog;

END_IF;

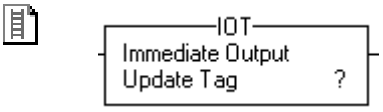
SSV(PROGRAM,DISCRETE,DISABLEFLAG,discrete_prog_flag);
```

立即输出 (Immediate Output, IOT)

IOT 指令立即更新指定的输出数据 （输出标记或生成的标记）。

操作数：

梯形图



操作数：	类型：	格式：	说明：
更新标记		标记	您想要更新的标记： • I/O 模块的输出标记 • 生成的标记 《不要》选择标记的成员或元素。例如， Local:5:0 正确，但是 Local:5:0.Data 《不》正确。

结构化文本

```
IOT(output_tag);
```

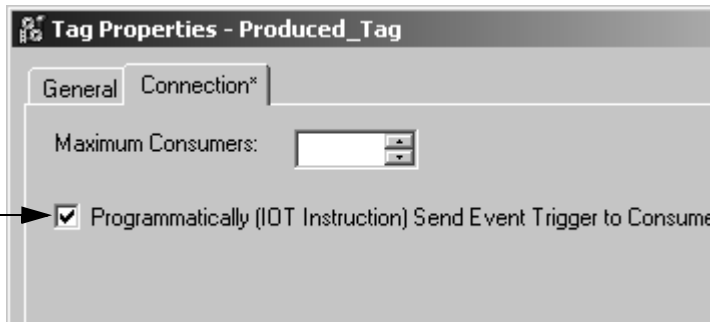
操作数与梯形图 IOT 指令的操作数相同。

说明： IOT 指令重写输出连接的请求包时间间隔 (RPI)，并通过该连接发送新数据。

- 输出连接是与 I/O 模块的输出标记或与生成的标记关联的连接。
- 如果该连接用于生成的标记，则 IOT 指令还将事件触发程序发送到使用控制器。这使 IOT 指令可以在使用控制器中触发事件任务。

要使用 IOT 指令和生成的标记在使用控制器中触发事件任务，请按如下方式配置生成的标记：

选中此框。
这会将标记配置为仅通过 IOT 指令更新其事件触发程序。

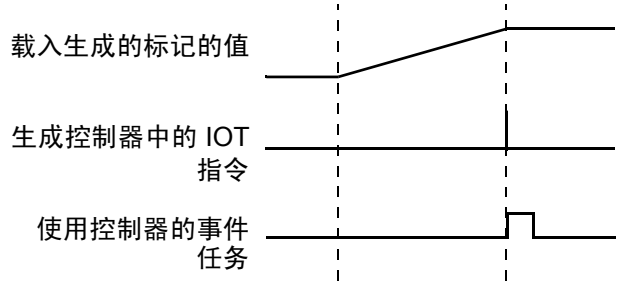


控制器之间的网络类型决定了使用控制器何时通过 IOT 指令接收新数据和事件触发程序。

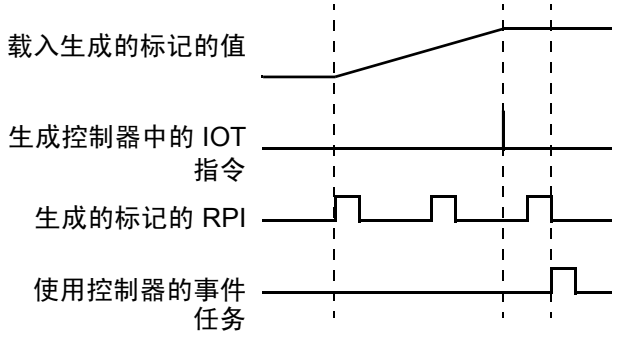
借助此控制器：	通过此网络：	使用设备接收数据和事件触发程序的时间：
ControlLogix	底板	立即
	EtherNet/IP 网络	立即
	ControlNet 网络	在使用的标记 （连接）的实际包间隔之内
SoftLogix5800	只能通过 ControlNet 网络生成和使用标记。	在使用的标记 （连接）的实际包间隔之内

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

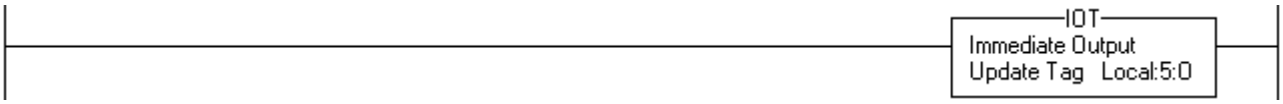
错误条件： 无

执行：

条件：	梯形图操作：	结构化文本操作：
扫描前	梯级输出条件设置为 false。	不执行任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	无
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	无
EnableIn 已设置	无	EnableIn 始终为设置状态。 指令执行。
指令执行	指令： • 更新指定的标记的连接。 • 重置连接的 RPI 计时器	
扫描后	梯级输出条件设置为 false。	不执行任何操作。

示例 1： 当 IOT 指令执行时，它立即将 *Local:5:0* 标记的值发送到输出模块。

梯形图

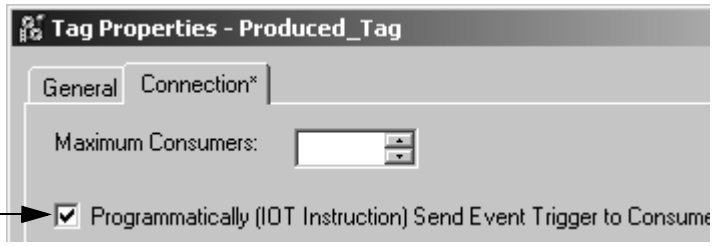


结构化文本

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

示例 2： 此控制器控制第 24 站，并为下一个站 （第 25 站）生成数据。要使用 IOT 指令来指示新数据的传输，请按如下方式配置生成的标记：

Produced_Tag 配置为通过 IOT 指令更新其事件触发程序。

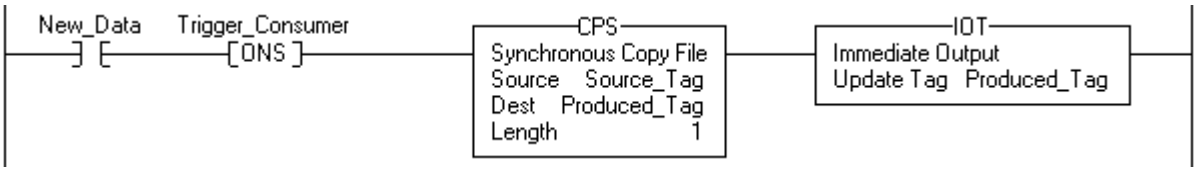


梯形图

如果 *New_Data* = 打开，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 指令更新 *Produced_Tag*，并将此更新发送到使用控制器 （第 25 站）。当使用控制器收到此更新时，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，可以使用表达式或特定的比较指令来比较值。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
基于表达式来比较值	CMP	梯形图 结构化文本(1)	第 4 章 -2
测试两个值是否相等	EQU	梯形图 结构化文本(2) 功能块	第 4 章 -7
测试某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	第 4 章 -11
测试某个值是否大于另一个值	GRT	梯形图 结构化文本 (1) 功能块	第 4 章 -15
测试某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	第 4 章 -19
测试某个值是否小于另一个值	LES	梯形图 结构化文本 (1) 功能块	第 4 章 -23
测试某个值是否在另外两个值之间	LIM	梯形图 结构化文本 (1) 功能块	第 4 章 -27
通过一个掩码传递两个值，并测试它们是否相等	MEQ	梯形图 结构化文本 (1) 功能块	第 4 章 -33
测试某个值是否不等于另一个值	NEQ	梯形图 结构化文本 (1) 功能块	第 4 章 -38

(1) 没有等效的结构化文本指令。使用其他结构化文本进行编程获得同样的结果。请参见该指令的说明。

(2) 没有等效的结构化文本指令。在表达式中使用运算符。

可以比较不同数据类型的值，如浮点数和整数。

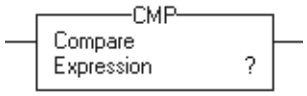
对于梯形图指令，**粗体**数据类型表示最佳数据类型。如果指令的所有操作数都使用同样的最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

比较 (Compare, CMP)

CMP 指令对表达式中指定的算术运算执行比较运算。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Expression（表达式）	SINT INT **DINT** **REAL** **string**	立即数 标记	表达式由标记和 / 或由运算符分开的立即值组成
	SINT 或 INT 标记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，但您可以使用 IF...THEN 结构和表达式实现相同的结果。

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

有关结构化文本内的结构和表达式的语法的信息，请参见“附录 C”。

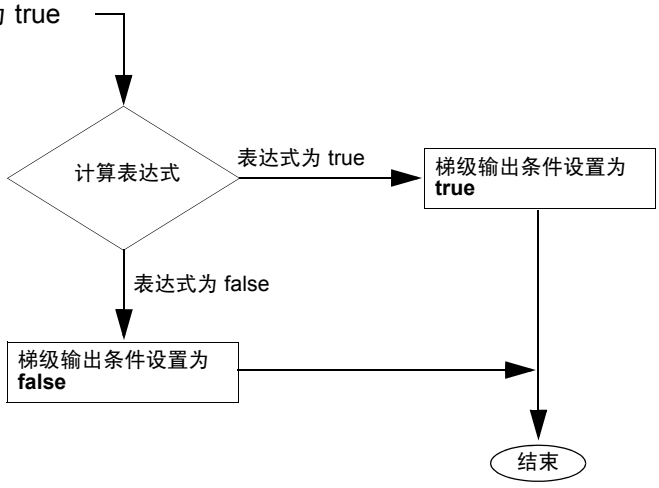
说明： 使用运算符、标记和立即数定义 CMP 表达式。使用括号 ( ) 定义更复杂的表达式的各部分。

CMP 指令执行起来比其他比较指令要稍微慢一些，使用的内存也比其他比较指令多。 CMP 指令的优点在于它允许您在一个指令中输入复杂的表达式。

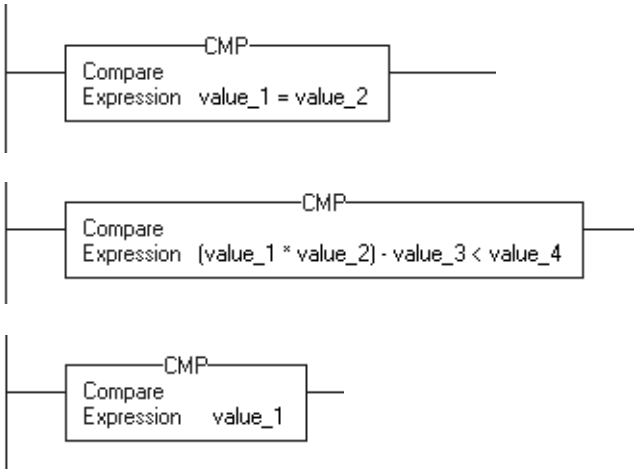
算术状态标志： 只有当表达式中包含影响算术状态标志的运算符（如 +、 –、 *、 /）时，CMP 指令才会影响算术状态标志。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

示例： 如果 CMP 指令发现表达式为 true，则梯级输出条件设置为 true。



如果输入不带比较运算符的表达式，如 *value_1 + value_2* 或 *value_1*，则该指令将如下所述计算表达式：

如果表达式为：	则梯级输出条件设置为：
非零	true
零	false

CMP 表达式

CMP 指令中的程序表达式与 FSC 指令中的程序表达式相同。下面几节内容提供了有关有效的运算符、格式、运算顺序的信息，这些说明对于这两种指令都适用。

有效运算符

运算符：	说明：	最优：
+	加	DINT、 REAL
-	减 / 求负	DINT、 REAL
*	乘	DINT、 REAL
/	除	DINT、 REAL
=	等于	DINT、 REAL
<	小于	DINT、 REAL
<=	小于或等于	DINT、 REAL
>	大于	DINT、 REAL
>=	大于或等于	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：	最优：
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

格式化表达式

对于表达式中使用的每个运算符，必须提供一个或两个操作数 （标记或立即值）。请根据下表设置表达式中运算符和操作数的格式：

运算符涉及的操作数的个数：	使用格式：	示例：
1	运算符 (操作数)	ABS(*tag_a*)
2	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*)

确定运算次序

指令按预定的次序执行写入表达式中的运算 （不一定是写入次序）。可以使用括号将运算条目分组，强制指令在执行其他运算之前先执行括号中的运算，从而更改运算次序。

次序相等的运算从左向右执行。

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

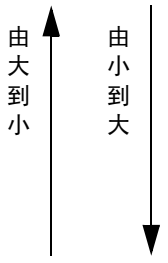
在表达式中使用字符串

可以使用梯形图或结构化文本表达式比较字符串数据类型。要在表达式中使用字符串，请遵循以下准则：

- 表达式可以比较两个字符串标记。
- 不能 在表达式中直接输入 ASCII 字符。
- 仅允许使用以下运算符

运算符：	说明：
=	等于
<	小于
<=	小于或等于
>	大于
>=	大于或等于
<>	不等于

- 如果两个字符串的字符相匹配，则这两个字符串相等。
- 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一个字符串是否小于或大于另一个字符串。要获知某字符的十六进制代码，请参见本手册的封底。
- 如果两个字符串按电话目录的方式排序，则字符串的顺序决定字符串的大小。



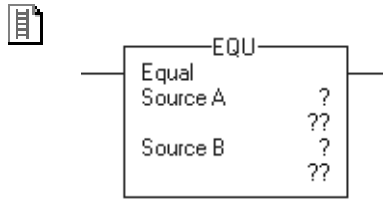
ASCII 字符	十六进制代码	
1ab	$31$61$62	
1b	$31$62	
A	$41	
AB	$41$42	AB < B
B	$42	
a	$61	a > B
ab	$61$62	

等于 (Equal to, EQU)

EQU 指令测试 Source A 是否等于 Source B。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source B 进行测试的值
Source B	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source A 进行测试的值

- 如果您输入 SINT 或 INT 标记，该值将通过符号扩展转换为 DINT 值。
- REAL 值很少完全相等。如果需要确定两个 REAL 值是否相等，可使用 LIM 指令。
- 字符串数据类型为：

  衆 t 默认 STRING 数据类型

  衆 t 您创建的任何新的字符串数据类型
- 要测试字符串的字符，请为 Source A 和 Source B 都输入一个字符串标记。

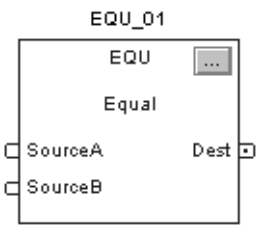
结构化文本

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

在表达式中将等号 “=”用作运算符。该表达式计算 *sourceA* 是否等于 *sourceB*。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
EQU 标记	FBD_COMPARE	结构	EQU 结构

FBD_COMPARE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	针对 SourceB 进行测试的值。 任何浮点数都有效
SourceB	REAL	针对 SourceA 进行测试的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。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 ASCII 字符串。比较字符串时：

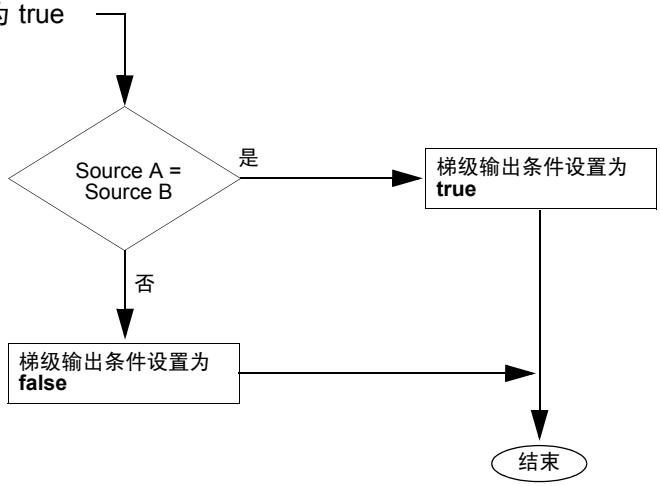
- 如果字符串的字符都一致，则字符串相等。
- ASCII 字符是区分大小写的。大写“A”($41) *不*等于小写“a”($61)。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 如果 *value_1* 等于 *value_2*，则设置 *light_a*。如果 *value_1* 不等于 *value_2*，则清除 *light_a*。

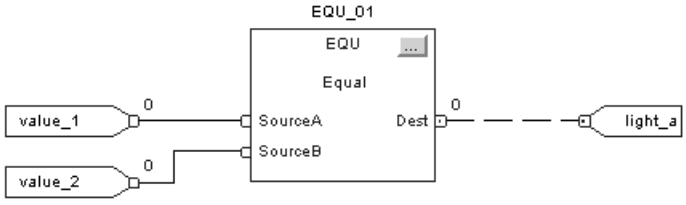
梯形图



结构化文本

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

功能块

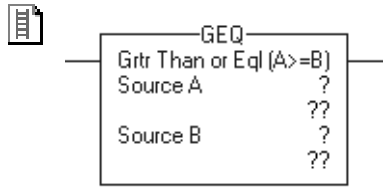


大于或等于 (Greater than or Equal to, GEQ)

GEQ 指令检查 Source A 是否大于或等于 Source B。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source B 进行测试的值
Source B	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source A 进行测试的值

- 如果您输入 SINT 或 INT 标记，该值将通过符号扩展转换为 DINT 值。
- 字符串数据类型为：

  衆 t 默认 STRING 数据类型

  衆 t 您创建的任何新的字符串数据类型
- 要测试字符串的字符，请为 Source A 和 Source B 都输入一个字符串标记。

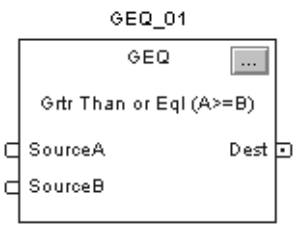
结构化文本

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

在表达式中使用相邻的大于号和等于号 “>=”作为运算符。该表达式计算 *sourceA* 是否大于或等于 *sourceB*。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
GEQ 标记	FBD_COMPARE	结构	GEQ 结构

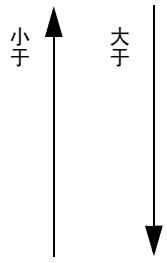
FBD_COMPARE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	针对 SourceB 进行测试的值。 任何浮点数都有效
SourceB	REAL	针对 SourceA 进行测试的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。这等效于梯形图 GEQ 指令的梯级输出条件。

说明： GEQ 指令检查 Source A 是否大于或等于 Source B。

比较字符串时：

- 字符的十六进制值确定两个字符串之间的大小关系。要查找字符的十六进制代码，请参见本手册的封底。
- 当两个字符串按在电话号码簿中的方式排序时，字符串的顺序决定了哪一个是较大的。



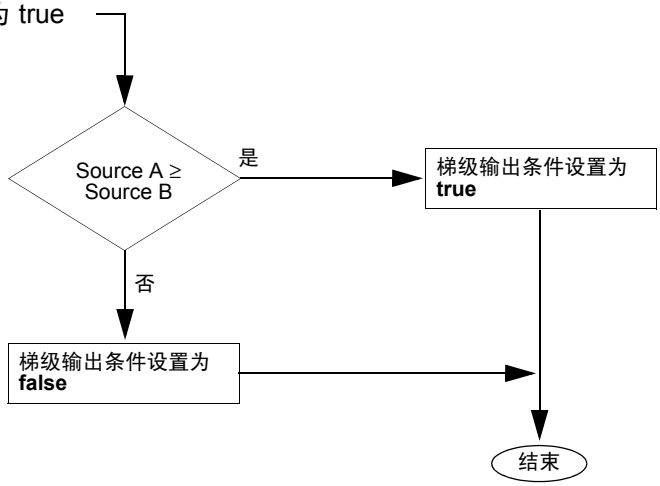
ASCII 字符	十六进制代码	
1ab	$31$61$62	
1b	$31$62	
a	$41	
AB	$41$42	AB < B
B	$42	
a	$61	a > B
AB	$61$62	

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 如果 *value_1* 大于或等于 *value_2*，则设置 *light_b*。如果 *value_1* 小于 *value_2*，则清除 *light_b*。

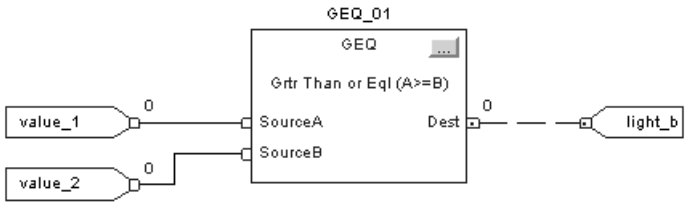
梯形图



结构化文本

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

功能块

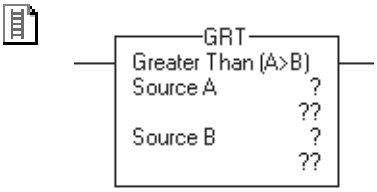


大于 (Greater Than, GRT)

GRT 指令测试 Source A 是否大于 Source B。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source B 进行测试的值
Source B	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source A 进行测试的值

- 如果您输入 SINT 或 INT 标记，该值将通过符号扩展转换为 DINT 值。
- 字符串数据类型为：

  录 t 默认 STRING 数据类型

  录 t 您创建的任何新的字符串数据类型
- 要测试字符串的字符，请为 Source A 和 Source B 都输入一个字符串标记。

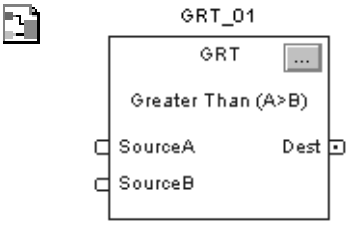
结构化文本

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

在表达式中使用大于号 “>”作为运算符。该表达式计算 *sourceA* 是否大于 *sourceB*。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
GRT 标记	FBD_COMPARE	结构	GRT 结构

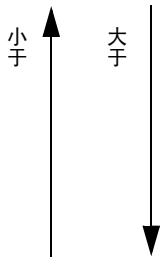
FBD_COMPARE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	针对 SourceB 进行测试的值。 任何浮点数都有效
SourceB	REAL	针对 SourceA 进行测试的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。这等效于梯形图 GRT 指令的梯级输出条件。

说明： GRT 指令测试 Source A 是否大于 Source B。

比较字符串时：

- 字符的十六进制值确定两个字符串之间的大小关系。要查找字符的十六进制代码，请参见本手册的封底。
- 当两个字符串按在电话号码簿中的方式排序时，字符串的顺序决定了哪一个是较大的。



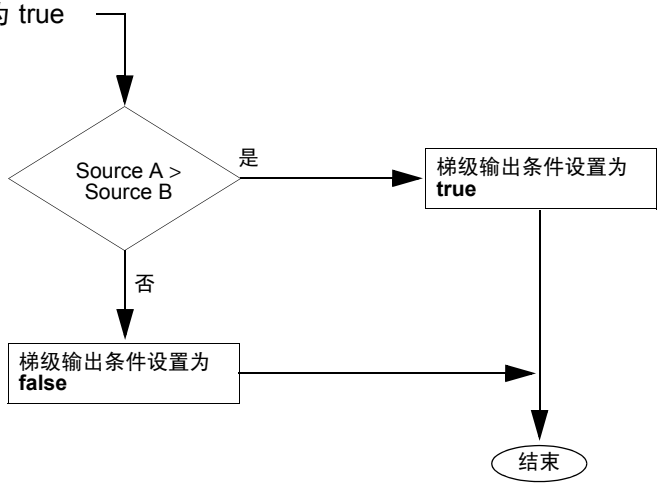
ASCII 字符	十六进制代码	
1ab	$31$61$62	
1b	$31$62	
a	$41	
AB	$41$42	AB < B
B	$42	
a	$61	a > B
AB	$61$62	

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 如果 *value_1* 大于 *value_2*，则设置 *light_1*。如果 *value_1* 小于或等于 *value_2*，则清除 *light_1*。

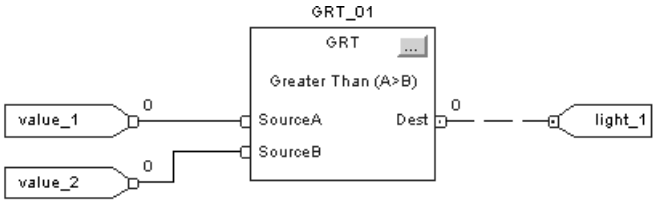
梯形图



结构化文本

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

功能块

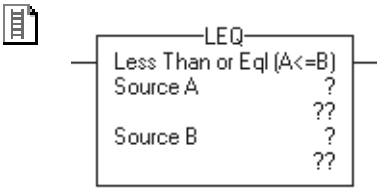


小于或等于 (Less Than or Equal to, LEQ)

LEQ 指令测试 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：
Source A	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source B 进行测试的值
Source B	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source A 进行测试的值

- 如果您输入 SINT 或 INT 标记，该值将通过符号扩展转换为 DINT 值。
- 字符串数据类型为：

  t 默认 STRING 数据类型

  t 您创建的任何新的字符串数据类型
- 要测试字符串的字符，请为 Source A 和 Source B 都输入一个字符串标记。

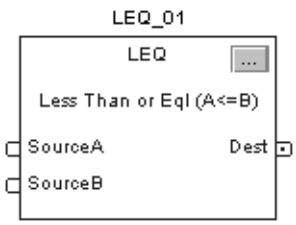
结构化文本

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

在表达式中使用相邻的小于号和等于号 “<=”作为运算符。该表达式计算 *sourceA* 是否小于或等于 *sourceB*。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
LEQ 标记	FBD_COMPARE	结构	LEQ 结构

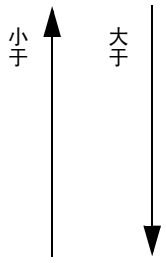
FBD_COMPARE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	针对 SourceB 进行测试的值。 任何浮点数都有效
SourceB	REAL	针对 SourceA 进行测试的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。这等效于梯形图 LEQ 指令的梯级输出条件。

说明： LEQ 指令测试 Source A 是否小于或等于 Source B。

比较字符串时：

- 字符的十六进制值确定两个字符串之间的大小关系。要查找字符的十六进制代码，请参见本手册的封底。
- 当两个字符串按在电话号码簿中的方式排序时，字符串的顺序决定了哪一个是较大的。



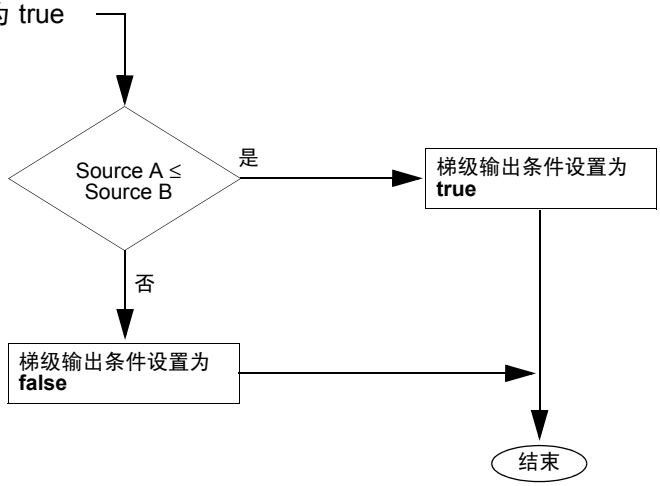
ASCII 字符	十六进制代码	
1ab	$31$61$62	
1b	$31$62	
a	$41	
AB	$41$42	AB < B
B	$42	
a	$61	a > B
AB	$61$62	

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 如果 *value_1* 小于或等于 *value_2*，则设置 *light_2*。如果 *value_1* 大于 *value_2*，则清除 *light_2*。

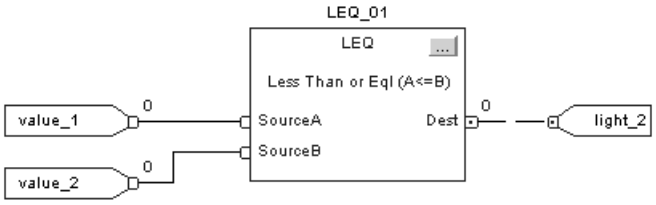
梯形图



结构化文本

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

功能块

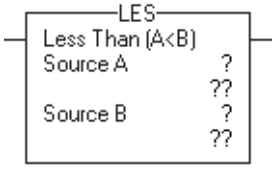


小于 (Less Than, LES)

LES 指令检查 Source A 是否小于 Source B。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source B 进行测试的值
Source B	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source A 进行测试的值

- 如果您输入 SINT 或 INT 标记，该值将通过符号扩展转换为 DINT 值。
- 字符串数据类型为：

  录 t 默认 STRING 数据类型
- 您创建的任何新的字符串数据类型
- 要测试字符串的字符，请为 Source A 和 Source B 都输入一个字符串标记。

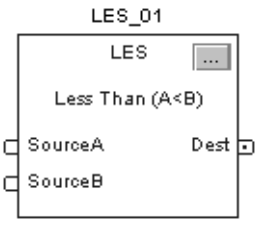
结构化文本

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

在表达式中使用小于号 “<”作为运算符。该表达式计算 *sourceA* 是否小于 *sourceB*。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
LES 标记	FBD_COMPARE	结构	LES 结构

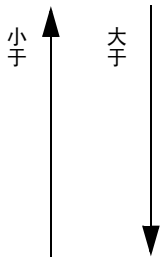
FBD_COMPARE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	针对 SourceB 进行测试的值。 任何浮点数都有效
SourceB	REAL	针对 SourceA 进行测试的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。这等效于梯形图 LES 指令的梯级输出条件。

说明： LES 指令检查 Source A 是否小于 Source B。

比较字符串时：

- 字符的十六进制值确定两个字符串之间的大小关系。要查找字符的十六进制代码，请参见本手册的封底。
- 当两个字符串按在电话号码簿中的方式排序时，字符串的顺序决定了哪一个是较大的。



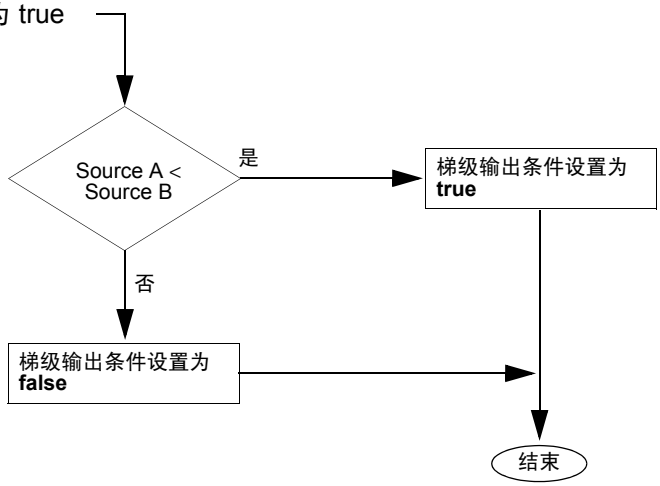
ASCII 字符	十六进制代码	
1ab	$31$61$62	
1b	$31$62	
a	$41	
AB	$41$42	AB < B
B	$42	
a	$61	a > B
AB	$61$62	

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 为 false	清除 EnableOut。
EnableIn 为 true	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 如果 *value_1* 小于 *value_2*，则设置 *light_3*。如果 *value_1* 大于或等于 *value_2*，则清除 *light_3*。

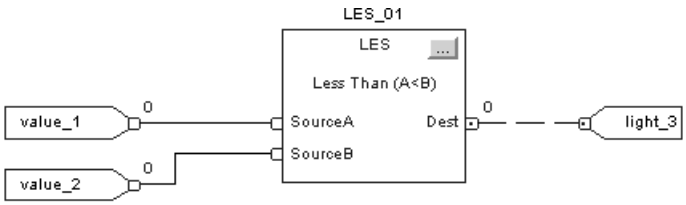
梯形图



结构化文本

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

功能块

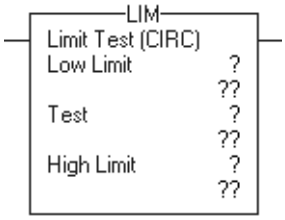


限制 (Limit, LIM)

LIM 指令测试 Test 值是否在 Low Limit 到 High Limit 的范围内。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Low limit	SINT INT **DINT** **REAL**	立即数 标记	下限值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Test	SINT INT **DINT** **REAL**	立即数 标记	要测试的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
High limit	SINT INT **DINT** **REAL**	立即数 标记	上限值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		

结构化文本

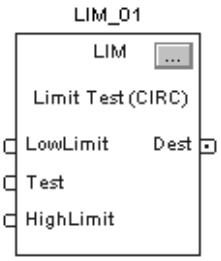
结构化文本没有 LIM 指令，但您可以使用结构化文本实现相同的结果。

```
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 标记	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，则不执行指令，也不更新输出。 如果设置，则该指令以 “执行”下所描述的方式执行。 默认为置位状态。
LowLimit	REAL	下限值。 任何浮点数都有效
Test	REAL	要测试是否在限制范围内的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。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

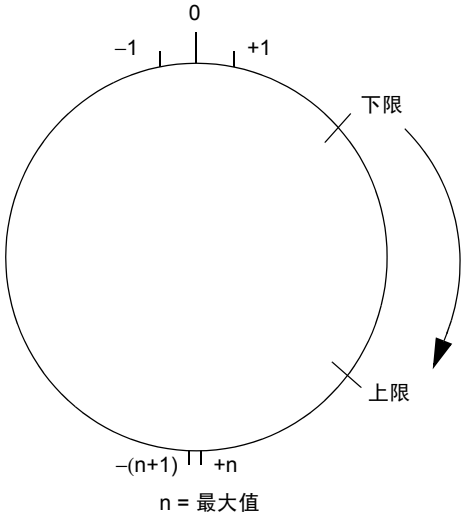
说明： LIM 指令测试 Test 值是否在 Low Limit 到 High Limit 的范围内。

如果 Low Limit：	且 Test 值：	则梯级输出条件为：
≤ High Limit	等于界限值或在界限值内	true
	不等于界限值或在界限值外	false
≥ High Limit	等于界限值或在界限值外	true
	不等于界限值或在界限值内	false

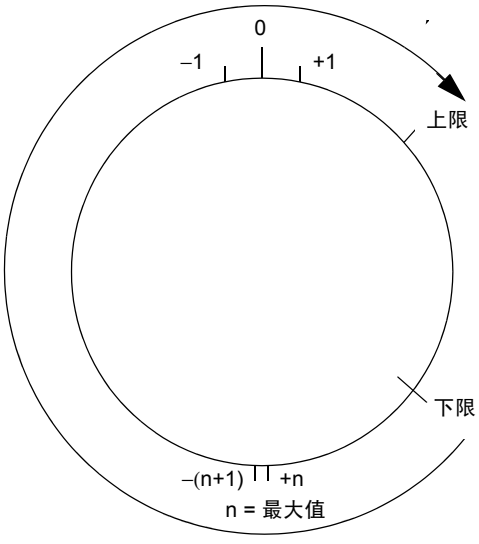
当设置了最有效位时，带符号的整数从最大正数反转为最大负数。例如，在 16 位整数 （INT 类型）中，最大正整数为 32,767，这在十六进制中表示为 16#7FFF （第 0 位到第 14 位均为置位状态）。如果在该数值的基础上加一，则结果为 16#8000 （第 15 位为置位状态）。对于带符号的整数，十六进制数 16#8000 等于十进制数 -32,768。如果从该数值递增直到全部 16 位均为置位状态，则最终结果为 16#FFFF，它等于十进制数 -1。

这可以显示为循环数字线 （见下图）。 LIM 指令从 Low Limit 开始，按顺时针方向递增，直到到达 High Limit。从 Low Limit 到 High Limit 的顺时针范围内的任何 Test 值都将梯级输出条件设置为 true。从 High Limit 到 Low Limit 的顺时针范围内的任何 Test 值都将梯级输出条件设置为 false。

Low Limit ≤ High Limit
如果测试值等于界限值或在上下限之间，则该指令为 true



Low Limit ≥ High Limit
如果测试值等于界限值或不在上下限范围内，则该指令为 true。

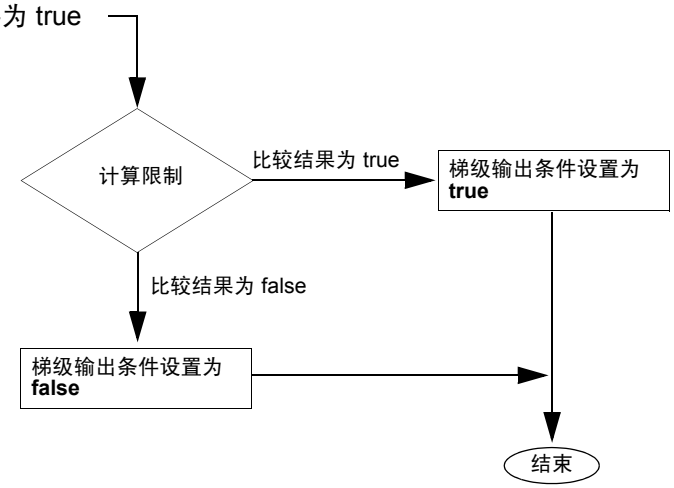


算术状态标志： 不受影响

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut，指令不执行任何操作，也不更新输出。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例 1： Low Limit ≤ High Limit:

当 0 ≤ *value* ≥ 100 时，设置 *light_1*。如果 *value* < 0 或 *value* >100，则清除 *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

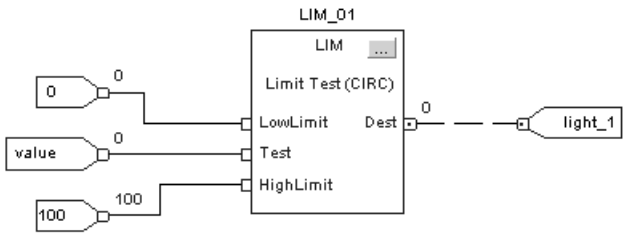
          light_1 := 1;

ELSE

          light_1 := 0;

END_IF;
```

功能块



示例 2： Low Limit ≥ High Limit:
当 *value* ≥ 0 或 *value* ≤ -100 时，设置 *light_1*。如果 *value* < 0 或 *value* >-100，则清除 *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

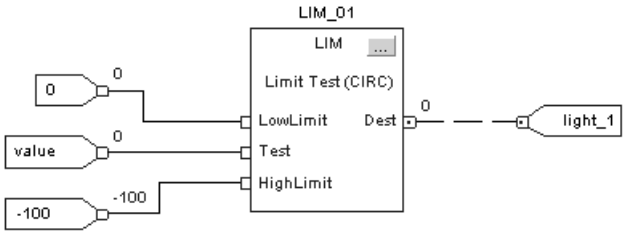
          light_1 := 1;

ELSE

          light_1 := 0;

END_IF;
```

功能块

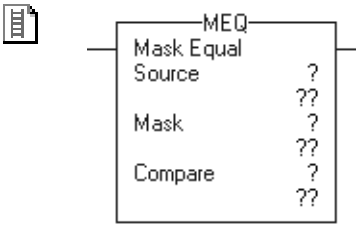


掩码等于 (Mask Equal to, MEQ)

MEQ 指令通过 Mask 传递 Source 和 Compare 值并比较结果。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT **DINT**	立即数 标记	要对照 Compare 进行测试的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Mask	SINT INT **DINT**	立即数 标记	定义要屏蔽或传递哪些位
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
比较	SINT INT **DINT**	立即数 标记	要对照 Source 进行测试的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		

结构化文本

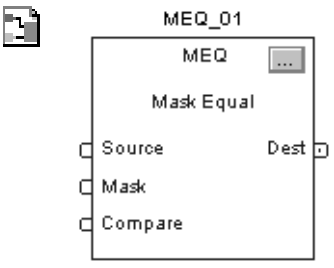
结构化文本没有 MEQ 指令，但您可以使用结构化文本实现相同的结果。

```
IF (Source AND Mask) = (Compare AND Mask) THEN

    <statement>;

END_IF;
```

功能块



操作数：	类型：	格式：	说明：
MEQ 标记	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，则不执行指令，也不更新输出。 如果设置，则该指令以 “执行”下所描述的方式执行。 默认为置位状态。
Source	DINT	要对照 Compare 进行测试的值。 任何整数都有效
Mask	DINT	定义要阻止 （屏蔽）哪些位。 任何整数都有效
比较	DINT	比较值。 任何整数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。这等效于梯形图 MEQ 指令的梯级输出条件。

说明： 掩码中的 “1”表示传递该数据位。掩码中的 “0”表示屏蔽该数据位。通常， Source、 Mask 和 Compare 值是同一种数据类型。

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的大小与最大的数据类型相同。

输入立即数掩码值

输入掩码时，编程软件默认为十进制值。如果要使用其他格式输入掩码，请在值的前面加上正确的前缀。

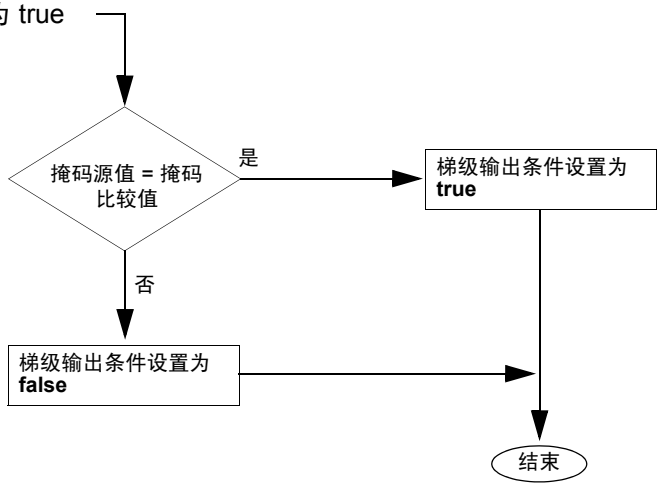
前缀：	说明：
16#	十六进制 例如， 16#0F0F
8#	八进制 例如， 8#16
2#	二进制 例如， 2#00110011

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut，指令不执行任何操作，也不更新输出。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例 1： 如果应用掩码后的 *value_1* 等于应用掩码的 *value_2*，则设置 *light_1*。如果应用掩码后的 *value_1* 不等于应用掩码的 *value_2*，则清除 *light_1*。本示例显示出掩码值相等。掩码中的 0 禁止该指令比较该位（本例中显示为 x）。

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
应用掩码的 *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
应用掩码的 *value_2*	0	1	0	1	0	1	0	1	1	1	1	1	X	X	X	X

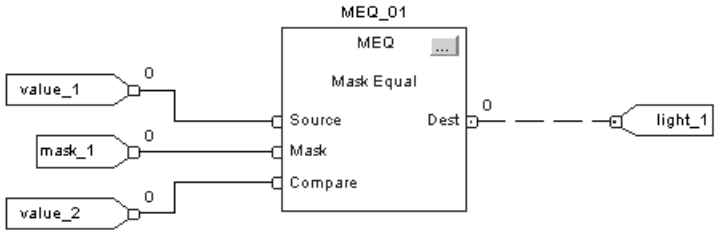
梯形图



结构化文本

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

功能块



示例 2： 如果应用掩码后的 *value_1* 等于应用掩码的 *value_2*，则设置 *light_1*。如果应用掩码后的 *value_1* 不等于应用掩码的 *value_2*，则清除 *light_1*。本示例显示出掩码值不相等。掩码中的 0 禁止该指令比较该位 （本例中显示为 x）。

value_1	0	1	0	1	0	1	0	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
应用掩码的 *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
应用掩码的 *value_2*	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0

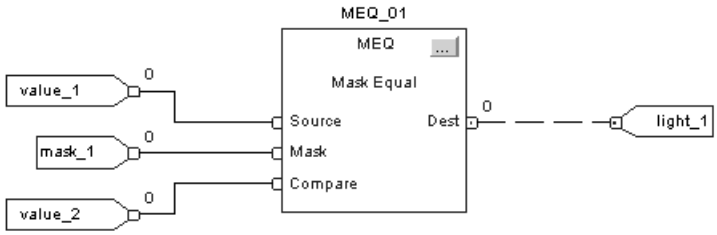
梯形图



结构化文本

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

功能块

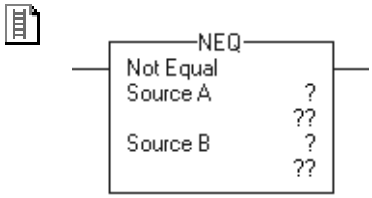


不等于 (Not Equal to, NEQ)

NEQ 指令检查 Source A 是否不等于 Source B。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source B 进行测试的值
Source B	SINT INT **DINT** **REAL** **string**	立即数 标记	针对 Source A 进行测试的值

- 如果您输入 SINT 或 INT 标记，该值将通过符号扩展转换为 DINT 值。
- 字符串数据类型为：

  衆 t 默认 STRING 数据类型

  衆 t 您创建的任何新的字符串数据类型
- 要测试字符串的字符，请为 Source A 和 Source B 都输入一个字符串标记。

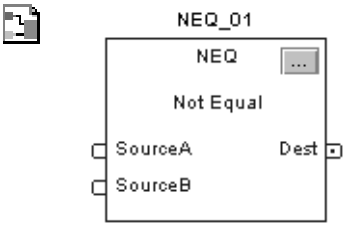
结构化文本

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

在表达式中同时使用小于号和大于号” <> “作为运算符。该表达式计算 *sourceA* 是否不等于 *sourceB*。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
NEQ 标记	FBD_COMPARE	结构	NEQ 结构

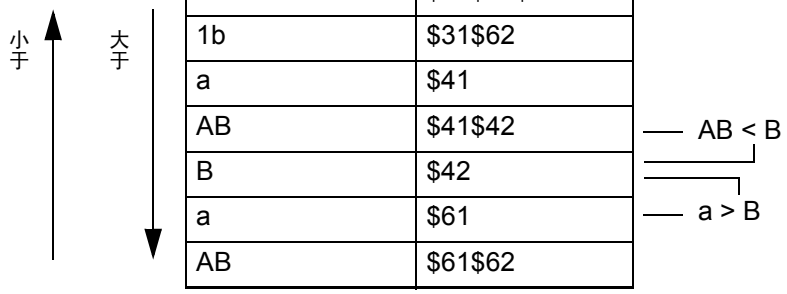
FBD_COMPARE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	针对 SourceB 进行测试的值。 任何浮点数都有效
SourceB	REAL	针对 SourceA 进行测试的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	BOOL	指令的执行结果。这等效于梯形图 NEQ 指令的梯级输出条件。

说明： NEQ 指令检查 Source A 是否不等于 Source B。

比较字符串时：

- 如果两个字符串的字符不匹配，则这两个字符串不相等。
- ASCII 字符串区分大小写。大写的 “A” (\$41) 《不》等于小写的 “a” (\$61)。



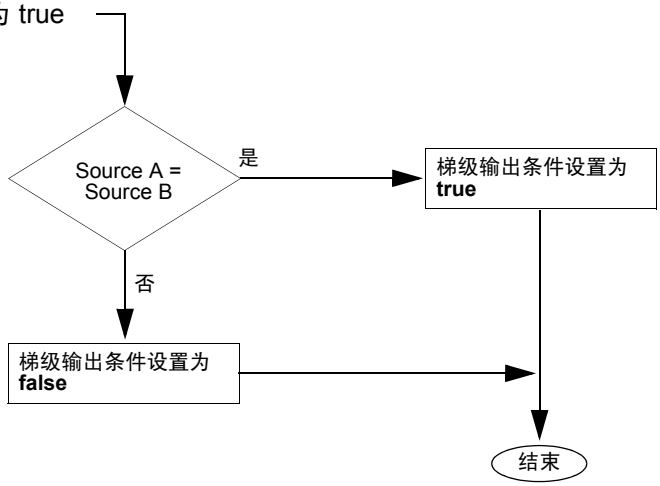
ASCII 字符	十六进制代码
1ab	\$31\$61\$62
1b	\$31\$62
a	\$41
AB	\$41\$42
B	\$42
a	\$61
AB	\$61\$62

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true 	
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 如果 *value_1* 不等于 *value_2*，则设置 *light_4*。如果 *value_1* 等于 *value_2*，则清除 *light_4*。

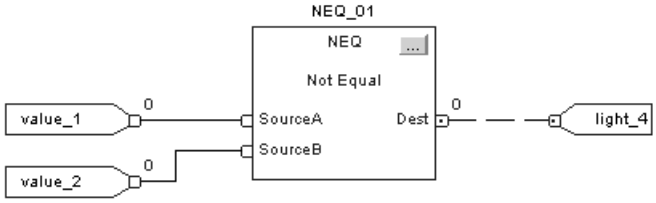
梯形图



结构化文本

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

功能块



说明：

计算 / 数学指令

（CPT、ADD、SUB、MUL、DIV、MOD、SQR、SQRT、NEG、 ABS）

简介

计算 / 数学指令使用表达式或特定的算术指令来执行算术运算。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
计算表达式	CPT	梯形图 结构化文本(1)	第 5 章 -2
将两个值相加	ADD	梯形图 结构化文本(2) 功能块	第 5 章 -6
将两个值相减	SUB	梯形图 结构化文本 (2) 功能块	第 5 章 -9
将两个值相乘	MUL	梯形图 结构化文本 (2) 功能块	第 5 章 -12
将两个值相除	DIV	梯形图 结构化文本 (2) 功能块	第 5 章 -15
确定一个值除另一个值后的余数	MOD	梯形图 结构化文本 (2) 功能块	第 5 章 -19
计算值的平方根	SQR SQRT(3)	梯形图 结构化文本 功能块	第 5 章 -23
对值求相反值	NEG	梯形图 结构化文本 (2) 功能块	第 5 章 -26
对值求绝对值	ABS	梯形图 结构化文本 功能块	第 5 章 -29

(1) 没有等效的结构化文本指令。使用其他结构化文本进行编程获得同样的结果。请参见该指令的说明。

(2) 没有等效的结构化文本指令。在表达式中使用运算符。

(3) 仅结构化文本。

您可以混用数据类型，但可能发生精度损失和舍入误差，并且指令需要更长的执行时间。检查 S:V 位，以确定结果是否被截断。

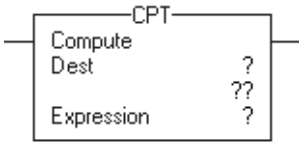
对于梯形图指令，**粗体**数据类型表示最佳数据类型。如果指令的所有操作数都使用同样的最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

计算 (Compute, CPT)

CPT 指令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：
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记
Expression（表达式）	SINT INT **DINT** **REAL**	立即数 标记	表达式由标记和 / 或由运算符分开的立即值组成
	SINT 或 INT 标记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，但您可以使用赋值语句和表达式实现相同的结果。

```
destination := numeric_expresion;
```

有关结构化文本内的赋值语句和表达式的语法的信息，请参见“附录 C”。

说明： CPT 指令执行在表达式中定义的算术运算。启用时， CPT 指令计算表达式并将结果放入 Destination 中。

相对于其他计算 / 数学指令， CPT 指令的执行速度要稍微慢一些，并且使用的内存也多。 CPT 指令的优点在于它允许您在一个指令中输入复杂的表达式。

提示 表达式的长度 《没有限制》。

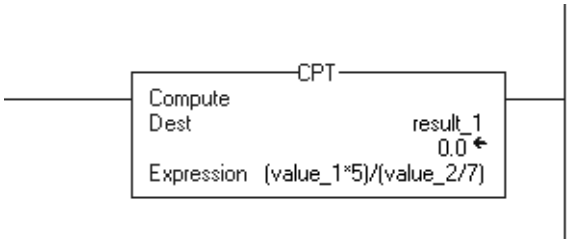
算术状态标志： 算术状态标志受影响。

错误条件： 无

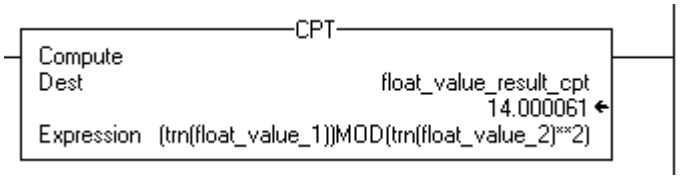
执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令计算 Expression 并将结果放入 Destination 中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

示例 1： 启用时，CPT 指令计算 *value_1* 乘以 5 的值，将该乘积再除以 *value_2* 除以 7 所得到的结果，并将最终结果放入 *result_1* 中。



示例 2： 启用时， CPT 指令截断 *float_value_1* 和 *float_value_2*，求截断后的 *float_value_2* 的 2 次幂，再用截断后的 *float_value_1* 除以该结果，并将除法运算的余数存储在 *float_value_result_cpt* 中。



有效运算符

运算符：	说明：	最优：
+	加	DINT、 REAL
-	减 / 求负	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：	最优：
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

格式化表达式

对于表达式中使用的每个运算符，必须提供一个或两个操作数 （标记或立即值）。请根据下表设置表达式中运算符和操作数的格式：

运算符涉及的操作数的个数：	使用格式：	示例：
1	运算符 (操作数)	ABS (*tag_a*)
2	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*)

确定运算次序

指令按预定的次序执行写入表达式中的运算 （不一定是写入次序）。可以使用括号将运算条目分组，强制指令在执行其他运算之前先执行括号中的运算，从而更改运算次序。

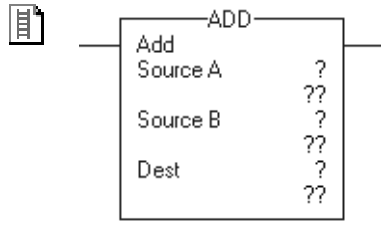
次序相等的运算从左向右执行。

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

加法 (Add, ADD)

ADD 指令将 Source A 与 Source B 相加，并将结果放在 Destination 中。

操作数：



梯形图

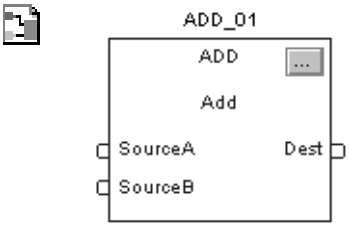
操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL**	立即数 标记	要与 Source B 相加的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Source B	SINT INT **DINT** **REAL**	立即数 标记	要与 Source A 相加的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

结构化文本

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

在表达式中使用加号 “+” 作为运算符。该表达式将 *sourceA* 和 *sourceB* 相加并将结果放入 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。



功能块

操作数：	类型：	格式：	说明：
ADD 标记	FBD_MATH	结构	ADD 结构

FBD_MATH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	要与 SourceB 相加的值。 任何浮点数都有效
SourceB	REAL	要与 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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： ADD 指令将 Source A 与 Source B 相加，并将结果放在 Destination 中。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

梯形图

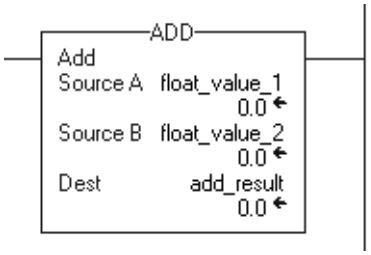
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	Destination = Source A + Source B 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 将 *float_value_1* 和 *float_value_2* 相加并将结果放入 *add_result* 中。

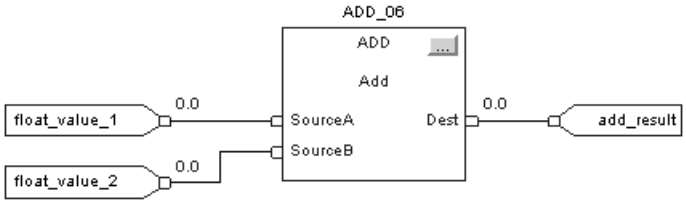
梯形图



结构化文本

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

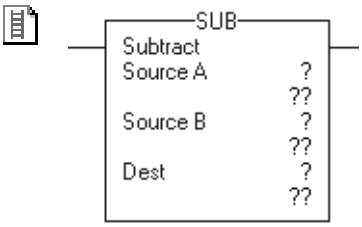
功能块



减法 (Subtract, SUB)

SUB 指令从 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：
Source A	SINT INT **DINT** **REAL**	立即数 标记	要从中减去 Source B 的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Source B	SINT INT **DINT** **REAL**	立即数 标记	要从 Source A 中减去的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

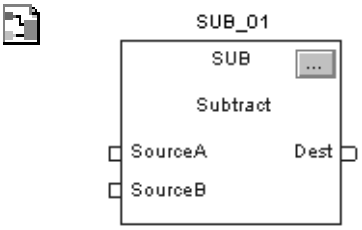
结构化文本

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

请在表达式中使用负号 “–” 作为运算符。该表达式从 *sourceA* 中减去 *sourceB* 并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
SUB 标记	FBD_MATH	结构	SUB 结构

FBD_MATH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	REAL	要从中减去 SourceB 的值。 任何浮点数都有效
SourceB	REAL	要从 SourceA 中减去的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： SUB 指令从 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

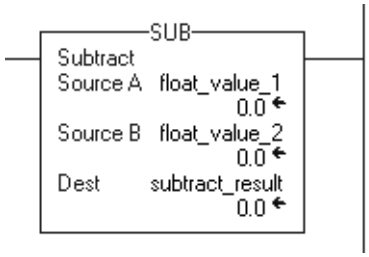
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	Destination = Source B - Source A 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 从 *float_value_1* 中减去 *float_value_2* 并将结果放入 *subtract_result* 中。

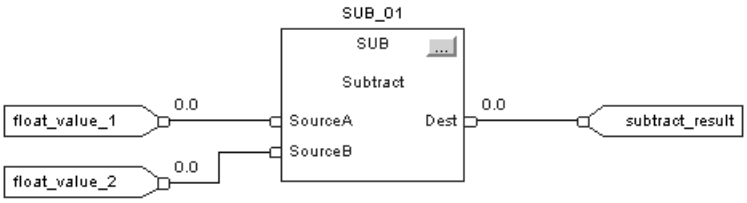
梯形图



结构化文本

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

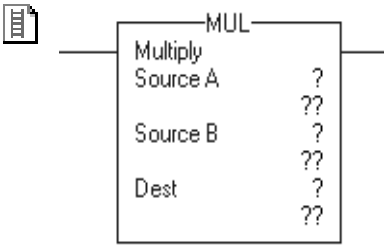
功能块



乘法 (Multiply, MUL)

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

操作数：



梯形图

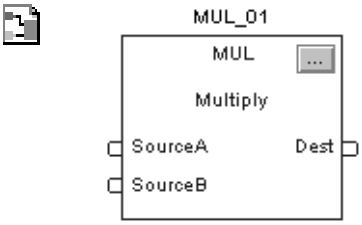
操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL**	立即数 标记	被乘数的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Source B	SINT INT **DINT** **REAL**	立即数 标记	乘数的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

结构化文本

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

在表达式中使用乘号 “*” 作为运算符。该表达式用 *sourceB* 乘以 *sourceA* 并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。



功能块

操作数：	类型：	格式：	说明：
MUL 标记	FBD_MATH	结构	MUL 结构

FBD_MATH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
Source A	REAL	被乘数的值。 任何浮点数都有效
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： MUL 指令将 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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	Destination = Source B x Source A 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

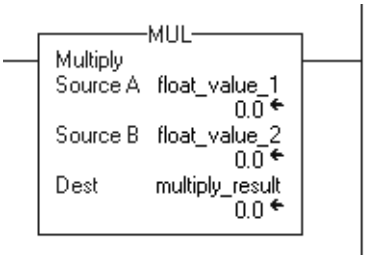


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 用 *float_value_2* 乘以 *float_value_1* 并将结果放入 *multiply_result*。

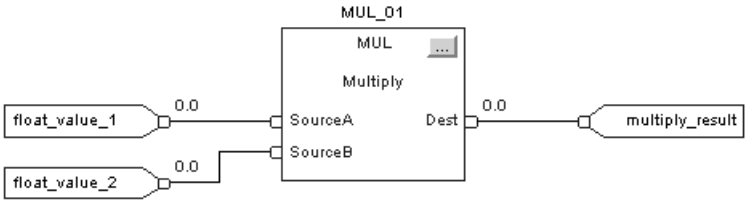
梯形图



结构化文本

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

功能块

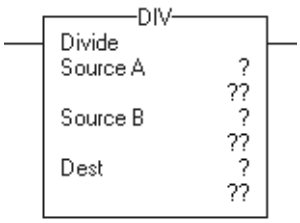


除法 (Divide, DIV)

DIV 指令用 Source B 除 Source A 并将结果放入 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL**	立即数 标记	被除数
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Source B	SINT INT **DINT** **REAL**	立即数 标记	除数
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

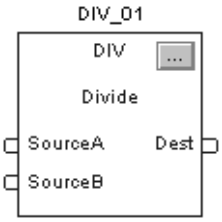
结构化文本

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

在表达式中使用除号 “/” 作为运算符。该表达式用 *sourceB* 除 *sourceA* 并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
DIV 标记	FBD_MATH	结构	DIV 结构

FBD_MATH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
Source A	REAL	被除数。 任何浮点数都有效
Source B	REAL	除数。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： 如果 Destination 《不》是 REAL，则该指令将如下所述处理计算结果的小数部分：

如果 Source A：	则结果的分数部分：	示例：		
和 Source B 都《不》是 REAL	截断	Source A	DINT	5
		Source B	DINT	3
		Destination	DINT	1
或 Source B 是 REAL	取整	Source A	REAL	5.0
		Source B	DINT	3
		Destination	DINT	2

如果 Source B （除数）为零：

- 发生轻微错误：
  - 类型 4：程序错误
  - 代码 4：算术溢出
- 按如下所述设置 destination：

如果 Source B 为零，并且：	目标是：	结果是：	则目标被设置为：
所有操作数都是整数（SINT、 INT 或 DINT）	→	→	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）

要检测是否存在除数为 0 的情况，请检查轻微错误位 (S:MINOR)。请参见 *Logix5000 Controllers Common Procedures* （Logix5000 控制器通用程序），出版号 1756-PM001。

算术状态标志： 算术状态标志受影响。

错误条件：

发生轻微错误的情况：	错误类型：	错误代码：
除数为零	4	4

执行：

梯形图

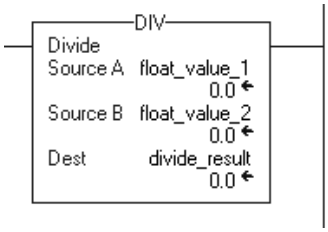
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	Destination = Source A / Source B 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例 1： 用 *float_value_2* 除 *float_value_1* 并将结果放入 *divide_result* 中。

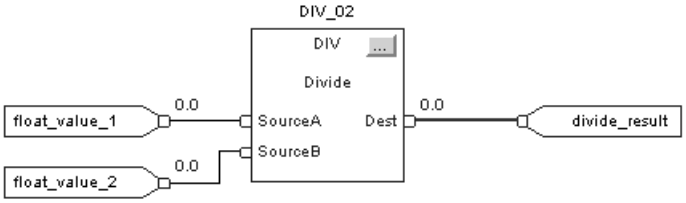
梯形图



结构化文本

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

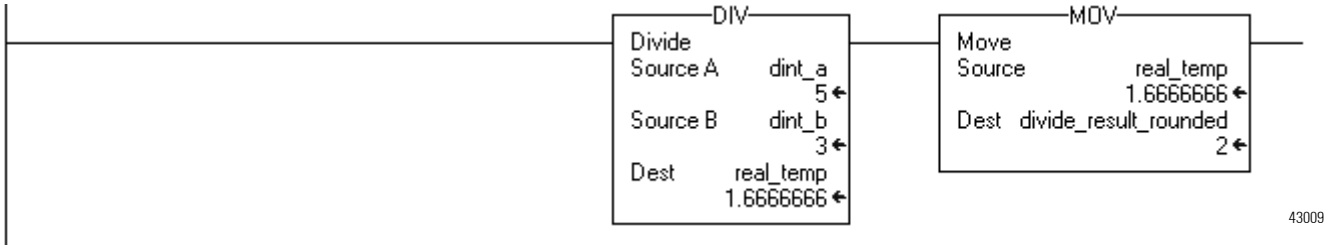
功能块



示例 2： DIV 和 MOV 指令一起使用可以计算两个整数的商，对结果取整并将结果放入一个整数标记中：

- DIV 指令用 *dint_b* 除 *dint_a*。
- 为了对结果取整， Destination 是一个 REAL 标记。（如果 destination 是一个整数标记 （SINT、 INT 或 DINT），则该指令会将结果截断。）
- MOV 指令将取整后的结果 (*real_temp*) 从 DIV 移到 *divide_result_rounded*。
- 由于 *divide_result_rounded* 是一个 DINT 标记，因此 *real_temp* 中的值取整后放入 DINT destination 中。

梯形图

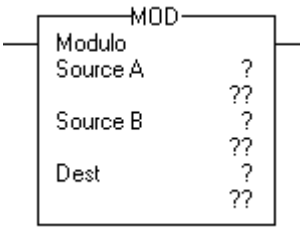


取余 (Modulo, MOD)

MOD 指令用 Source B 除 Source A 并将余数放入 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT** **REAL**	立即数 标记	被除数
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Source B	SINT INT **DINT** **REAL**	立即数 标记	除数
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

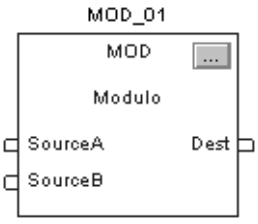
结构化文本

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

在表达式中使用 MOD 作为运算符。该表达式用 *sourceB* 除 *sourceA* 并将余数存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
MOD 标记	FBD_MATH	结构	MOD 结构

FBD_MATH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
Source A	REAL	被除数。 任何浮点数都有效
Source B	REAL	除数。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： 如果 Source B （除数）为零：

- 发生轻微错误：
  - 类型 4：程序错误
  - 代码 4：算术溢出
- 如下所述设置 destination：

如果 Source B 为零，并且：	目标是：	结果是：	则目标被设置为：
所有操作数都是整数（SINT、 INT 或 DINT）	→	→	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）

要检测是否存在除数为 0 的情况，请检查轻微错误位 (S:MINOR)。请参见 *Logix5000 Controllers Common Procedures* （Logix5000 控制器通用程序），出版号 1756-PM001。

算术状态标志： 算术状态标志受影响。

错误条件：

发生轻微错误的情况：	错误类型：	错误代码：
除数为零	4	4

执行：



梯形图

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

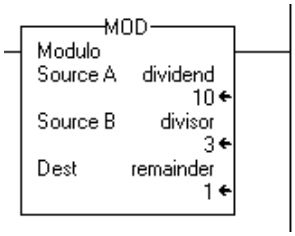


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
后扫描	不采取任何操作。

示例： 用 *dividend* 除以 *divisor* 并将余数放入 *remainder* 中。在本例中，10 是 3 的 3 倍，还余 1。

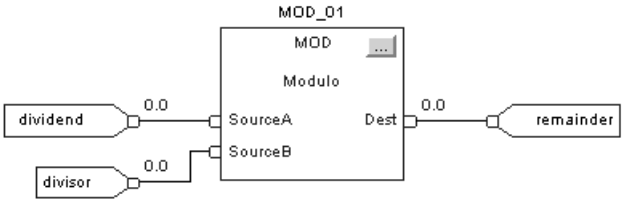
梯形图



结构化文本

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

功能块

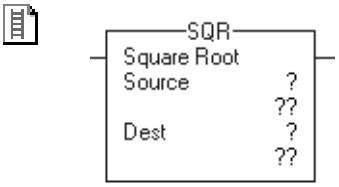


平方根 (Square Root, SQR)

SQR 指令计算 Source 的平方根并将结果放入 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT **DINT** **REAL**	立即数 标记	查找该值的平方根
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

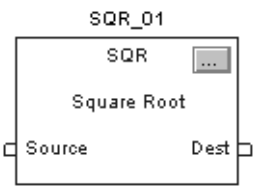
结构化文本

```
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。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
SQR 标记	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	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
Source	REAL	查找该值的平方根。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： 如果 Destination 《不是》 REAL，则该指令按如下所述处理计算结果的小数部分：

如果 Source 为：	则结果的分数部分：	示例：		
《非》 REAL	截断	Source	DINT	3
		Destination	DINT	1
REAL	取整	Source	REAL	3.0
		Destination	DINT	2

如果 Source 为负数，则该指令会在计算平方根之前先计算 Source 的绝对值。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	$Destination = \sqrt{Source}$ 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

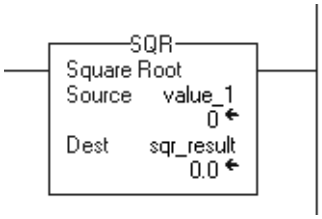


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 计算 *value_1* 的平方根并将结果放入 *sqr_result* 中。

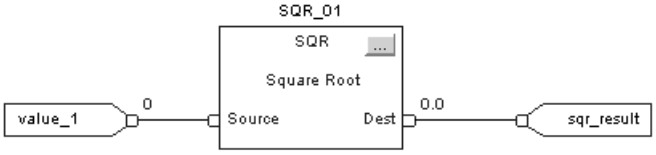
梯形图



结构化文本

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

功能块

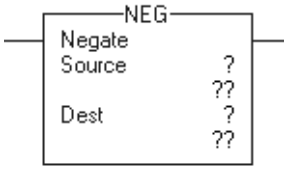


求反 (Negate, NEG)

NEG 指令改变 Source 的符号并将结果放入 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT **DINT** **REAL**	立即数 标记	要求反的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

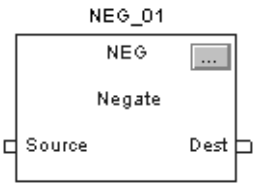
结构化文本

```
dest := source;
```

请在表达式中使用负号 “–”作为运算符。该表达式改变 *source* 的符号并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。

功能块



操作数：	类型：	格式：	说明：
NEG 标记	FBD_MATH_ADVANCED	结构	NEG 结构

FBD_MATH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 设置默认值
Source	REAL	要求反的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： 如果对负值求反，结果为正数。如果对正值求反，结果为负数。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	Destination = 0 – Source 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

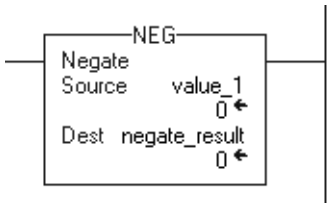


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 改变 *value_1* 的符号并将结果放入 *negate_result* 中。

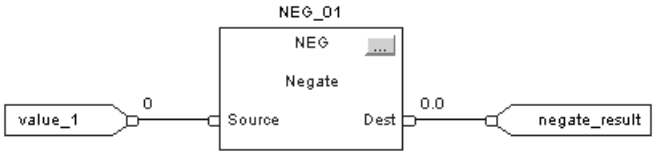
梯形图



结构化文本

```
negate_result := value_1;
```

功能块

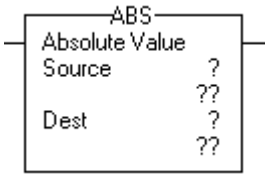


绝对值 (Absolute Value, ABS)

ABS 指令获取 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：
Source	SINT INT **DINT** **REAL**	立即数 标记	要计算绝对值的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记



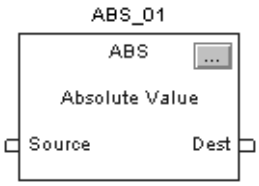
结构化文本

```
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。

有关结构化文本中的表达式语法的信息，请参见 “附录 C”。





功能块

操作数：	类型：	格式：	说明：
ABS 标记	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	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
Source	REAL	要计算绝对值的值。 任何浮点数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： ABS 指令获取 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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	Destination = | Source | 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

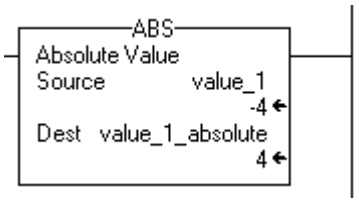


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 将 *value_1* 的绝对值放入 *value_1_absolute* 中。在本例中， -4 的绝对值为 4。

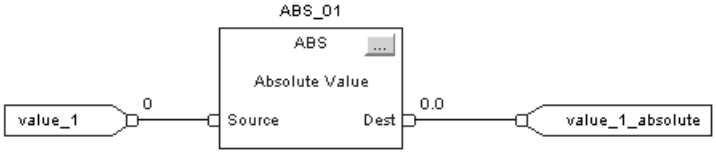
梯形图



结构化文本

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

功能块



说明：

第 6 章

移动 / 逻辑指令

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

简介

您可以混用数据类型，但可能发生精度损失和舍入误差，并且指令需要更长的执行时间。检查 S:V 位，以确定结果是否被截断。

对于梯形图指令，**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样的最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

移动指令用于修改和移动位。

操作目的：	操作指令：	适用语言：	参考页面：
复制值	MOV	梯形图 结构化文本(1)	第 6 章 -3
复制整数的特定部分	MVM	梯形图	第 6 章 -5
复制功能块中的整数的特定部分	MVMT	结构化文本 功能块	第 6 章 -8
在整数内或整数之间移动位	BTD	梯形图	第 6 章 -11
在功能块中的整数内或整数之间移动位	BTDT	结构化文本 功能块	第 6 章 -14
值清零	CLR	结构化文本 (1) 梯形图	第 6 章 -17
重新排列 INT、 DINT 或 REAL 标记的字节	SWPB	梯形图 结构化文本	第 6 章 -19

(1) 没有等效的结构化文本指令。使用其他结构化文本进行编程获得同样的结果。请参见该指令的说明。

逻辑指令执行位运算。

操作目的：	操作指令：	适用语言：	参考页面：
按位与运算	按位与 &(1)	梯形图 结构化文本(2) 功能块	第 6 章 -23
按位或运算	按位或	梯形图 结构化文本 (2) 功能块	第 6 章 -26
按位异或运算	按位异或	梯形图 结构化文本 (2) 功能块	第 6 章 -29
按位非运算	按位非	梯形图 结构化文本 (2) 功能块	第 6 章 -32
最多可以对 8 个布尔输入执行逻辑与运算。	布尔与 (BAND)	结构化文本 (2) 功能块	第 6 章 -35
最多可以对 8 个布尔输入执行逻辑或运算。	布尔或 (BOR)	结构化文本 (2) 功能块	第 6 章 -38
对 2 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	第 6 章 -41
对布尔输入求反。	布尔非 (BNOT)	结构化文本 (2) 功能块	第 6 章 -44

(1) 仅结构化文本。

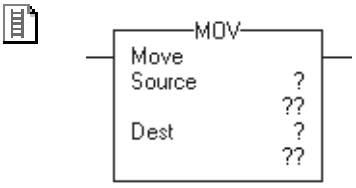
(2) 在结构化文本中， AND、 OR、 XOR 和 NOT 运算既可以按位执行，也可以按逻辑执行。

移动 (Move, MOV)

MOV 指令将 Source(源) 复制到 Destination （目标）。 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 标记	要移动 （复制）的值
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

结构化文本

```
dest := source;
```

将赋值运算符 “:=”与表达式结合使用。此赋值运算符将 *source* 中的值移到 *dest*。

有关结构化文本内的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： MOV 指令将 Source 复制到 Destination。 Source 保持不变。

算术状态标志： 算术状态标志受影响。

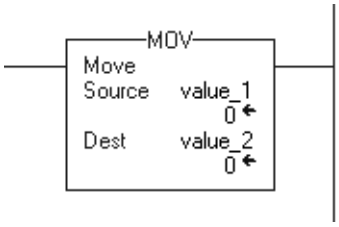
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令将 Source 复制到 Destination。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

示例： 将 *value_1* 中的数据移到 *value_2*。

梯形图



结构化文本

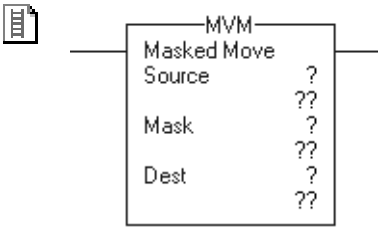
```
value_2 := value_1;
```

掩码移动 (Masked Move, MVM)

MVM 指令将 Source 复制到 Destination，并允许部分数据显示为掩码。

该指令在结构化文本和功能块中以 MVMT 的形式提供，请参见第 第 6 章 -8 页。

操作数：



梯形图

操作数：	类型：	格式：	说明：
Source	SINT INT **DINT**	立即数 标记	要移动的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Mask	SINT INT **DINT**	立即数 标记	要阻止或传递的位
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Destination	SINT INT **DINT**	标记	存储结果的标记

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 的形式提供。或者，您也可以在表达式中组合按位逻辑，并将结果赋给 Destination。该表达式对 *Source* 执行掩码移动指令。

有关结构化文本内的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： MVM 指令使用 Mask 来传递或屏蔽 Source 数据位。掩码中的“1”表示传递该数据位。掩码中的 “0”表示该数据位被屏蔽。

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的字节大小与最大的数据类型相同。

输入立即掩码值

输入掩码时，编程软件默认为十进制值。如果要使用其他格式输入掩码，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

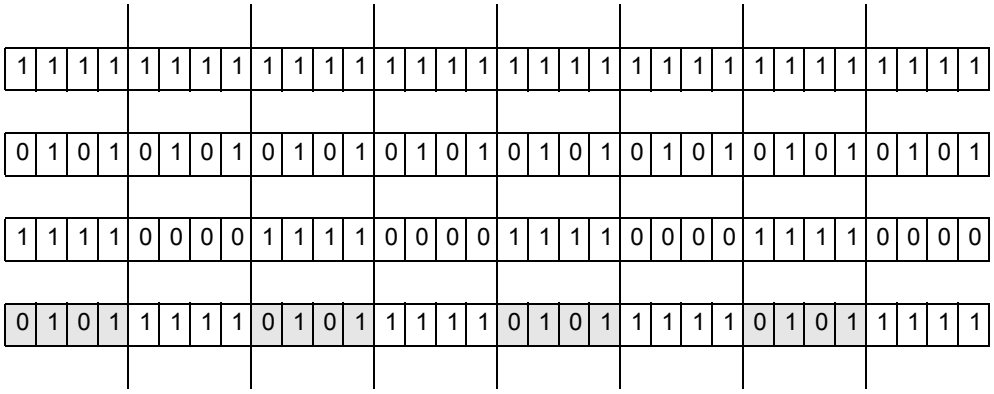
算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

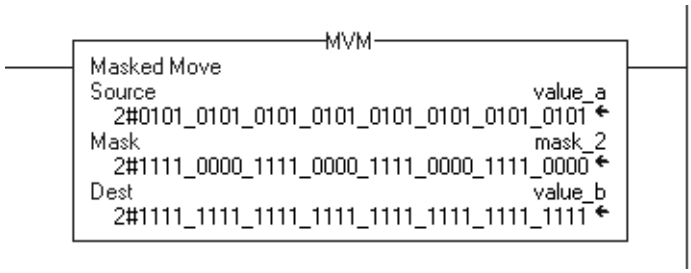
条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令通过 Mask 传递 Source，并将结果复制到 Destination 中。 Destination 中的非掩码位保持不变。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

示例： 将 *value_a* 中的数据复制到 *value_b*，同时允许数据显示为掩码（0 屏蔽 *value_a* 中的数据）。



带阴影的框显示的是 *value_b* 中已更改的位。

梯形图



结构化文本

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

带目标的掩码移动 (Masked Move with Target, MVMT)

MVMT 指令首先将 Target 复制到 Destination。然后，该指令将屏蔽后的 Source 与 Destination 进行比较，并对 Destination 进行任何必要的更改。 Target 和 Source 保持不变。

该指令在梯形图中以 MVM 的形式提供，请参见第 第 6 章 -5 页。

操作数：

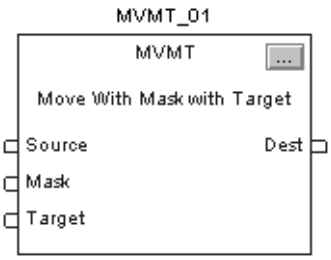


```
MVMT(MVMT_tag);
```

结构化文本

变量：	类型：	格式：	说明：
MVMT 标记	FBD_MASKED_MOVE	结构	MVMT 结构





功能模块图

操作数：	类型：	格式：	说明：
MVMT 标记	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，则不执行指令，也不更新输出。 如果为置位状态，则执行指令。 默认为置位状态。 **结构化文本：** 无影响。指令执行。
Source	DINT	要基于 Mask 的值移到 Destination 的输入值。 任何整数都有效
Mask	DINT	要从 Source 移到 Dest 的位的掩码。设置为 1 的所有位会导致对应的位从 Source 移到 Dest。设置为 0 的所有位会导致对应的位不从 Source 移到 Dest。 任何整数都有效
目标	DINT	要在通过 Mask 移动 Source 位之前移到 Dest 的输入值。 任何整数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	DINT	掩码移动指令的结果。此输出设置了算术状态标志。

说明： 启用时， MVMT 指令使用 Mask 来传递或阻止 Source 数据位。掩码中的 “1” 表示传递该数据位。掩码中的 “0” 表示该数据位被阻止。

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的字节大小与最大的数据类型相同。

使用输入引用输入立即掩码值

输入掩码时，编程软件默认为十进制值。如果要使用其他格式输入掩码，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

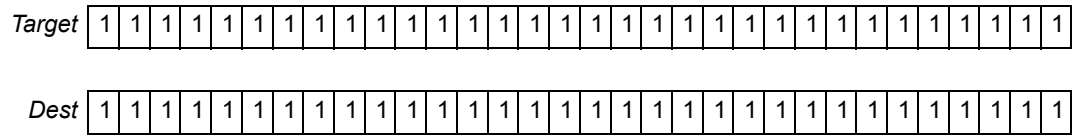
算术状态标志： 算术状态标志受影响。

错误条件： 无

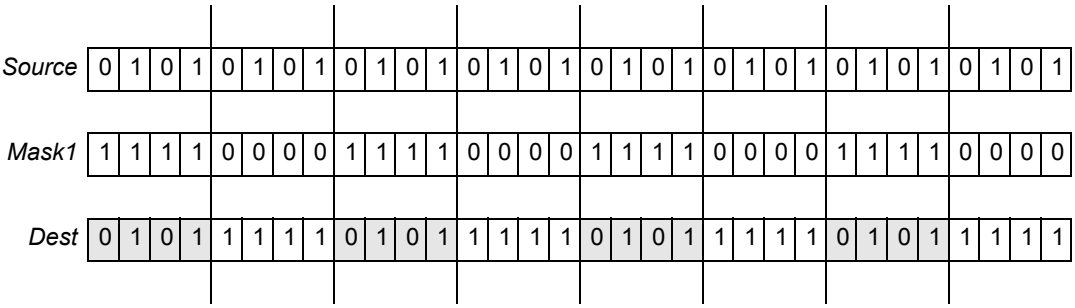
执行：

条件：	功能块操作：	结构化文本操作：
预扫描	不采取任何操作。	不采取任何操作。
指令第一次扫描	不采取任何操作。	不采取任何操作。
指令第一次运行	不采取任何操作。	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零，指令不执行任何操作，也不更新输出。	不可用
EnableIn 处于置位状态	指令执行。 设置 EnableOut。	EnableIn 始终为置位状态。 指令执行。
后期扫描	不采取任何操作。	不采取任何操作。

示例： **1.** 将 Target 复制到 Dest。



2. 屏蔽 Source 并将它与 Dest 进行比较。在 Dest 中进行任何必要的更改。 Source 和 Target 保持不变。掩码中的 0 禁止该指令比较该位 （在本例中显示为 x）。



带阴影的框显示已更改的位。

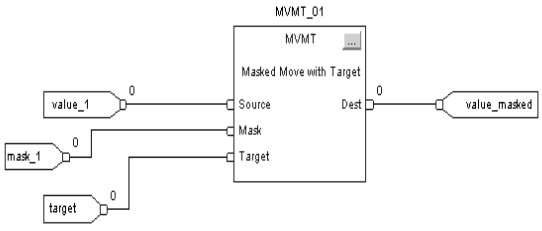
结构化文本

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

MVMT(MVMT_01);

value_masked := MVMT_01.Dest;
```

功能块



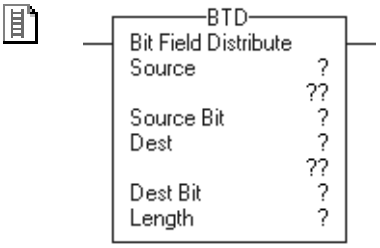
位域分配 (Bit Field Distribute, BTD)

BTD 指令复制 Source 中的指定位，将这些位移到适当的位置，并将这些位写入 Destination。

该指令在结构化文本和功能块中以 BTDT 的形式提供，请参见第 第 6 章 -14 页。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT **DINT**	立即数 标记	包含要移动的位的标记
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Source 位	DINT	立即数 (0-31 DINT) (0-15 INT) (0-7 SINT)	位编号 （最低位编号），将从该位置开始移动 必须在 Source 数据类型的有效范围内
Destination	SINT INT **DINT**	标记	位要移动到的位置的标记
目标位	DINT	立即数 (0-31 DINT) (0-15 INT) (0-7 SINT)	位编号 （最低位编号），将从该位置开始复制 Source 中的位 必须在 Destination 数据类型的有效范围内
Length	DINT	立即数 (1-32)	要移动的位数

说明： 启用时， BTD 指令将一组位从 Source 复制到 Destination。这组位由 Source 位 （组中最低的位编号）和 Length （要复制的位数）确定。Destination 位标识在 Destination 中作为起始位的最低位编号。Source 保持不变。

如果该位字段的长度超出 Destination 的长度，该指令不会保存多余的位。任何多余的位都不会放入下一个字中。

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的字节大小与最大的数据类型相同。

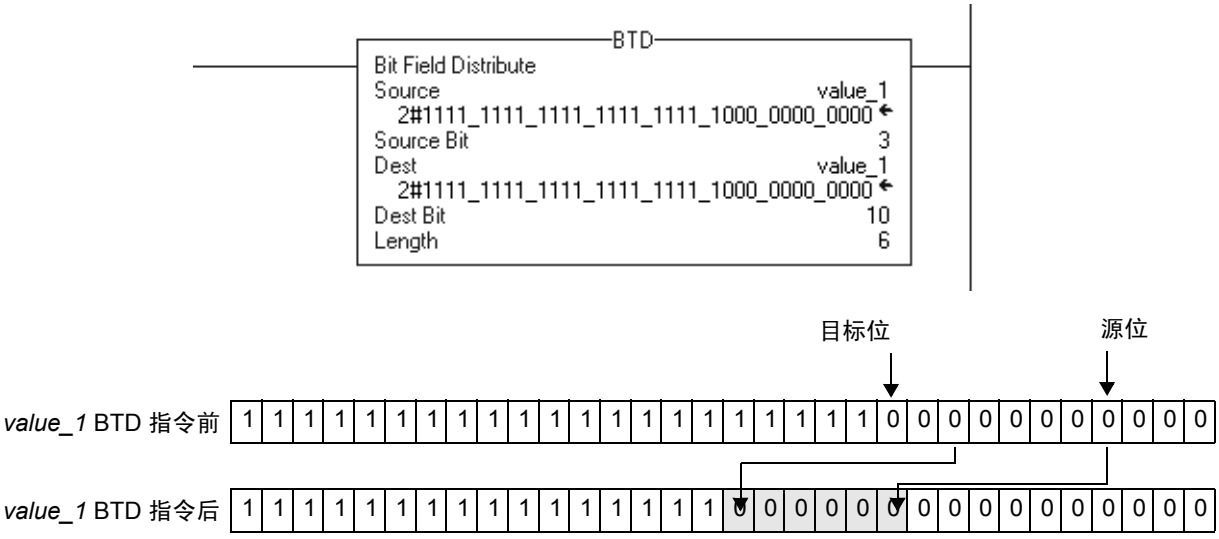
算术状态标志： 不受影响

错误条件： 无

执行：

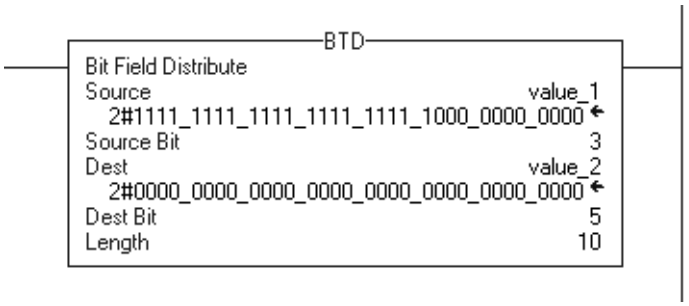
条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令复制 Source 位并将这些位移到 Destination。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

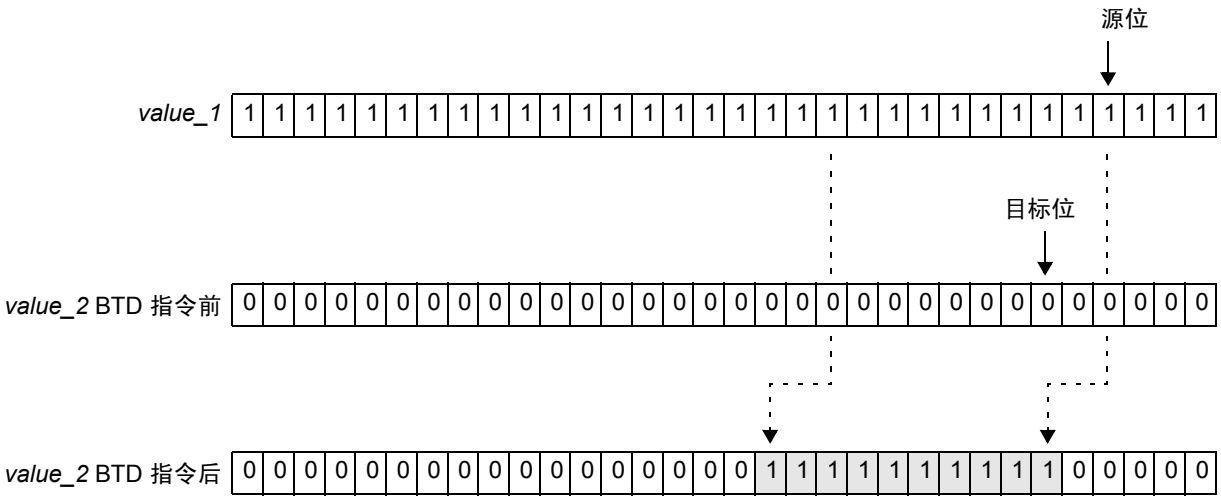
示例 1： 启用时， BTD 指令在 *value_1* 内部移动位。



带阴影的框显示 *value_1* 中已更改的位。

示例 2： 启用时， BTD 指令将 *value_1* 中的 10 个位移到 *value_2*。





带阴影的框显示 *value_2* 中已更改的位。

带目标的位域分配 (Bit Field Distribute with Target, BTDT)

BTDT 指令首先将 Target 复制到 Destination。然后，该指令复制 Source 中的指定位，将这些位移到适当的位置，并将这些位写入 Destination。 Target 和 Source 保持不变。

该指令在梯形图中以 BTD 的形式提供，请参见第 第 6 章 -11 页。

操作数：

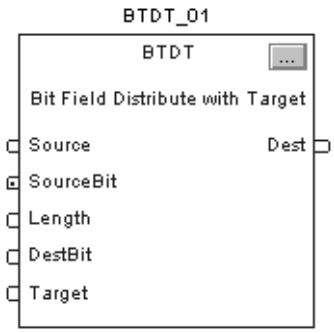


```
BTDT(BTDT_tag);
```

结构化文本

变量：	类型：	格式：	说明：
BTDT 标记	FBD_BIT_FIELD_DISTRIBUTE	结构	BTDT 结构





功能块

操作数：	类型：	格式：	说明：
BTDT 标记	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，则不执行指令，也不更新输出。 如果为置位状态，则执行指令。 默认为置位状态。 **结构化文本：** 无影响。指令执行。
Source	DINT	包含要移动到 Destination 的位的输入值。 任何整数都有效
SourceBit	DINT	Source 中的位的位置 （最低位编号，将从该位置开始移动）。 有效值 = 0-31
Length	DINT	要移动的位数 有效值 = 1-32
DestBit	DINT	Dest 中的位的位置 （最低位编号，位将复制到从该位置开始的位置）。 有效值 = 0-31
目标	DINT	在从 Source 移动位之前，要移到 Dest 的输入值。 任何整数都有效
EnableOut	BOOL	指令产生有效结果。
Dest	DINT	位移动操作的结果。此输出设置了算术状态标志。

说明： 启用时， BTD 指令将一组位从 Source 复制到 Destination。这组位由 Source 位 （组中最低的位编号）和 Length （要复制的位数）确定。Destination 位标识在 Destination 中作为起始位的最低位编号。Source 保持不变。

如果该位字段的长度超出 Destination 的长度，该指令不会保存多余的位。任何多余的位都不会放入下一个字中。

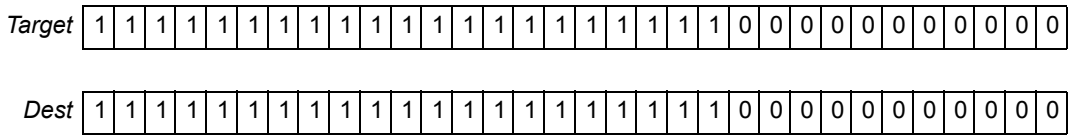
算术状态标志： 算术状态标志受影响

错误条件： 无

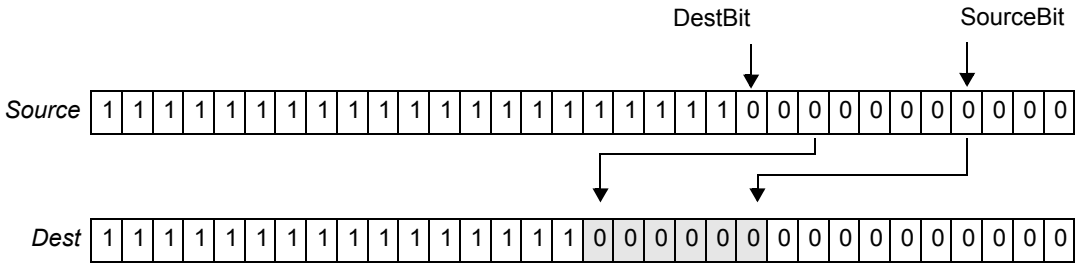
执行：

条件：	功能块操作：	结构化文本操作：
预扫描	不采取任何操作。	不采取任何操作。
指令第一次扫描	不采取任何操作。	不采取任何操作。
指令第一次运行	不采取任何操作。	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零，指令不执行任何操作，也不更新输出。	不可用
EnableIn 处于置位状态	指令执行。 设置 EnableOut。	EnableIn 始终为置位状态。 指令执行。
后期扫描	不采取任何操作。	不采取任何操作。

示例： **1.** 控制器将 Target 复制到 Dest。



2. SourceBit 和 Length 指定 Source 中要复制到 Dest 的那些位，要复制到的位置从 DestBit 开始。 Source 和 Target 保持不变。



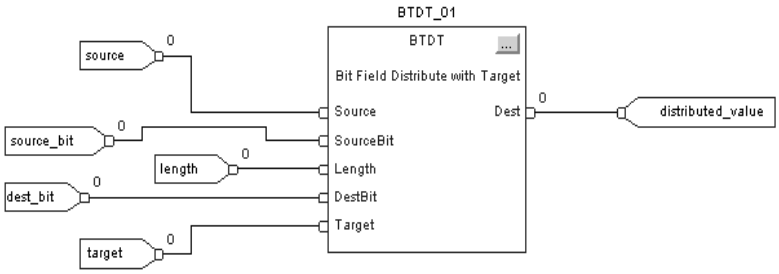
结构化文本

```
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

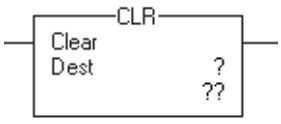


清零 (Clear, CLR)

CLR 指令将 Destination 的所有位清零。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Destination	SINT INT **DINT** **REAL**	标记	要清零的标记

结构化文本

```
dest := 0;
```

结构化文本没有 CLR 指令。应将 0 赋给要清零的标记。此赋值语句将 *dest* 清零。

有关结构化文本内的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 指令将 Destination 的所有位清零。

算术状态标志： 算术状态标志受影响。

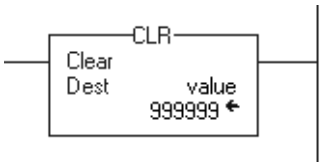
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令将 Destination 清零。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

示例： 将 *value* 的所有位清零为 0。

梯形图



结构化文本

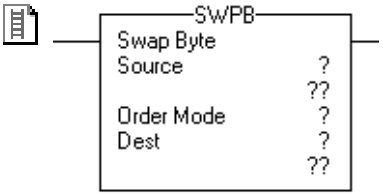
```
value := 0;
```

交换字节 (Swap Byte, SWPB)

SWPB 指令重新排列值的字节。

操作数：

梯形图



操作数：	类型：	格式：	输入：		
Source	INT DINT REAL	标记	包含要重新排列的字节的标记		
排序模式			**如果 Source 是：**	您要将字节更改为该模式（每个字母表示一个不同的字节）：	**则选择：**
			INT	n/a	任意选项
			DINT REAL	ABCD ⇒ DCBA	REVERSE（或输入 0）
				ABCD ⇒ CDAB	WORD（或输入 1）
				ABCD ⇒ BADC	HIGH/LOW（或输入 2）
Destination	INT DINT REAL	标记	要按新顺序存储字节的标记		
			如果 Source 是：	**则 Destination 必须是：**	
			INT	INT	
				DINT	
			DINT	DINT	
			REAL	REAL	

结构化文本

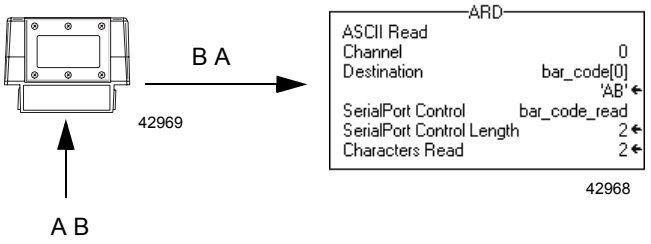
```
SWPB(Source,OrderMode,Dest);
```

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，请将它以 HIGHLOW 或 HIGH_LOW（不带斜杠）的形式输入。

说明： SWPB 指令重新排列 Source 的字节的顺序。它将结果放入 Destination 中。

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

条形码读取器



标记名称	值	样式	类型
bar_code[0]	AB	ASCII	INT

算术状态标志： 不受影响

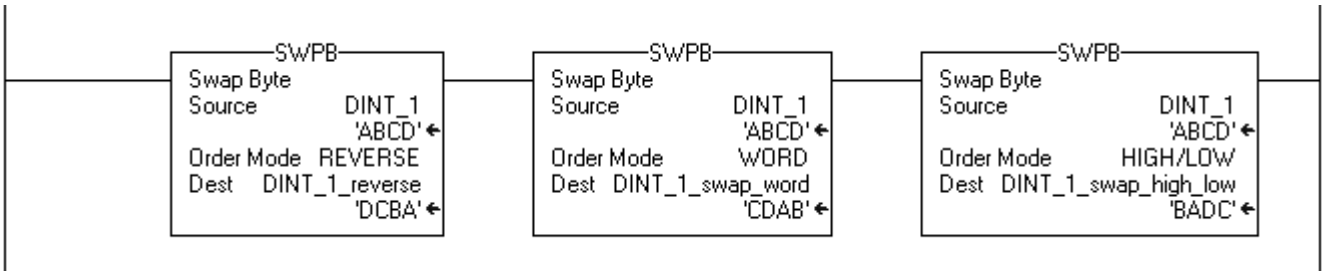
错误条件： 无

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 false。	无操作
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令重新排列指定的字节。	该指令重新排列指定的字节。
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例 1： 三个 SWPB 指令按照各不相同的排序模式重新排列 *DINT_1* 的字节。显示样式为 ASCII，每个字符均表示一个字节。每个指令都按照新的顺序将字节放入一个不同的 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： 下面的示例反转数组的每个元素中的字节。对于包含该示例的 RSLogix 5000 项目，打开 RSLogix 5000\Projects\Samples 文件夹中的 Swap_Bytes_in_Array.ACD 文件。

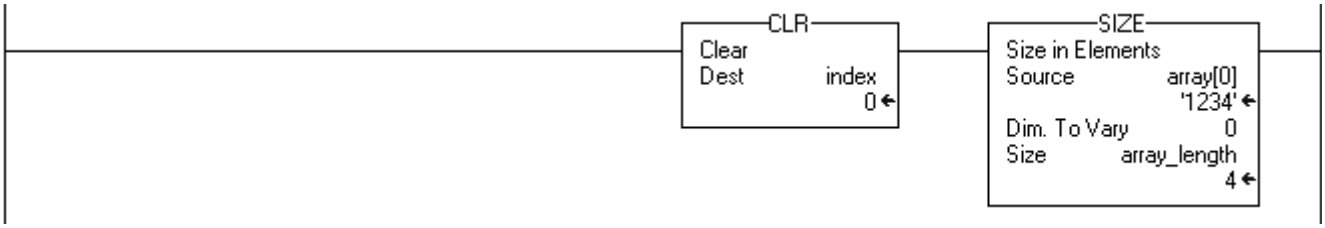
1. 初始化标记。 SIZE 指令查找 *array* 中的元素数并将该值存储在 *array_length* 中。后续指令使用该值来确定例程何时已作用于数组中的所有元素。

2. 反转 *array* 的一个元素中的字节。
   - SWPB 指令反转 *index* 的值所表示的元素编号的字节。例如，当 *index* 等于 0 时， SWPB 指令作用于 *array[0]*。
   - ADD 指令对 *index* 进行递增。当下一次执行该指令时， SWPB 指令作用于 *array* 中的下一个元素。

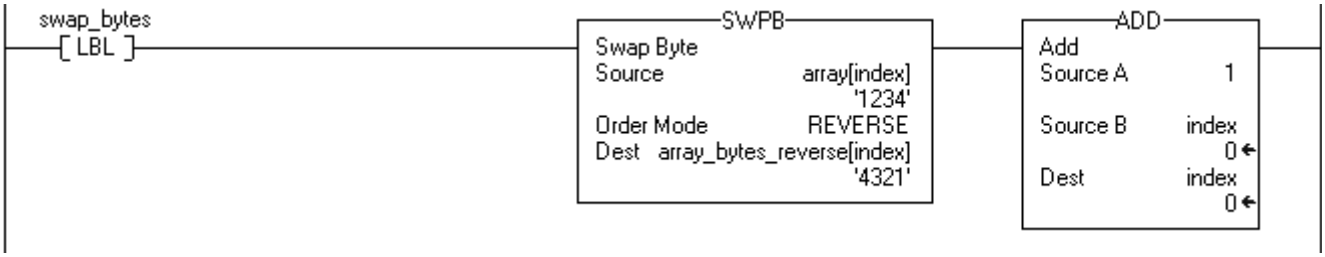
3. 确定 SWPB 指令何时已作用于数组中的所有元素。
   - 如果 *index* 小于数组中的元素数 (*array_length*)，那么继续作用于数组中的下一个元素。
   - 如果 *index* 等于 *array_length*，那么 SWPB 已作用于数组中的所有元素。

梯形图

初始化标记。



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;
```

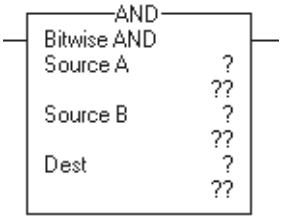
按位与 (Bitwise AND, AND)

AND 指令使用 Source A 和 Source B 中的位执行按位与运算，并将结果放入 Destination 中。

要执行逻辑与运算，请参见第 第 6 章 -35 页。

操作数：





梯形图

操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT**	立即数 标记	要与 Source B 进行与运算的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Source B	SINT INT **DINT**	立即数 标记	要与 Source A 进行与运算的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Destination	SINT INT **DINT**	标记	存储结果



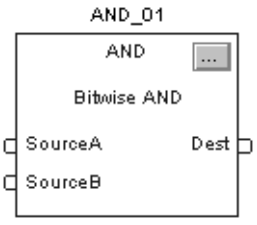
```
dest := sourceA AND sourceB
```

结构化文本

在表达式中使用 AND 或符号 “&”作为运算符。该表达式计算 *sourceA* AND *sourceB*。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。





功能块

操作数：	类型：	格式：	说明：
AND 标记	FBD_LOGICAL	结构	AND 结构

FBD_LOGICAL 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	DINT	要与 SourceB 进行与运算的值。 任何整数都有效
SourceB	DINT	要与 SourceA 进行与运算的值。 任何整数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	DINT	指令的执行结果。此输出设置了算术状态标志。

说明： 启用时，该指令执行与运算：

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

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的字节大小与最大的数据类型相同。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：



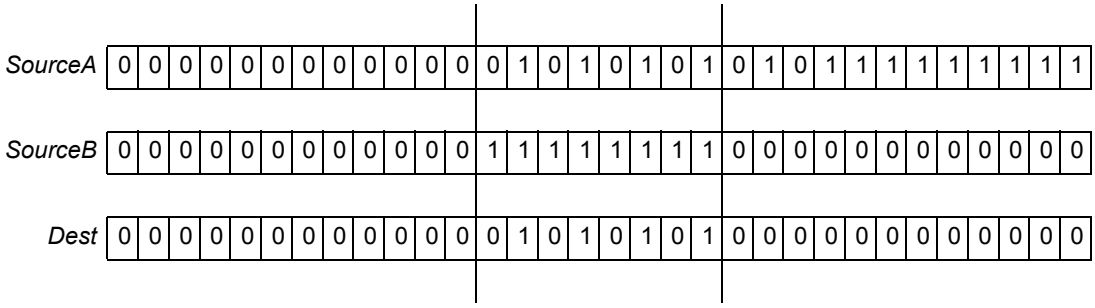
梯形图

条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令执行按位与运算。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

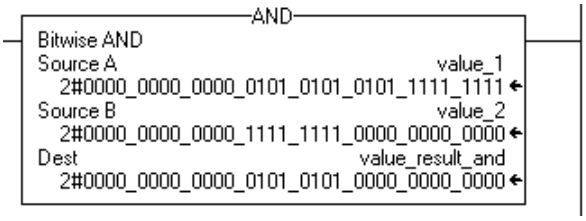
功能模块图

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 启用时， AND 指令对 SourceA 和 SourceB 执行按位与运算，并将结果放入 Dest 中。



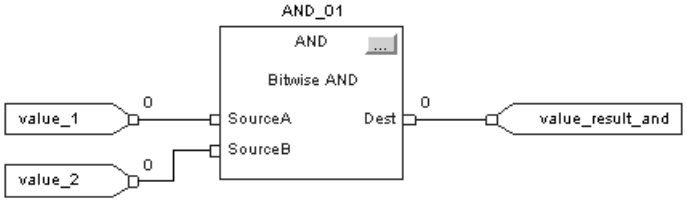
梯形图



结构化文本

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

功能块

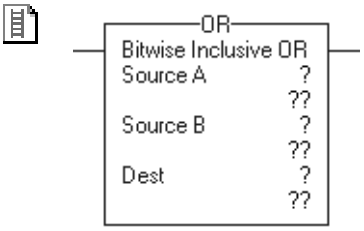


按位或 (Bitwise OR, OR)

OR 指令使用 Source A 和 Source B 中的位执行按位或运算，并将结果放入 Destination 中。

要执行逻辑或运算，请参见第 第 6 章 -38 页。

操作数：



梯形图

操作数：	类型：	格式：	说明：
Source A	SINT INT **DINT**	立即数 标记	要与 Source B 进行或运算的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Source B	SINT INT **DINT**	立即数 标记	要与 Source A 进行或运算的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Destination	SINT INT **DINT**	标记	存储结果

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

结构化文本

在表达式中使用 OR 作为运算符。该表达式计算 *sourceA* OR *sourceB*。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。



功能块

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

FBD_LOGICAL 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	DINT	要与 SourceB 进行或运算的值。 任何整数都有效
SourceB	DINT	要与 SourceA 进行或运算的值。 任何整数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	DINT	指令的执行结果。此输出设置了算术状态标志。

说明： 启用时，该指令执行或运算：

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

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的字节大小与最大的数据类型相同。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：



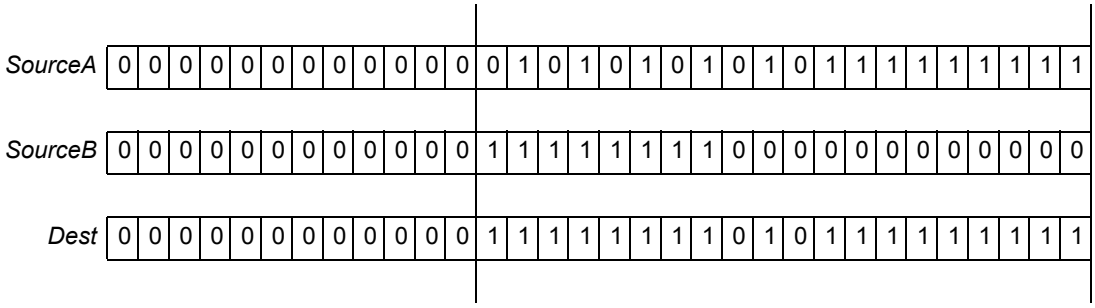
梯形图

条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令执行按位或运算。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

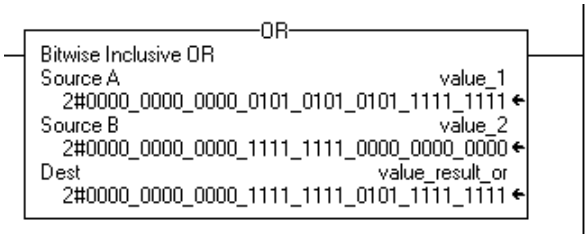
功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 启用时， OR 指令对 SourceA 和 SourceB 执行按位或运算，并将结果放入 Dest 中。



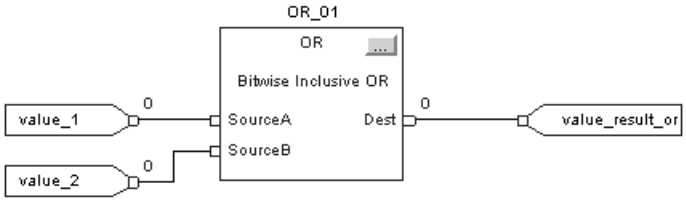
梯形图



结构化文本

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

功能块

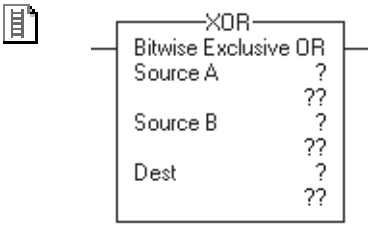


按位异或 (Bitwise Exclusive OR, XOR)

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

要执行逻辑异或运算，请参见第 第 6 章 -41 页。

操作数：



梯形图

操作数：	类型：	格式：	说明：
Source A	SINT INT DINT	立即数 标记	要与 Source B 进行异或运算的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Source B	SINT INT DINT	立即数 标记	要与 Source A 进行异或运算的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Destination	SINT INT DINT	标记	存储结果

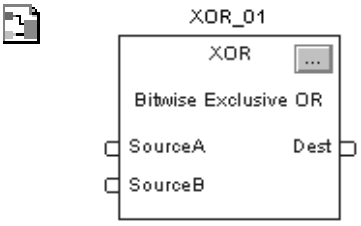


```
dest := sourceA XOR sourceB
```

结构化文本

在表达式中使用 XOR 作为运算符。该表达式计算 *sourceA* XOR *sourceB*。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。



功能块

操作数：	类型：	格式：	说明：
XOR 标记	FBD_LOGICAL	结构	XOR 结构

FBD_LOGICAL 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
SourceA	DINT	要与 SourceB 进行异或运算的值。 任何整数都有效
SourceB	DINT	要与 SourceA 进行异或运算的值。 任何整数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	DINT	指令的执行结果。此输出设置了算术状态标志。

说明： 启用时，该指令执行异或运算：

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

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的字节大小与最大的数据类型相同。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：



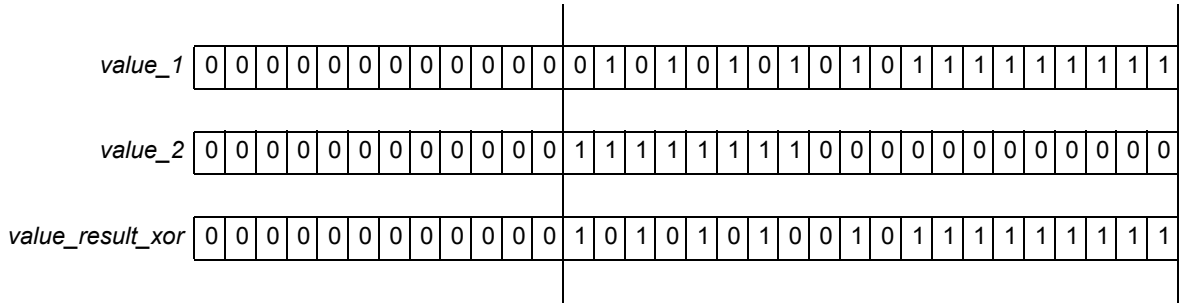
梯形图

条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令执行按位或运算。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

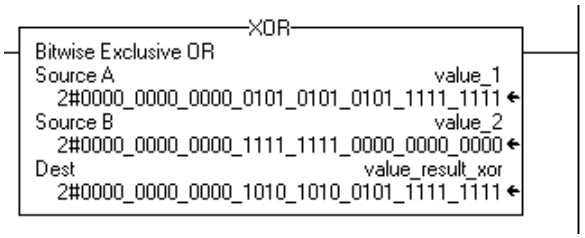
功能模块图

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 启用时， XOR 指令对 SourceA 和 SourceB 执行按位异或运算，并将结果放入 destination 标记中。



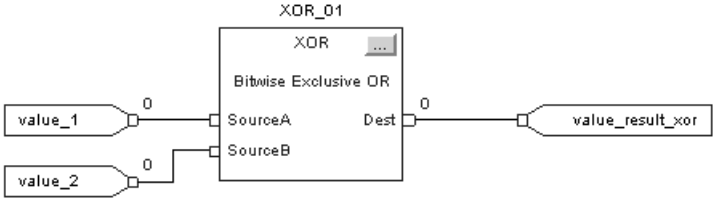
梯形图



结构化文本

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

功能块



按位非 (Bitwise NOT, NOT)

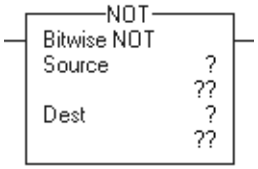
NOT 指令使用 Source 中的位执行按位非运算并将结果放入 Destination 中。

要执行逻辑非运算，请参见第 第 6 章 -44 页。

操作数：

梯形图





操作数：	类型：	格式：	说明：
Source	SINT INT DINT	立即数 标记	要执行非运算的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Destination	SINT INT DINT	标记	存储结果

结构化文本



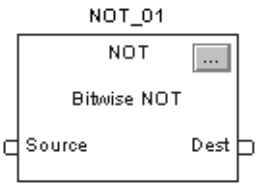
```
dest := NOT source
```

在表达式中使用 NOT 作为运算符。该表达式计算 NOT *source*。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。

功能块





操作数：	类型：	格式：	说明：
NOT 标记	FBD_LOGICAL	结构	NOT 结构

FBD_LOGICAL 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	启用输入。如果为清零状态，则不执行指令，也不更新输出。 设置默认值
Source	DINT	要执行非运算的值。 任何整数都有效
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	指令产生有效结果。
Dest	DINT	指令的执行结果。此输出设置了算术状态标志。

说明： 启用时，该指令执行非运算：

如果 Source 中的位为：	则 Destination 中的位是：
0	1
1	0

如果您混用整数数据类型，该指令将用 0 填充较小的整数数据类型的高位，以使它们的字节大小与最大的数据类型相同。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	该指令执行按位非运算。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。



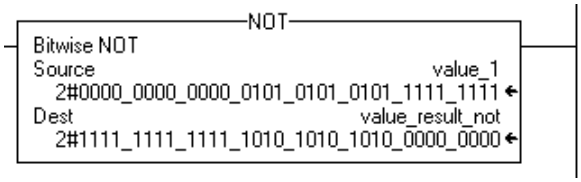
功能模块图

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 启用时， NOT 指令对 Source 执行按位非运算并将结果放入 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

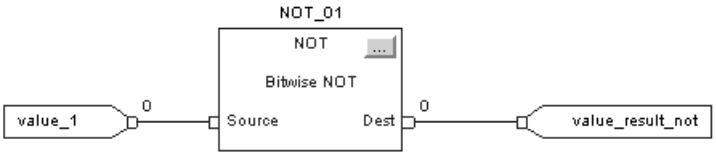
梯形图



结构化文本

```
value_result_not := NOT value_1;
```

功能块



布尔与 (Boolean AND, BAND)

BAND 指令最多可以对 8 个布尔输入执行逻辑与运算。

要执行按位与运算，请参见第 第 6 章 -23 页。

操作数：



结构化文本

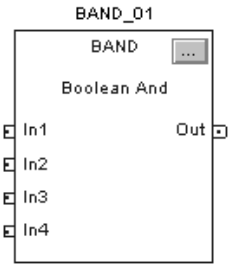
```
IF operandA AND operandB THEN
    <statement>;
END_IF;
```

在表达式中使用 AND 或符号 “&” 作为运算符。操作数必须是 BOOL 值或计算结果为 BOOL 值的表达式。该表达式计算 *operandA* 和 *operandB* 是否均为置位状态 (true)。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。



功能块



操作数：	类型：	格式：	说明：
BAND 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 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。
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	输出使能。
Out	BOOL	该指令的输出。

说明： BAND 指令最多可以对 8 个布尔输入执行与运算。如果不使用输入，它默认为置位状态 (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	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

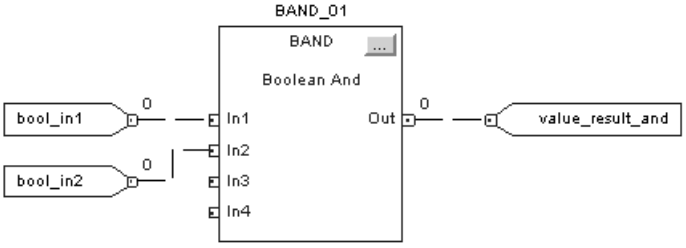
示例 1： 该示例对 *bool_in1* 和 *bool_in2* 执行与运算并将结果放入 *value_result_and* 中。

如果 bool_in1 是：	bool_in2 是：	则 value_result_and 为：
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* 均为置位状态 (true)，则 *light1* 为置位状态 (on)。否则， *light1* 清零 (off)。

结构化文本

```
IF bool_in1 AND bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

布尔或 (Boolean OR, BOR)

BOR 指令最多可以对 8 个布尔输入执行逻辑或运算。

要执行按位或运算，请参见第 第 6 章 -26 页。

操作数：



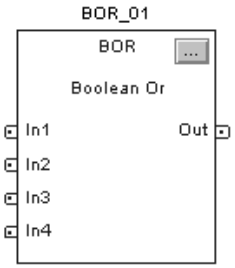
```
IF operandA OR operandB THEN
    <statement>;
END_IF;
```

结构化文本

在表达式中使用 OR 作为运算符。操作数必须是 BOOL 值或计算结果为 BOOL 值的表达式。该表达式计算 *operandA* 或 *operandB* 或者二者是否均为置位状态 (true)。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。





功能块

操作数：	类型：	格式：	说明：
BOR 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 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。
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	输出使能。
Out	BOOL	该指令的输出。

说明： BOR 指令最多可以对 8 个布尔输入执行或运算。如果不使用输入，则它默认为清零状态 (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	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

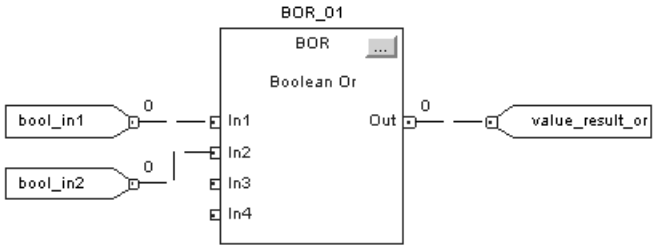
示例 1： 本示例对 *bool_in1* 和 *bool_in2* 执行或运算并将结果放入 *value_result_or* 中。

如果 bool_in1 是：	bool_in2 是：	则 value_result_or 为：
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： 在本例中，如果满足以下条件，则 *light1* 为置位状态 (on)：

- 仅设置 *bool_in1* (true)。
- 仅设置 *bool_in2* (true)。
- *bool_in1* 和 *bool_in2* 均为置位状态 (true)。

否则， *light1* 清零 (off)。

结构化文本

```
IF bool_in1 OR bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

布尔异或 (Boolean Exclusive OR, BXOR)

BXOR 对 2 个布尔输入执行异或运算。

要执行按位异或运算，请参见第 第 6 章 -29 页。

操作数：



结构化文本

```
IF operandA XOR operandB THEN
    <statement>;
END_IF;
```

在表达式中使用 XOR 作为运算符。操作数必须是 BOOL 值或计算结果为 BOOL 值的表达式。该表达式计算是否只有 *operandA* 或只有 *operandB* 为置位状态 (true)。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。



功能块



操作数：	类型：	格式：	说明：
BXOR 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 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	输出使能。
Out	BOOL	该指令的输出。

说明： BXOR 指令对 2 个布尔输入执行异或运算。

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	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

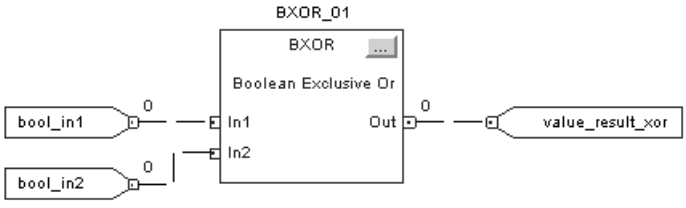
示例 1： 本示例对 *bool_in1* 和 *bool_in2* 执行异或运算并将结果放入 *value_result_xor* 中。

如果 bool_in1 是：	bool_in2 是：	则 value_result_xor 为：
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： 在本例中，如果满足以下条件，则 *light1* 为置位状态 (on)：

- 仅设置 *bool_in1* (true)。
- 仅设置 *bool_in2* (true)。

否则， *light1* 清零 (off)。

结构化文本

```
IF bool_in1 XOR bool_in2 THEN
    light1 := 1;
ELSE
    light1 := 0;
END_IF;
```

布尔非 (Boolean NOT, BNOT)

BNOT 指令对布尔输入取非。

要执行按位非运算，请参见第 第 6 章 -32 页。

操作数：



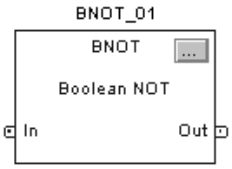
```
IF NOT operand THEN
    <statement>;
END_IF;
```

结构化文本

在表达式中使用 NOT 作为运算符。该操作数必须是布尔值或者计算结果为布尔值的表达式。该表达式计算 *operand* 是否为清除状态 (false)。

有关结构化文本内的表达式语法的信息，请参见 “结构化文本编程”。





功能模块图

操作数：	类型：	格式：	说明：
BNOT 标记	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	启用输入。如果为清除状态，则不执行指令，也不更新输出。 默认为置位状态。
In	BOOL	该指令的输入。 默认为置位状态。
输出参数：	**数据类型：**	**说明：**
EnableOut	BOOL	输出使能。
Out	BOOL	该指令的输出。

说明： BNOT 指令对布尔输入取非。

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	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清除状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

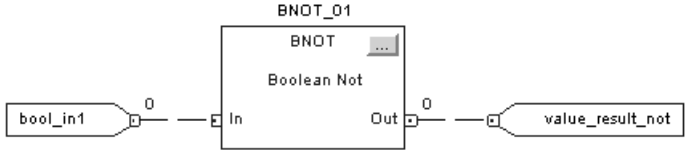
示例 1： 本示例对 *bool_in1* 求反并将结果放入 *value_result_not* 中。

如果 bool_in1 是：	则 value_result_not 为：
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* 为清除状态，则 *light1* 为清除状态 (off)。否则，*light1* 为置位状态 (on)。

结构化文本

```
IF NOT bool_in1 THEN
    light1 := 0;
ELSE
    light1 := 1;
END_IF;
```

说明：

数组（文件）/ 综合指令
（FAL、FSC、COP、CPS、FLL、AVE、SRT、STD、SIZE）

简介

文件 / 综合指令对数组数据进行操作。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
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)	第 7 章 -7
搜索并比较数组中的值	FSC	梯形图	第 7 章 -19
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	第 7 章 -28
不间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	第 7 章 -28
用特定数据填充数组	FLL	梯形图 结构化文本 (1)	第 7 章 -34
计算数组中一组值的平均值	AVE	梯形图 结构化文本 (1)	第 7 章 -38
将数组的某一维度中的数据按升序排列	SRT	梯形图 结构化文本	第 7 章 -43
计算数组中一组数据的标准偏差	STD	梯形图 结构化文本 (1)	第 7 章 -48
获取数组的某一维度的大小	SIZE	梯形图 结构化文本	第 7 章 -53

(1) 没有等效的结构化文本指令。使用其他结构化文本进行编程获得同样的结果。请参见该指令的说明。

您可以混用数据类型，但可能发生精度损失和舍入误差，并且指令需要更长的执行时间。检查 S:V 位，以确定结果是否被截断。

对于梯形图指令，**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样的最佳数据类型（通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

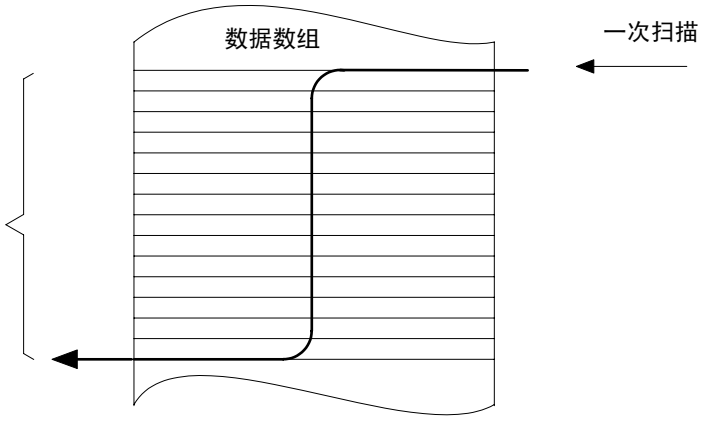
选择运算模式

对于 FAL 和 FSC 指令，模式告知控制器如何分配数组运算。

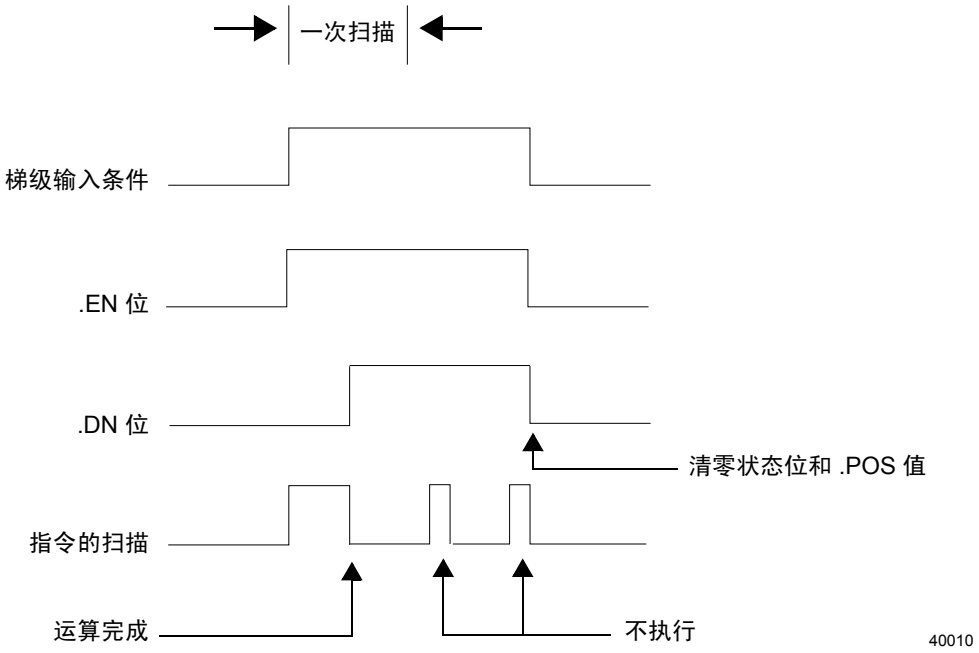
如果需要：	选择模式：
在继续执行下一条指令前对数组中的所有指定元素进行运算	整体
在几次扫描过程中分配数组运算 输入每次扫描要运算的元素数目 (1-2147483647)	数值
每当梯级输入条件从 false 变成 true 时对数组中的一个元素进行操作	增量

整体模式

在整体模式中，在继续执行下一条指令以前，数组中的所有指定元素都将进行运算。当指令的梯级输入条件从 false 变为 true 时开始进行运算。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 值时运算停止。



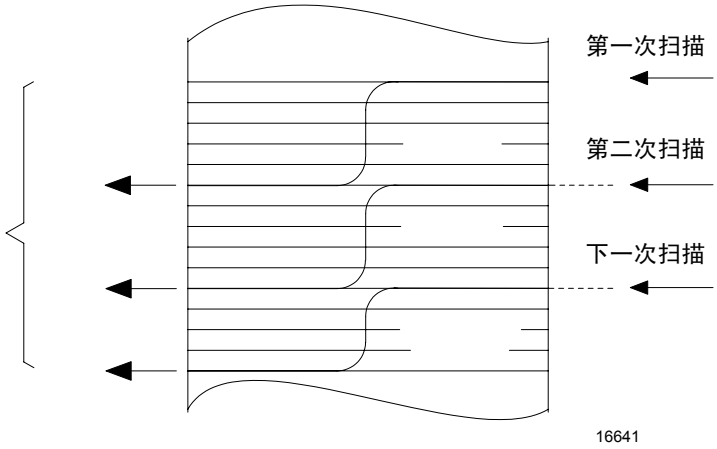
下面的时序图说明了状态位与指令运算之间的关系。当指令执行完成时，.DN 位将处于置位状态。当梯级输入条件为 false 时， .DN 位、 .EN 位和 .POS 值清零。只有此时，另一个指令的执行才能被梯级输入条件从 false 到 true 的转换触发。



数值模式

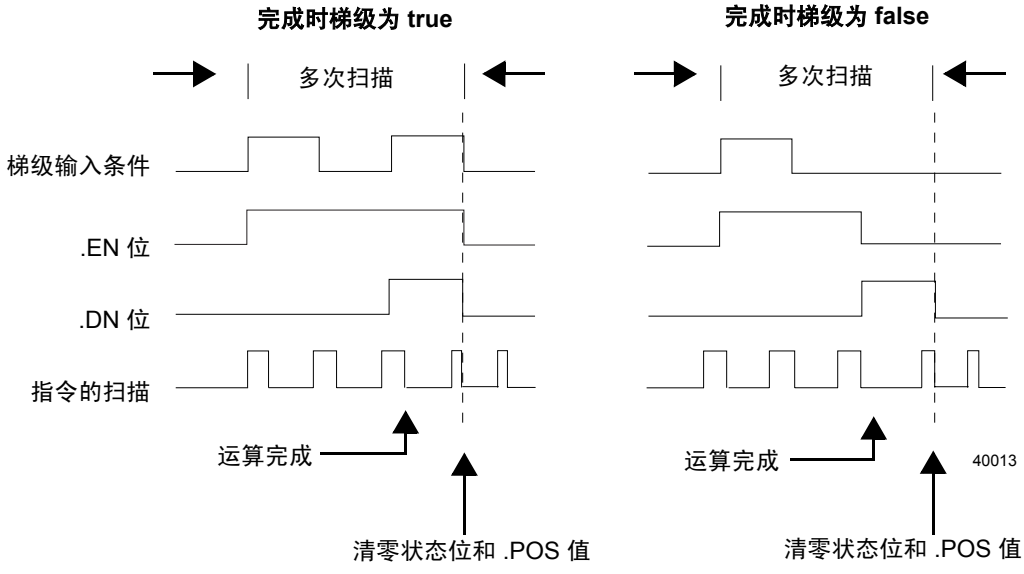
数值模式将数组运算分配给几次扫描过程。此模式对于使用非时间关键的数据或大量数据非常有用。您输入每次扫描时要运算的元素的数目，这样可以缩短扫描时间。

当梯级输入条件从 false 变成 true 时触发执行。一旦触发，指令将在每次扫描时执行，扫描次数为完成整个数组的运算所需的次数。一旦触发，梯级输入条件可以反复更改，而无需中断指令的执行。



重要事项 在 .DN 位处于置位状态之前，避免使用数值模式下文件指令运算的结果。

下面的时序图说明了状态位与指令运算之间的关系。当指令执行结束后， .DN 位将处于置位状态。

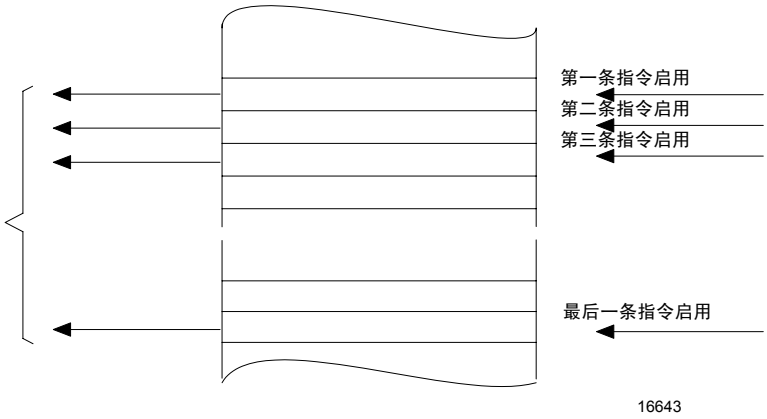


如果完成时梯级输入条件为 true， .EN 和 .DN 位将处于置位状态，直到梯级输入条件变成 false。当梯级输入条件变为 false 时，这些状态位和 .POS 值清零。

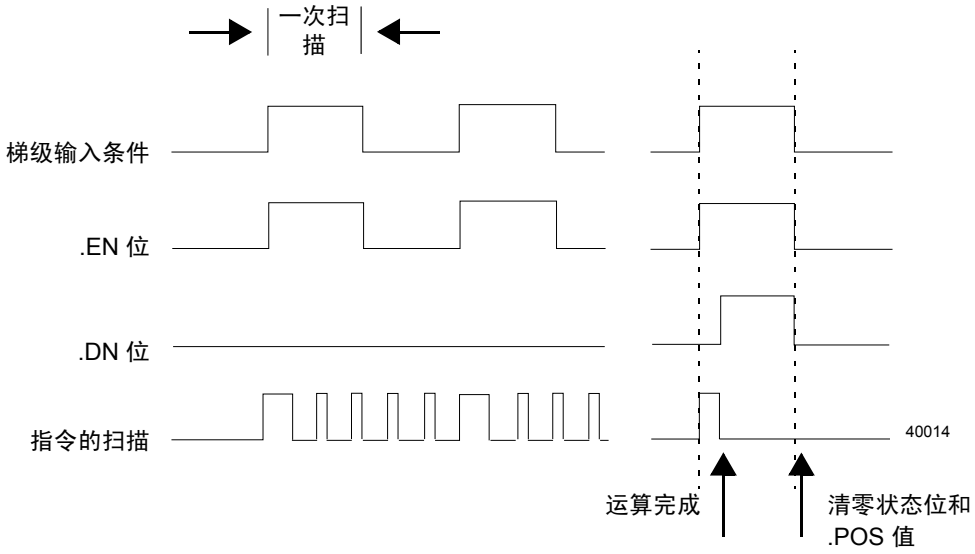
如果完成时梯级输入条件为 false， .EN 位立即清零。 .EN 位清零后经过一次扫描， .DN 位和 .POS 值清零。

增量模式

每当指令的梯级输入条件从 false 变成 true 时，增量模式就操作数组中的一个元素。



下面的时序图说明了状态位与指令运算之间的关系。只有在梯级输入条件从 false 变为 true 的扫描中才发生执行。每次执行时，只对数组中的一个元素进行操作。如果梯级输入条件保持 true 的过程中出现多次扫描，只在第一次扫描过程中执行指令。



当梯级输入条件为 true 时， .EN 将处于置位状态。当数组中的最后一个元素完成操作以后， .DN 位将处于置位状态。当最后一个元素完成操作并且梯级输入条件变成 false 时， .EN 位、 .DN 位和 .POS 值清零。

增量模式与每次扫描操作一个元素的数值模式的不同之处在于：

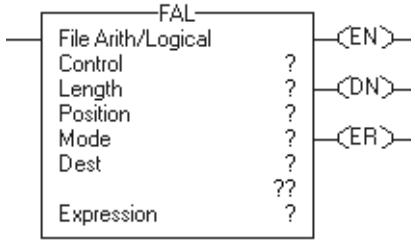
- 只需梯级输入条件从 false 到 true 的一次转换，每次扫描操作任意数目的元素的数值模式就可以开始执行。每次扫描指令都继续执行指定数目的元素，直到完成，而与梯级输入条件的状态无关。
- 增量模式需要梯级输入条件由 false 变成 true，才能操作数组中的一个元素。

文件算术和逻辑 (File Arithmetic and Logic, FAL)

FAL 指令对存储在数组中的数据执行复制、算术、逻辑和函数运算。

操作数：

梯形图



操作数：	类型：	格式：	说明：
CONTROL	CONTROL	标记	控制运算的结构
Length	DINT	立即数	数组中要操作的元素数
Position	DINT	立即数	数组中的当前元素 初始值通常为 0
Mode （模式）	DINT	立即数	如何分配运算 选择 INC、ALL，或输入一个数字
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记
Expression （表达式）	SINT INT **DINT** **REAL**	立即数 标记	表达式由标记和 / 或由运算符分开的立即数值组成
	SINT 或 INT 标记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，但是您可以使用 SIZE 指令和 FOR...DO 或其他循环结构实现相同的结果。

```
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的语法的信息，请参见 “附录 C”。

CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示 FAL 指令已启用。
.DN	BOOL	当指令完成对最后一个元素的操作时 (.POS = .LEN)，完成位将处于置位状态。
.ER	BOOL	如果表达式产生溢出 （S:V 处于置位状态），错误位将处于置位状态。该指令停止执行过程，直到程序清除 .ER 位为止。 .POS 值包含导致溢出的元素的位置。
.LEN	DINT	长度指定 FAL 指令操作的数组中的元素数目。
.POS	DINT	位置包含该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 第 7 章 -14 页开始的示例显示如何使用 .POS 值对数组执行单步操作。如果 Destination （目标）表达式的下标超出范围， FAL 指令将产生一个严重错误 （类型为 4，代码为 20）。

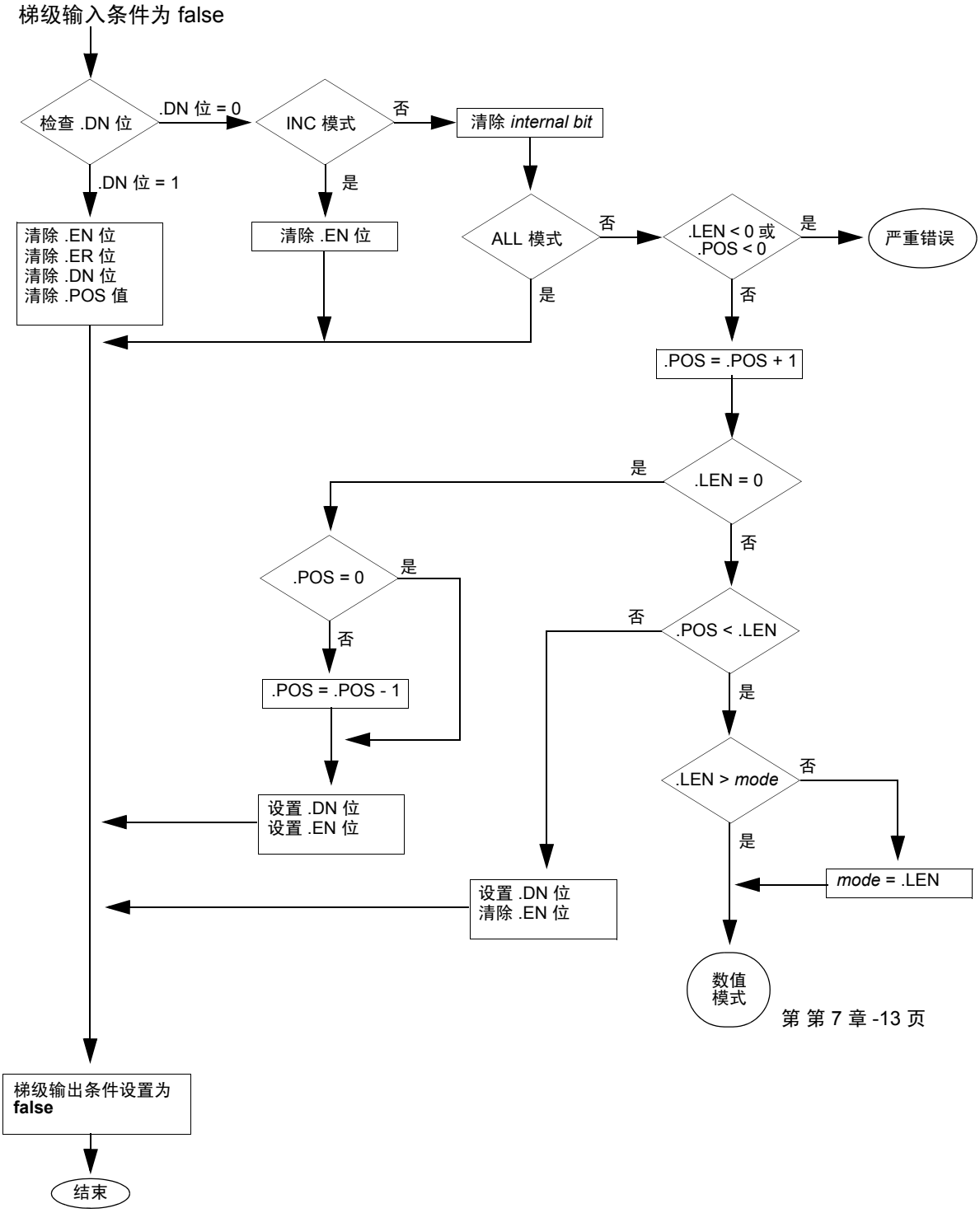
算术状态标志： 算术状态标志受影响。

错误条件：

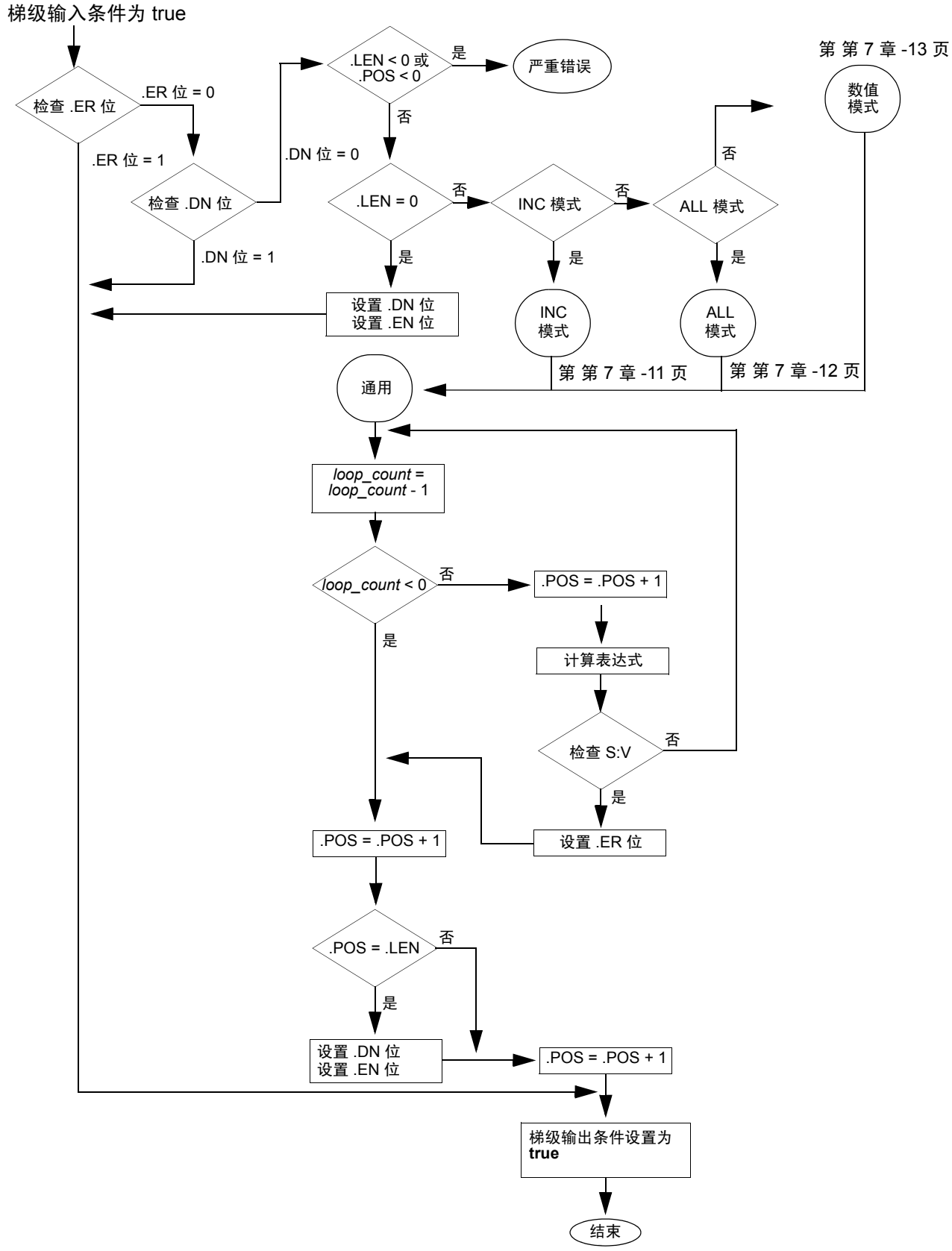
出现严重错误的条件：	错误类型：	错误代码：
下标超出范围	4	20
.POS < 0 或 .LEN < 0	4	21

执行：

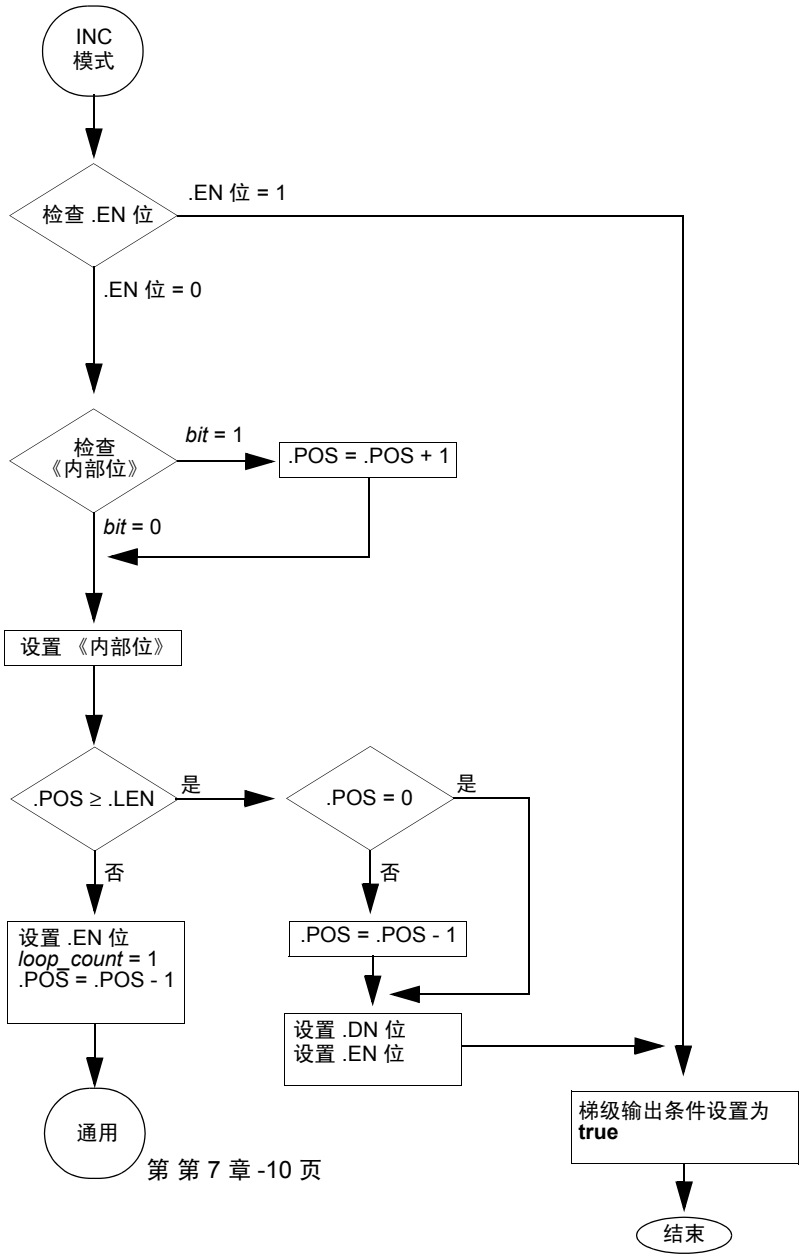
条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。



条件： **梯形图操作：**

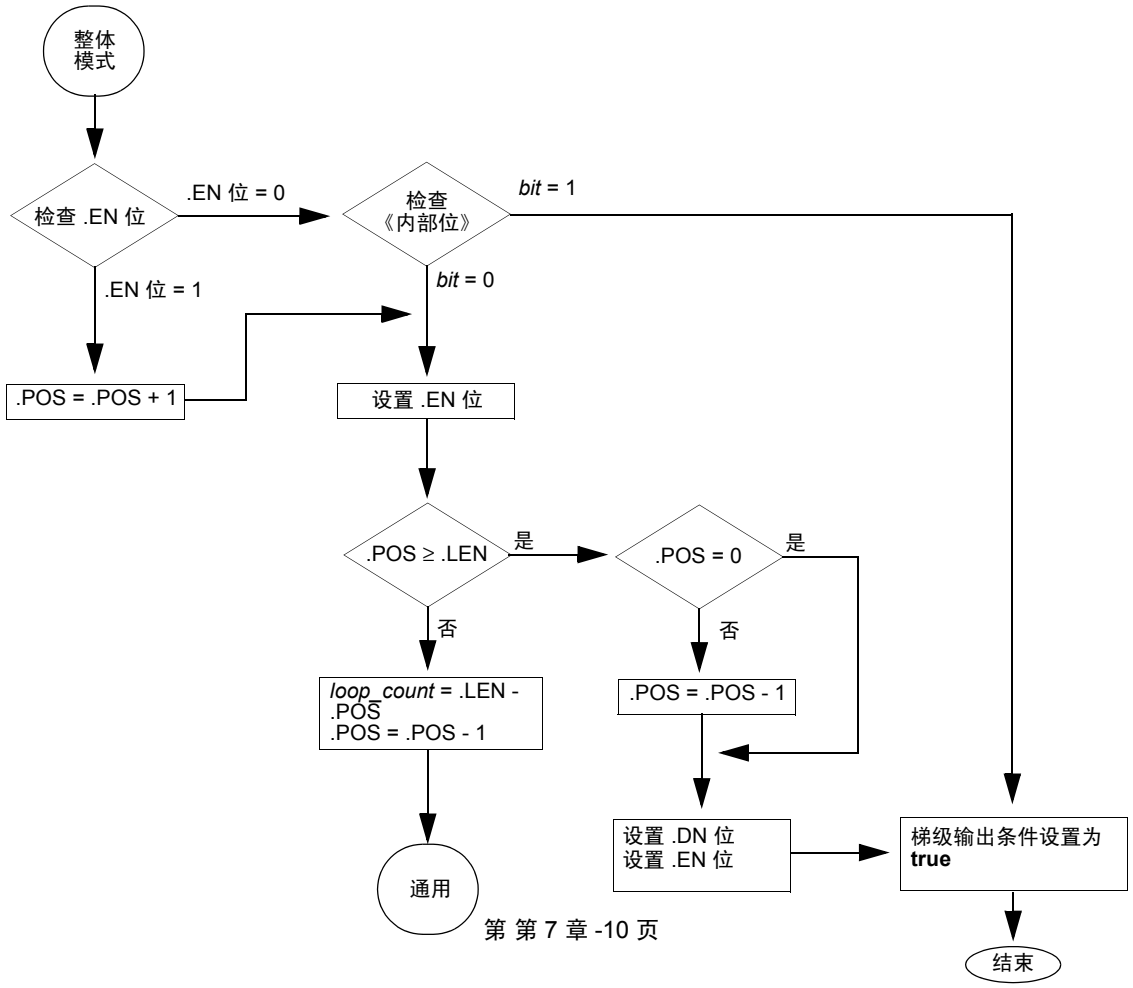

梯级输入条件为 true
检查 .ER 位
.ER 位 = 0
.ER 位 = 1
检查 .DN 位
.DN 位 = 0
.DN 位 = 1
.LEN < 0 或 .POS < 0
是
严重错误
否
.LEN = 0
否
是
INC 模式
否
是
ALL 模式
否
是
数值模式
第 第 7 章 -13 页
设置 .DN 位
设置 .EN 位
INC 模式
ALL 模式
第 第 7 章 -11 页
第 第 7 章 -12 页
通用
loop_count = loop_count - 1
loop_count < 0
否
.POS = .POS + 1
计算表达式
检查 S:V
否
是
是
.POS = .POS + 1
设置 .ER 位
.POS = .LEN
否
是
设置 .DN 位
设置 .EN 位
.POS = .POS + 1
梯级输出条件设置为 true
结束

条件：	梯形图操作：

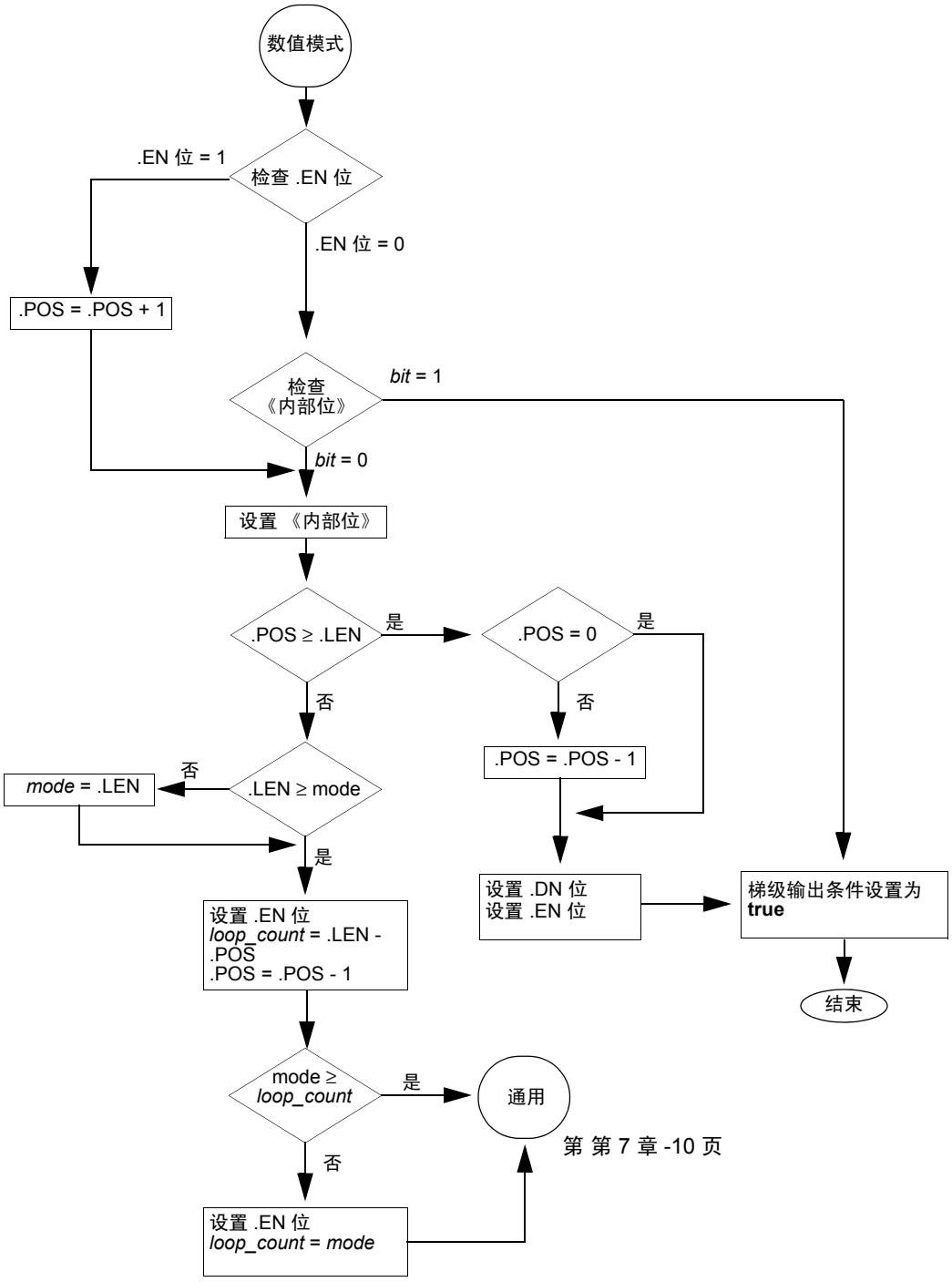


第 第 7 章 -10 页

条件：	梯形图操作：

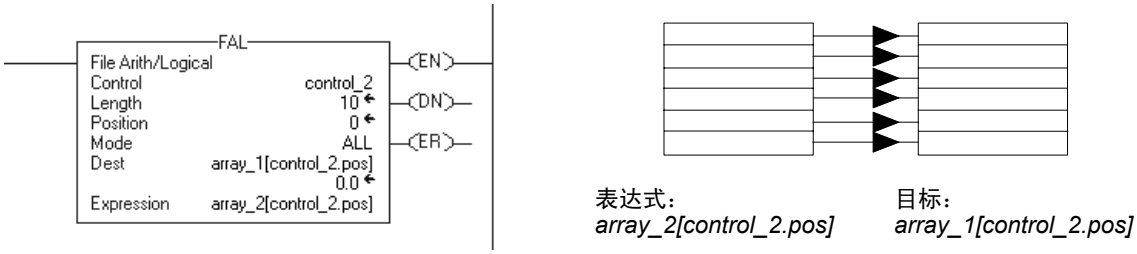

整体
模式
检查 .EN 位
.EN 位 = 0
检查
《内部位》
bit = 1
.EN 位 = 1
bit = 0
.POS = .POS + 1
设置 .EN 位
.POS ≥ .LEN
是
.POS = 0
是
否
否
loop_count = .LEN - .POS
.POS = .POS - 1
.POS = .POS - 1
设置 .DN 位
设置 .EN 位
梯级输出条件设置为 true
通用
第 第 7 章 -10 页
结束

条件：	梯形图操作：

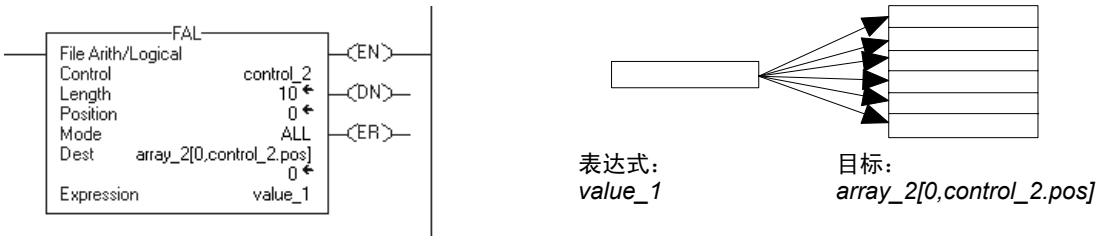


后扫描	梯级输出条件设置为 false。

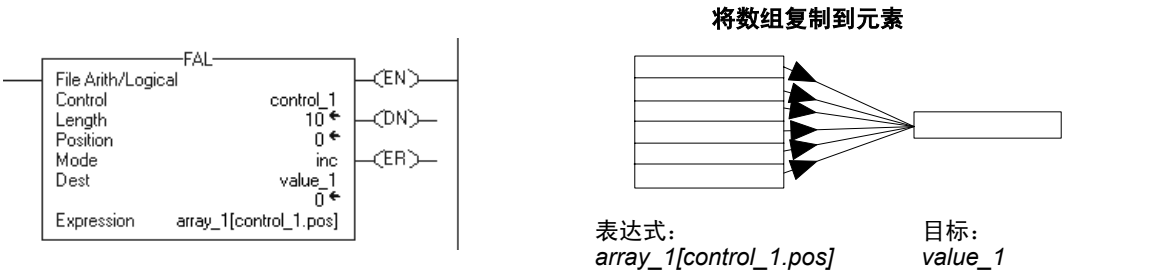
示例 1： 启用时， FAL 指令将 *array_2* 中的每个元素复制到 *array_1* 的相同位置中。



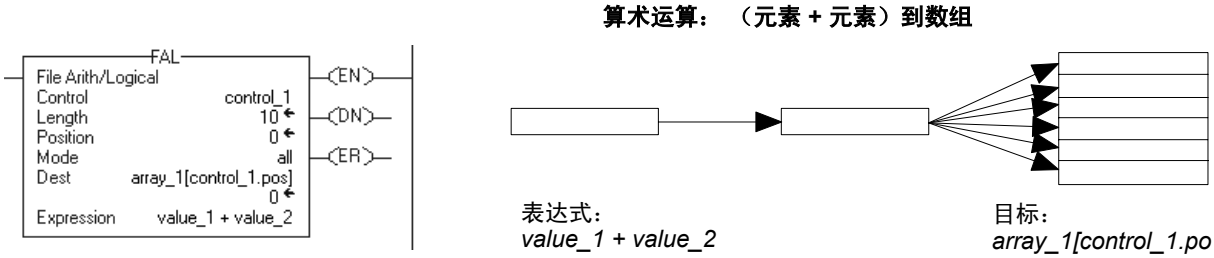
示例 2： 启用时， FAL 指令将 *value_1* 复制到 *array_2* 的第二维的前 10 个位置中。



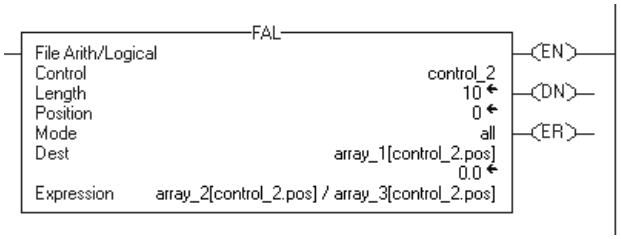
示例 3： 每当启用 FAL 指令时，它将 *array_1* 中的当前值复制到 *value_1* 中。因为 FAL 指令使用增量模式，所以每当启用指令时，只复制一个数组值。下次启用指令时，指令用 *array_1* 中的下一个值覆盖 *value_1*。



示例 4： 启用时， 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： 启用时， FAL 指令将 *array_2* 的当前位置的值除以 *array_3* 的当前位置的值，并将结果存储到 *array_1* 的当前位置中。



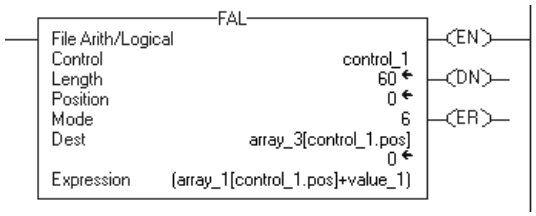
算术运算： （数组 / 数组）到数组



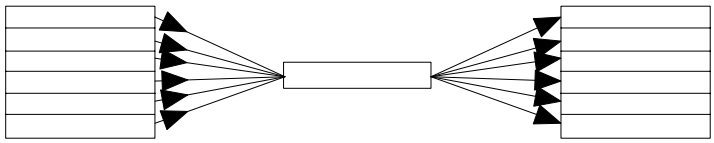
表达式：
array_2[control_2.pos] / array_3[control_2.pos]

目标：
array_1[control_2.pos]

示例 6： 启用时， FAL 指令将 *array_1* 的当前位置中的值与 *value_1* 相加，并将结果存储到 *array_3* 的当前位置中。如果要对 *array_1* 和 *array_3* 中的所有值进行操作，则指令必须执行 10 次。



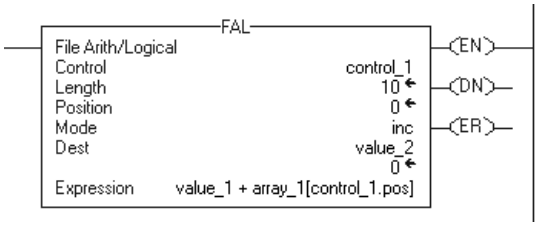
算术运算： （数组 + 元素）到数组



表达式：
array_1[control_1.pos] + value_1

目标：
array_3[control_1.pos]

示例 7： 每当启用 FAL 指令时，它将 *value_1* 与 *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，所以每次启用指令时，只有一个数组值与 *value_1* 相加。下次启用指令时，指令将覆盖 *value_2*。



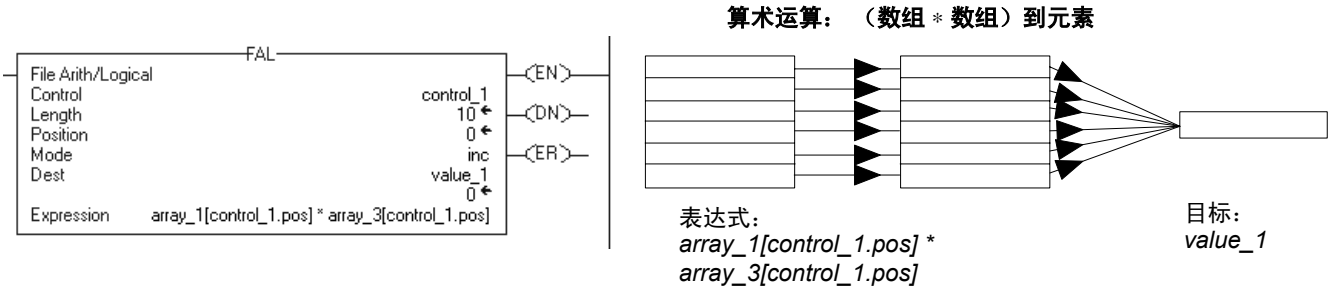
算术运算： （元素 + 数组）到元素



表达式：
value_1 + array_1[control_1.pos]

目标：
value_2

示例 8： 启用时， FAL 指令将 *array_1* 的当前值与 *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，所以每次启用指令时，只有一对数组值相乘。下次启用指令时，指令将覆盖 *value_1*。



FAL 表达式

您可以对 FAL 指令中的表达式进行编程，就像对 CPT 指令中的表达式编程一样。下面几节内容提供了有关有效的运算符、格式、运算顺序的信息，这些说明对于这两种指令都适用。

有效运算符

运算符：	说明：	最优：
+	加	DINT、 REAL
-	减 / 求负	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：	最优：
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

格式化表达式

对于表达式中使用的每个运算符，必须提供一个或两个操作数 （标记或立即值）。请根据下表设置表达式中运算符和操作数的格式：

运算符涉及的操作数的个数：	使用格式：	示例：
1	运算符 (操作数)	ABS (*tag_a*)
2	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*)

确定运算次序

指令按预定的次序执行写入表达式中的运算 （不一定是写入次序）。可以使用括号将运算条目分组，强制指令在执行其他运算之前先执行括号中的运算，从而更改运算次序。

次序相等的运算从左向右执行。

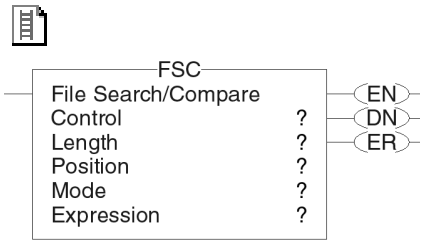
次序：	运算：
1.	( )
2.	ABS、 ACS、 ASN、 ATN、 COS、 DEG、FRD、LN、LOG、RAD、SIN、 SQR、 TAN、 TOD、 TRN
3.	**
4.	–（求负）、 NOT
5.	*、 /、 MOD
6.	–（减）、 +
7.	AND
8.	XOR
9.	OR

文件搜索和比较 (File Search and Compare, FSC)

FSC 指令比较数组中每个元素的值。

操作数：

梯形图



操作数：	类型：	格式：	说明：
CONTROL	CONTROL	标记	控制运算的结构
Length	DINT	立即数	数组中要操作的元素数
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	启用位指示 FSC 指令已启用。
.DN	BOOL	当指令完成对最后一个元素的操作时 (.POS = .LEN)，完成位将处于置位状态。
.ER	BOOL	不修改错误位。
.IN	BOOL	禁止位指示 FSC 指令检测到一个结果为 true 的比较。您必须将此位清零，才能继续搜索操作。
.FD	BOOL	发现位指示 FSC 指令检测到一个结果为 true 的比较。
.LEN	DINT	长度指定该指令所操作的数组元素的数目。
.POS	DINT	位置包含该指令正在访问的当前元素的位置。

说明： 当启用 FSC 指令且比较结果为 true 时，指令将设置 .FD 位，同时 .POS 位将反映指令找到结果为 true 的比较的数组位置。该指令将设置 .IN 位，阻止继续搜索。

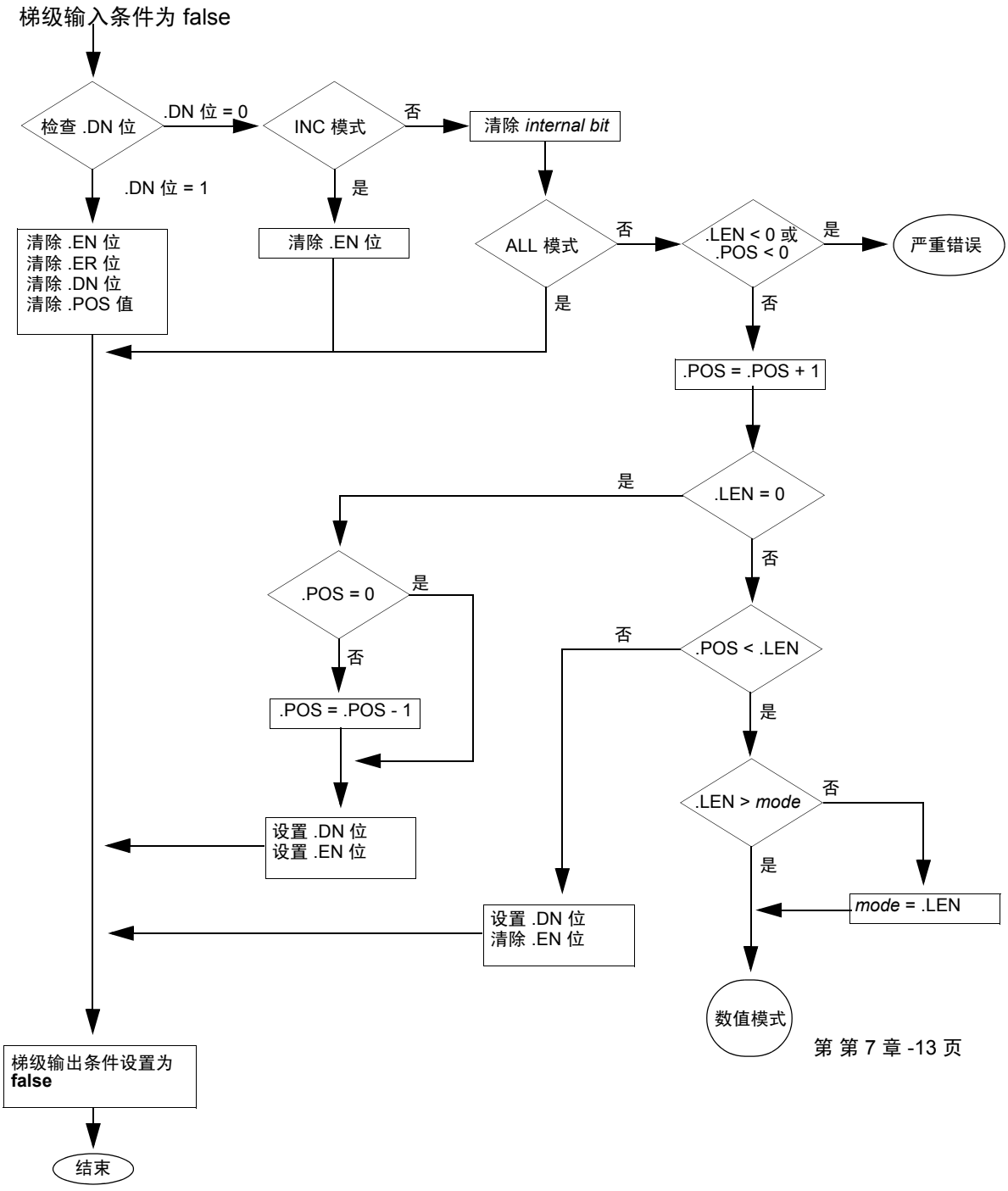
算术状态标志： 算术状态标志受影响。

错误条件：

出现严重错误的条件：	错误类型：	错误代码：
.POS < 0 或 .LEN < 0	4	21

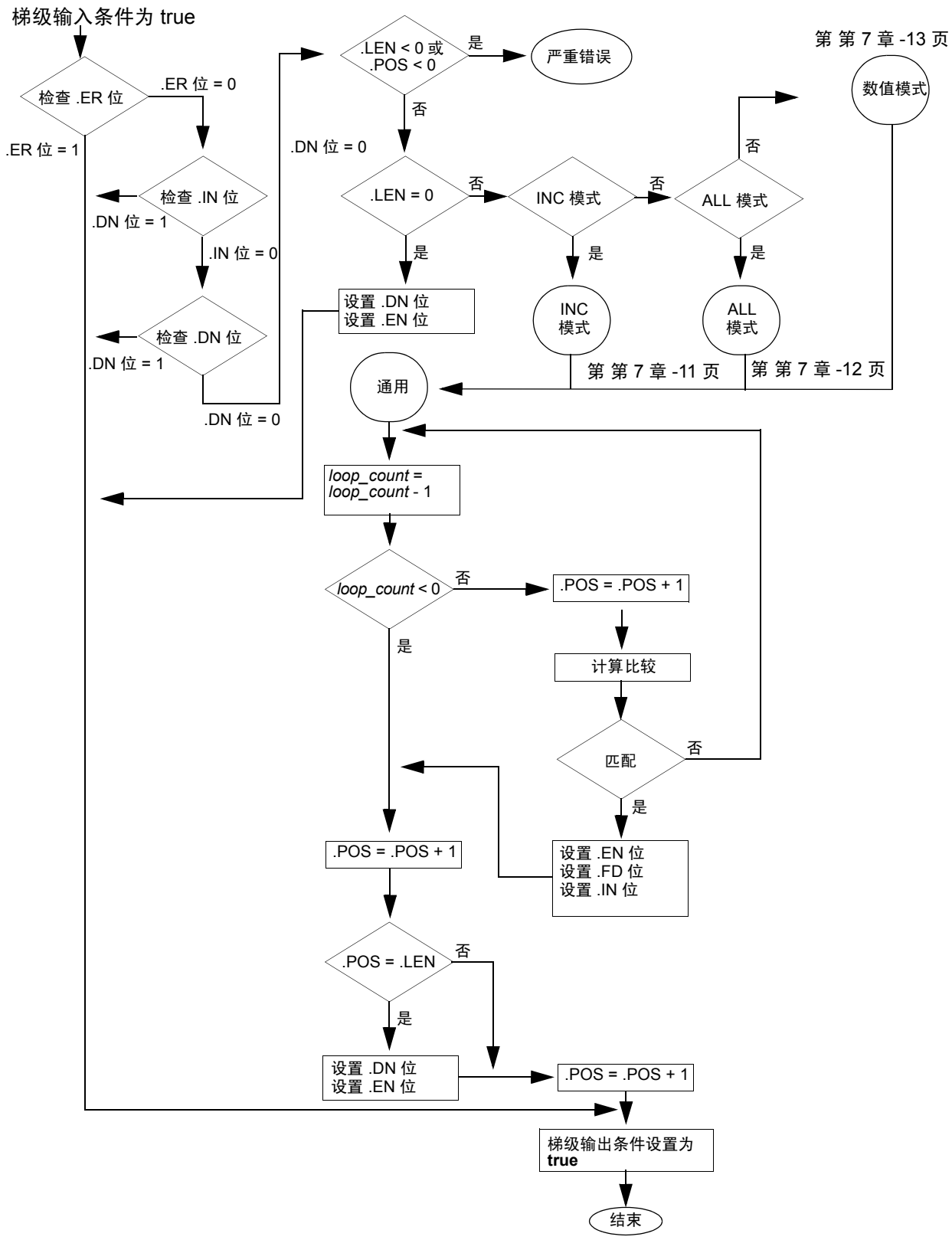
执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。



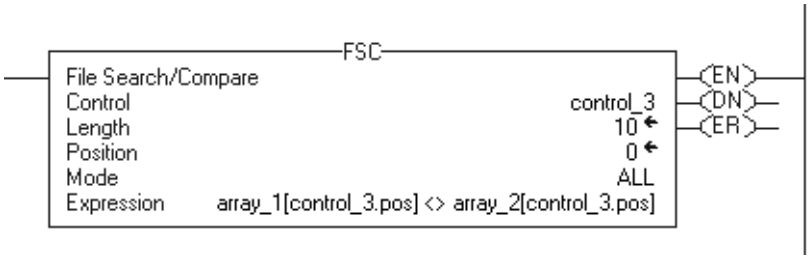
第 第 7 章 -13 页

条件：	**梯形图操作：**



后扫描	梯级输出条件设置为 false。

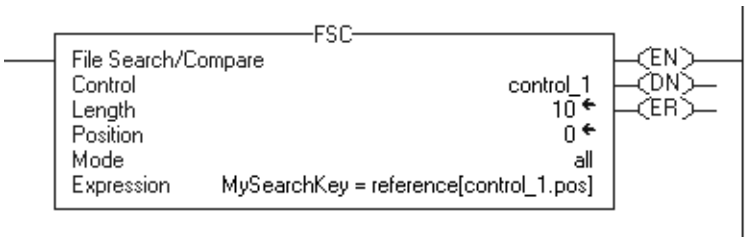
示例 1： 搜索两个数组之间的匹配处。启用时，FSC 指令将 *array_1* 中的前 10 个元素的每一个元素与 *array_2* 中的对应元素进行比较。



array_1	*array_2*	*control_3.pos*
00000000000000000000000000000000	00000000000000000000000000000000	0
00000000000000000000000000000000	00000000000000000000000000000000	1
00000000000000000000000000000000	00000000000000000000000000000000	2
00000000000000000000000000000000	00000000000000000000000000000000	3
00000000000000000111111111111111	11111111111111110000000000000000	4
11111111111111111111111111111111	11111111111111111111111111111111	5

FSC 指令发现这些元素不相等。该指令设置 .FD 和 .IN 位。 .POS 值 (4) 指示不相等的元素的位置。要继续比较数组的其余部分，请清除 .IN 位。

示例 2： 搜索数组中的匹配项。启用时， FSC 指令将 *MySearchKey* 与 *array_1* 中的 10 个元素进行比较。



MySearchKey

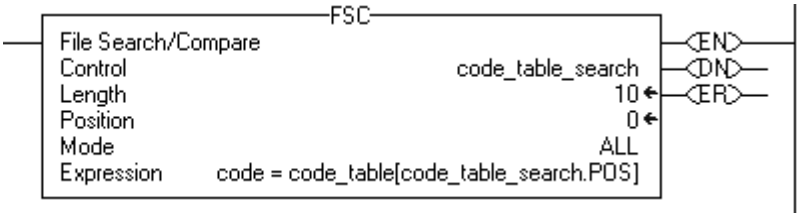
11111111111111110000000000000000

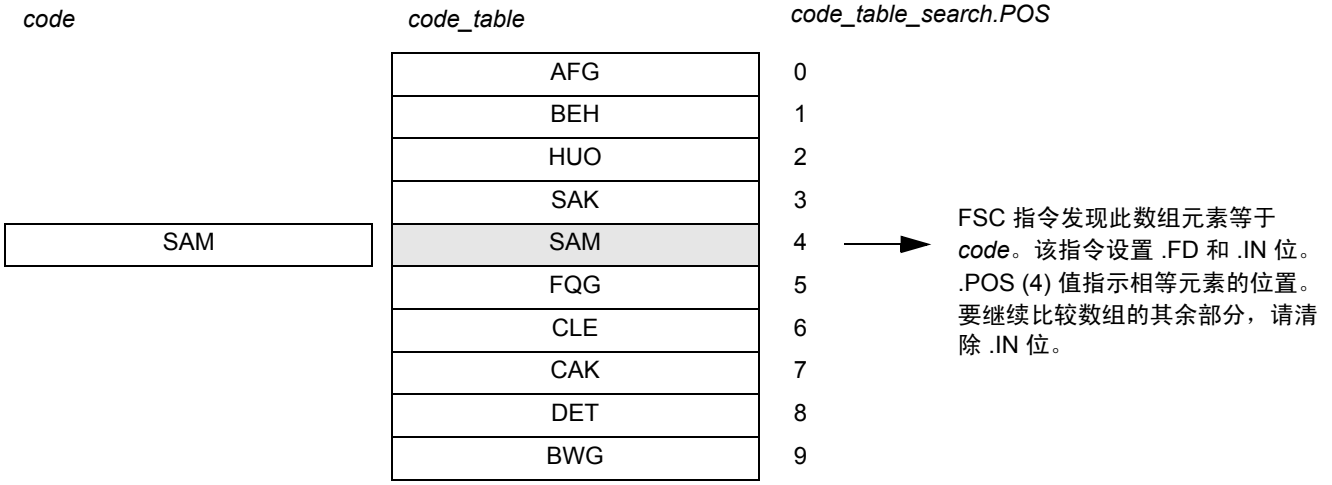
reference

	control_3.pos
00000000000000000000000000000000	0
00000000000000000000000000000000	1
00000000000000000000000000000000	2
00000000000000000000000000000000	3
11111111111111110000000000000000	4
11111111111111111111111111111111	5

FSC 指令发现此数组元素等于 *MySearchKey*。该指令设置 .FD 和 .IN 位。 .POS (4) 值指示相等元素的位置。要继续比较数组的其余部分，请清除 .IN 位。

示例 3： 在字符串数组中搜索字符串。启用时， FSC 指令将 *code* 中的字符与 *code_table* 中的 10 个元素进行比较。





FSC 表达式

您可以对 FSC 指令中的表达式进行编程，就像对 CMP 指令中的表达式编程一样。下面几节内容提供了有关有效的运算符、格式、运算顺序的信息，这些说明对于这两种指令都适用。

有效运算符

运算符：	说明：	最优：
+	加	DINT、 REAL
-	减 / 求负	DINT、 REAL
*	乘	DINT、 REAL
/	除	DINT、 REAL
=	等于	DINT、 REAL
<	小于	DINT、 REAL
<=	小于或等于	DINT、 REAL
>	大于	DINT、 REAL
>=	大于或等于	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：	最优：
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

格式化表达式

对于表达式中使用的每个运算符，必须提供一个或两个操作数 （标记或立即值）。请根据下表设置表达式中运算符和操作数的格式：

运算符涉及的操作数的个数：	使用格式：	示例：
1	运算符 (操作数)	ABS(*tag_a*)
2	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*)

确定运算次序

指令按预定的次序执行写入表达式中的运算 （不一定是写入次序）。可以使用括号将运算条目分组，强制指令在执行其他运算之前先执行括号中的运算，从而更改运算次序。

次序相等的运算从左向右执行。

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

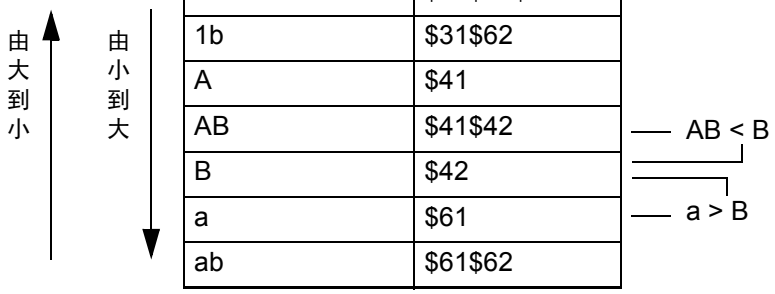
在表达式中使用字符串

可以使用梯形图或结构化文本表达式比较字符串数据类型。要在表达式中使用字符串，请遵循以下准则：

- 表达式可以比较两个字符串标记。
- 不能 在表达式中直接输入 ASCII 字符。
- 仅允许使用以下运算符

运算符：	说明：
=	等于
<	小于
<=	小于或等于
>	大于
>=	大于或等于
<>	不等于

- 如果两个字符串的字符相匹配，则这两个字符串相等。
- 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一个字符串是否小于或大于另一个字符串。要获知某字符的十六进制代码，请参见本手册的封底。
- 如果两个字符串按电话目录的方式排序，则字符串的顺序决定字符串的大小。



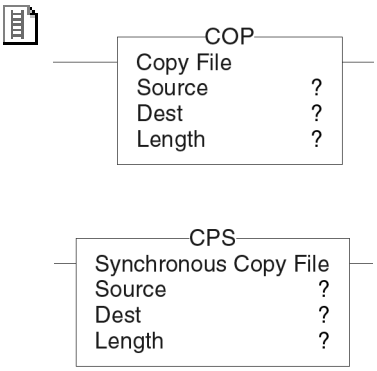
ASCII 字符	十六进制代码
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

复制文件 (COPY FILE, COP) 同步复制文件 (Synchronous Copy File, CPS)

COP 和 CPS 指令将 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** **string** 结构	标记	要复制的初始元素 **重要事项：**Source 操作数和 Destination 操作数的数据类型应当相同，否则可能产生意外的结果
Destination	**SINT** **INT** **DINT** **REAL** **string** 结构	标记	Source 要覆盖的第一个元素 **重要事项：**Source 操作数和 Destination 操作数的数据类型应当相同，否则可能产生意外的结果
Length	DINT	立即数 标记	要复制的 Destination 元素数目

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，其他控制器操作可能试图中断复制操作并更改源数据或目标数据：

如果源或目标为：	并且需要：	则选择：	说明：
• 生产者标记 • 消费者标记 • I/O 数据 • 另一个任务可以覆盖的数据	防止在复制操作过程中数据发生更改	CPS	• 延迟试图中断 CPS 指令的任务，直到指令完成。 • 若要估计 CPS 指令的执行时间，请参见 *ControlLogix System User Manual*（ControlLogix 系统用户手册），出版号 1756-UM001。
	允许在复制操作过程中数据发生更改	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，将为剩余元素复制无法预料的数据。

重要事项

您 **《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 指令对连续的内存进行操作。它们进行直接的字节到字节的内存复制。在某些情况下，它们将跨过数组写入标记的其他成员。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

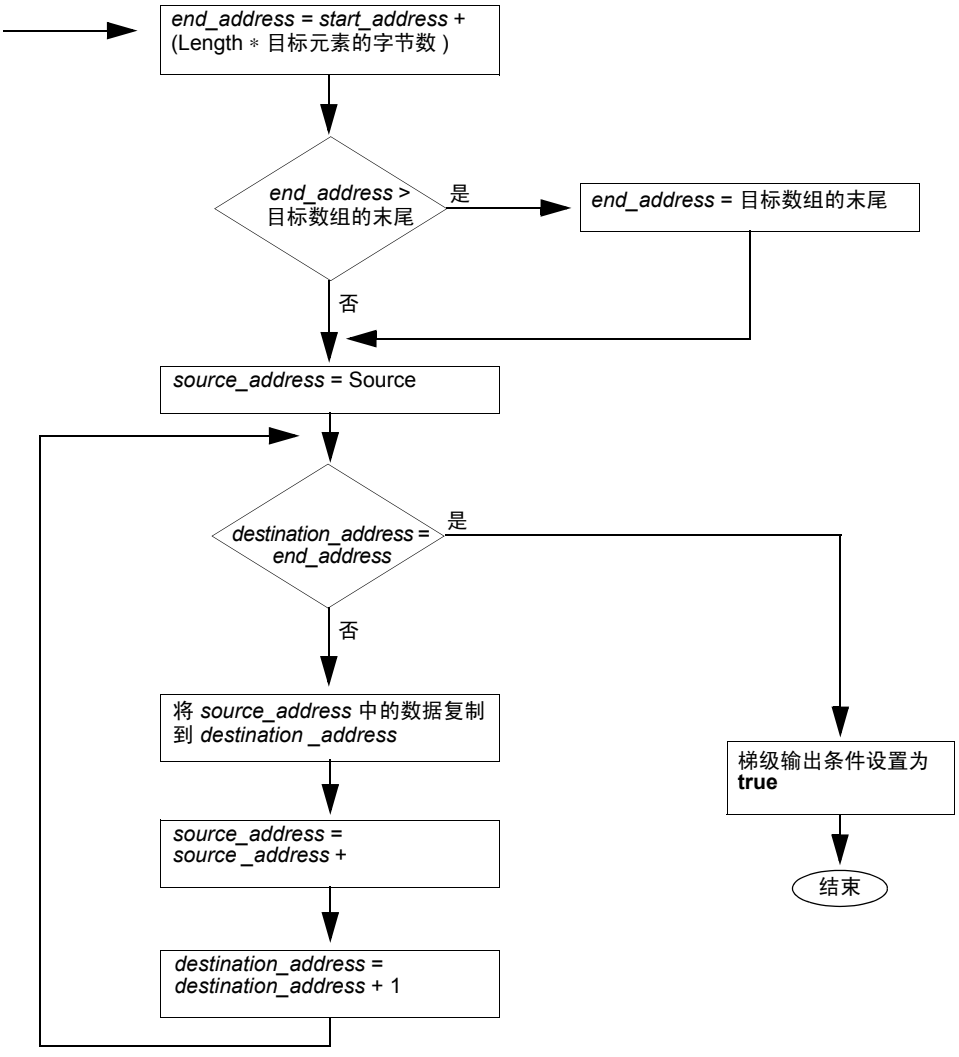
如果标记是	则
用户自定义的数据类型	如果长度过大，该指令将跨过数组末尾写入标记的其他成员。它停止于标记的末尾。未发生严重错误。
非用户自定义的数据类型	如果长度过大，该指令将停止于数组末尾。未发生严重错误。

如果长度值超过 Destination 数组中的总元素数，则表示长度过大。

算术状态标志： 不受影响

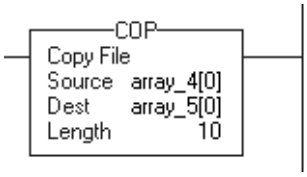
错误条件： 无

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行		
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例 1： *array_4* 和 *array_5* 的数据类型相同。启用时， COP 指令将 *array_4* 中的前 10 个元素复制到 *array_5* 中的前 10 个元素中。

梯形图

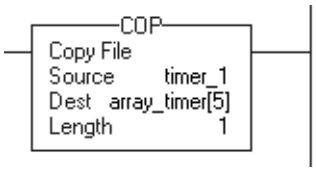


结构化文本

```
COP(array_4[0],array_5[0],10);
```

示例 2： 启用时， COP 指令将结构 *timer_1* 复制到 *array_timer* 的第 5 个元素中。指令只将一个结构复制到一个数组元素中。

梯形图



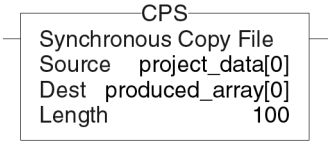
结构化文本

```
COP(timer_1,array_timer[5],1);
```

示例 3： *project_data* 数组 （100 个元素）存储各种在应用程序中的不同时间发生变化的值。若要将 *project_data* 某一时刻的完整图像及时发送给另一个控制器， CPS 指令将 *project_data* 复制到 *produced_array*。

- 当 CPS 指令复制数据时，任何 I/O 更新或其他任务都不能更改该数据。
- *produced_array* 标记在 ControlNet 网络上生成数据，供其他控制器使用。
- 若要使用数据的相同图像 （即数据的同步副本），使用控制器将用 CPS 指令将数据从消费者标记复制到另一个标记，以便在应用程序中使用。

梯形图



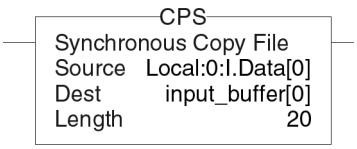
结构化文本

```
CPS(project_data[0],produced_array[0],100);
```

示例 4： *Local:0:I.Data* 存储与槽 0 中 1756-DNB 模块相连的 DeviceNet 网络的输入数据。为了使输入与应用程序同步，CPS 指令将输入数据复制到 *input_buffer*。

- 当 CPS 指令复制数据时，任何 I/O 更新都不能更改该数据。
- 当应用程序执行时，它将 *input_buffer* 中的输入数据作为输入。

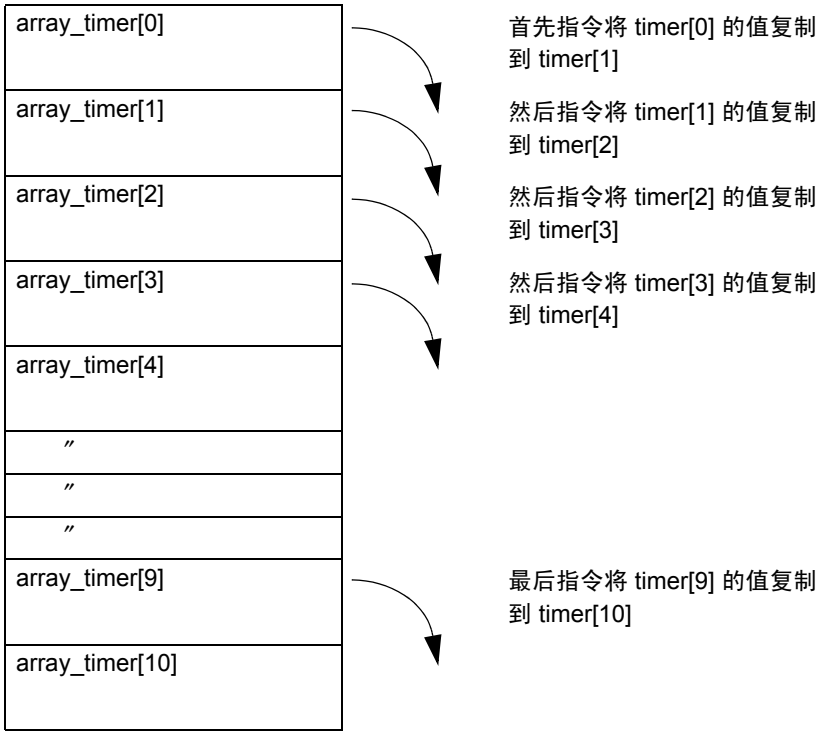
梯形图



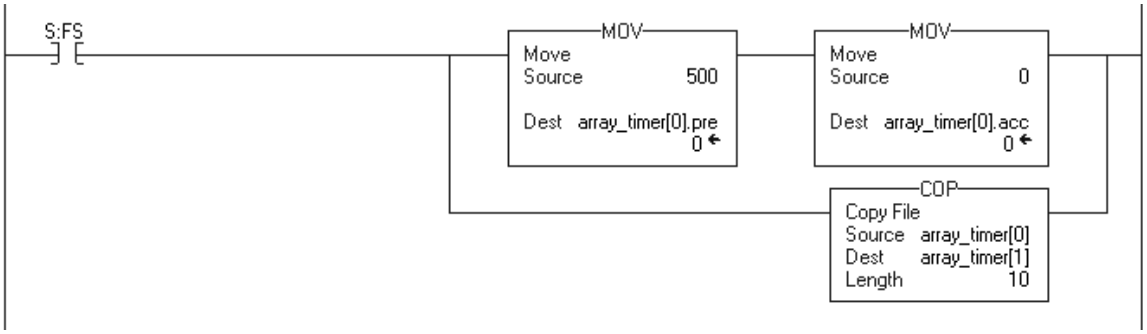
结构化文本

```
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。启用时， MOV 指令初始化第一个 *array_timer* 元素的 .PRE 和 .ACC 值。启用时， COP 指令复制从 *array_timer[0]* 开始的连续字节块。长度是九个计时器结构。



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);

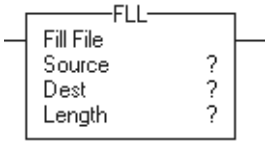
END_IF;
```

文件填充(File Fill, 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 标记	要复制的元素 **重要事项：** Source 操作数和 Destination 操作数的数据类型应当相同，否则可能产生意外的结果
Destination	**SINT** **INT** **DINT** **REAL** 结构	标记	Source 要覆盖的第一个元素 **重要事项：** Source 操作数和 Destination 操作数的数据类型应当相同，否则可能产生意外的结果 初始化结构的首选方式是使用 COP 指令。
Length	DINT	立即数	要填充的元素的数目

结构化文本

结构化文本没有 FLL 指令，但是您可以使用 SIZE 指令和 FOR...DO 或其他循环结构实现相同的结果。

```
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的语法的信息，请参见 “附录 C”。

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

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

FLL 指令对连续的数据内存进行操作。在某些情况下，该指令将跨过数组写入标记的其他成员。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

如果标记是	则
用户自定义的数据类型	如果长度过大，该指令将跨过数组末尾写入标记的其他成员。它停止于标记的末尾。未发生严重错误。
非用户自定义的数据类型	如果长度过大，该指令将停止于数组末尾。未发生严重错误。

如果长度值超过 Destination 数组中的总元素数，则表示长度过大。

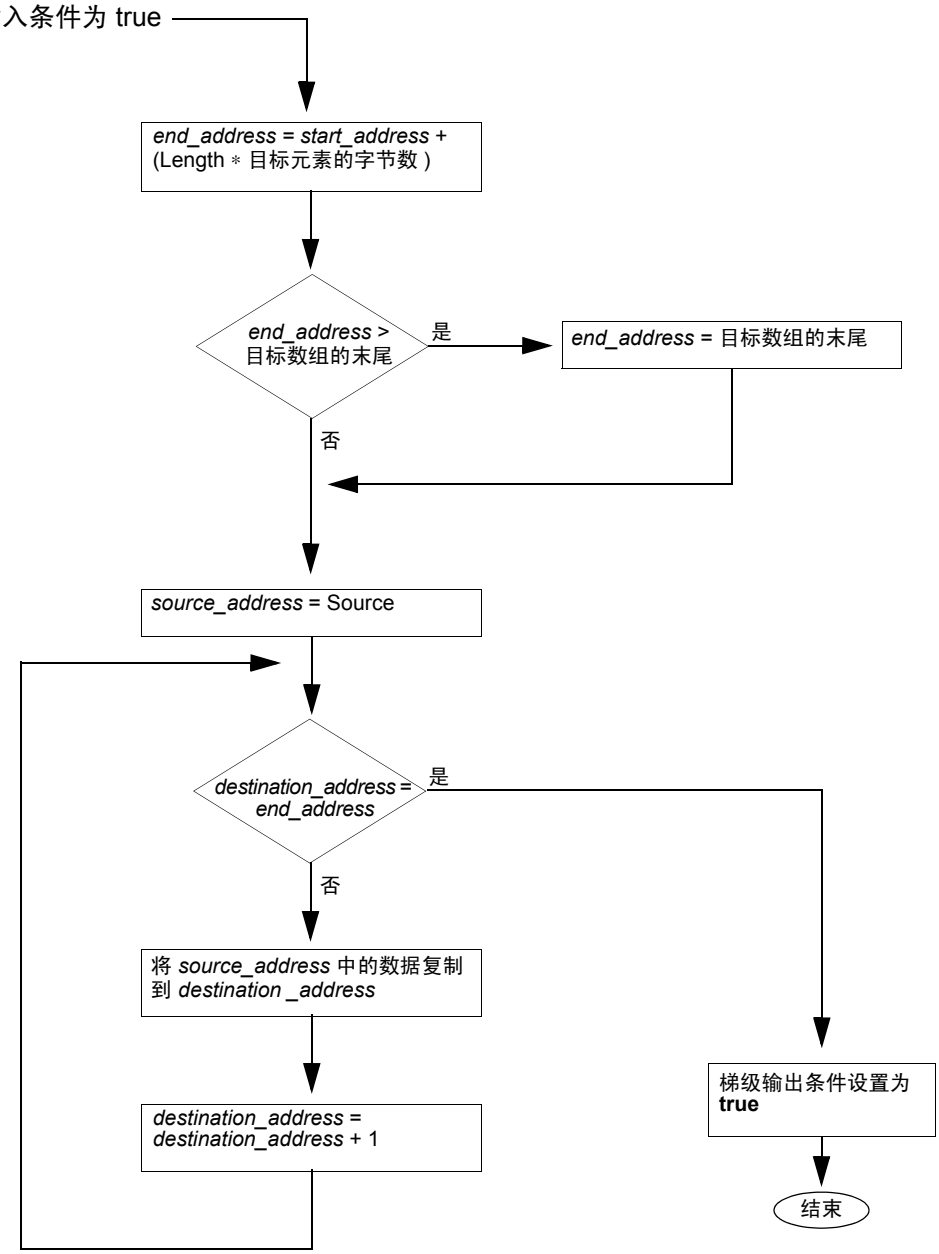
为了获得最好的结果， Source 和 Destination 应为同一类型。如果需要填充一个结构，请使用 COP 指令 （请参见第 第 7 章 -32 页中的示例 3）。如果您将 Source 和 Destination 的数据类型混用，则转换后的 Source 值会填充 Destination 元素。

如果 Source 为：	Destination 为：	Source 将被转换为：
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

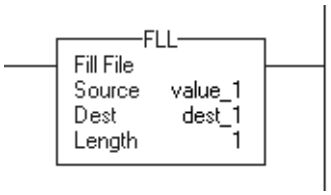
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	
后扫描	梯级输出条件设置为 false。

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：**	**FLL 指令执行后的 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

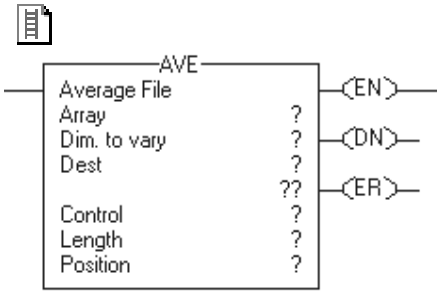
```
dest_1 := value_1;
```

文件平均值 (File Average, 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**	数组标记	计算此数组中的值的平均值 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)	使用哪个维度 取决于维数，顺序是 array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Destination	SINT INT DINT **REAL**	标记	运算结果
CONTROL	CONTROL	标记	控制运算的结构
Length	DINT	立即数	要计算平均值的数组的元素数目
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

结构化文本没有 AVE 指令，但是您可以使用 SIZE 指令和 FOR...DO 或其他循环结构实现相同的结果。

```
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的语法的信息，请参见 “附录 C”。

CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示 AVE 指令已启用。
.DN	BOOL	当指令完成对数组中最后一个元素的操作时 (.POS = .LEN)，完成位将处于置位状态。
.ER	BOOL	当该指令产生溢出时，错误位将处于置位状态。该指令停止执行过程，直到程序清除 .ER 位为止。导致溢出的元素的位置存储在 .POS 值中。
.LEN	DINT	长度指定该指令所操作的数组元素的数目。
.POS	DINT	位置包含该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。

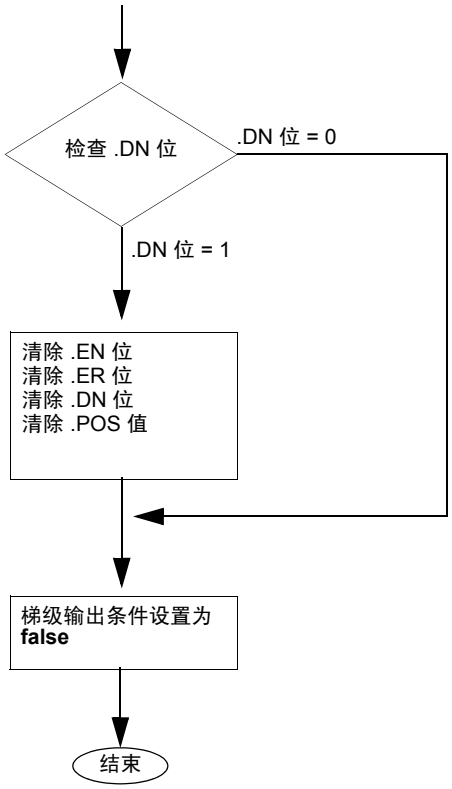
重要事项 确保长度不会导致该指令超出指定的 Dimension to vary。如果发生此情况， Destination 将不正确。

算术状态标志： 算术状态标志受影响。

错误条件：

出现严重错误的条件：	错误类型：	错误代码：
.POS < 0 或 .LEN < 0	4	21
对于指定的数组， Dimension to vary 不存在	4	20

执行：

条件：	梯形图操作：
预扫描	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。 梯级输出条件设置为 false。
梯级输入条件为 false	
梯级输入条件为 true	AVE 指令通过将数组中的所有指定元素相加并除以元素的数目来计算平均值。 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 = ALL 有关 FAL 指令如何执行的详细信息，请参见第 第 7 章 -9 页。
后扫描	梯级输出条件设置为 false。

示例 1： 计算 *array_dint* 的平均值，它是 DINT[4,5]。

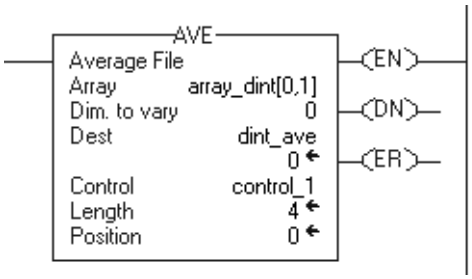
维度 1

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

$$AVE = \frac{19 + 14 + 9 + 4}{4} = \frac{46}{4} = 11.5$$

$dint\_ave = 12$

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，它是 DINT[4,5]。

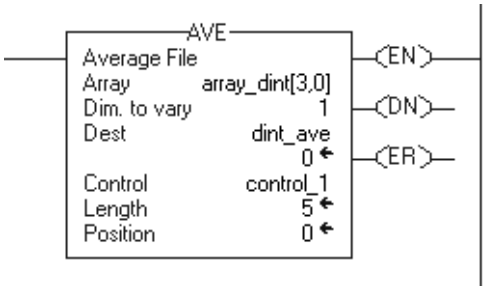
维度 1

维度 0 / 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

$$AVE = \frac{5+4+3+2+1}{5} = \frac{15}{5} = 3$$

$$dint\_ave = 3$$

梯形图



结构化文本

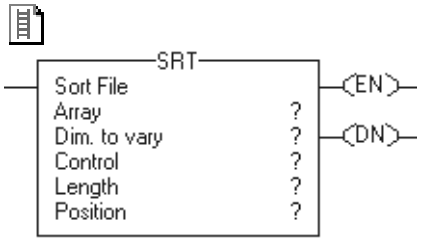
```
SIZE(array_dint,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;
```

文件排序 (File Sort, SRT)

SRT 指令将数组的某一维度 (Dim to vary) 中的值的集合按升序排列。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Array	**SINT** **INT** **DINT** **REAL**	数组标记	要排序的数组 指定要排序的元素组中的第一个元素 **不要** 在下标中使用 CONTROL.POS
Dimension to vary	DINT	立即数 (0, 1, 2)	使用哪个维度 取决于维数，顺序是 array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
CONTROL	CONTROL	标记	控制运算的结构
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。不过，可以通过访问 CONTROL 结构的 .LEN 和 .POS 成员指定 Length 和 Position 值，而不必在操作数列表中提供这些值。

CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示 SRT 指令已启用。
.DN	BOOL	当指定的元素完成排序以后，完成位将处于置位状态。
.ER	BOOL	当 .LEN < 0 或 .POS < 0 时，错误位将处于置位状态。这两种情况中的任意一种都将产生严重错误。
.LEN	DINT	长度指定指令要排序的数组的元素数目
.POS	DINT	位置包含该指令正在访问的当前元素的位置。

说明： SRT 指令将数组的某一维度 (Dim to vary) 中的值的集合按升序排列。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

SRT 指令对连续的内存进行操作。在某些情况下，该指令会更改标记的其他成员中的数据。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

重要事项

确保长度不会导致该指令超出指定的 Dimension to vary。如果出现这种情况，将会产生意外的结果。

这是一个可转换指令：

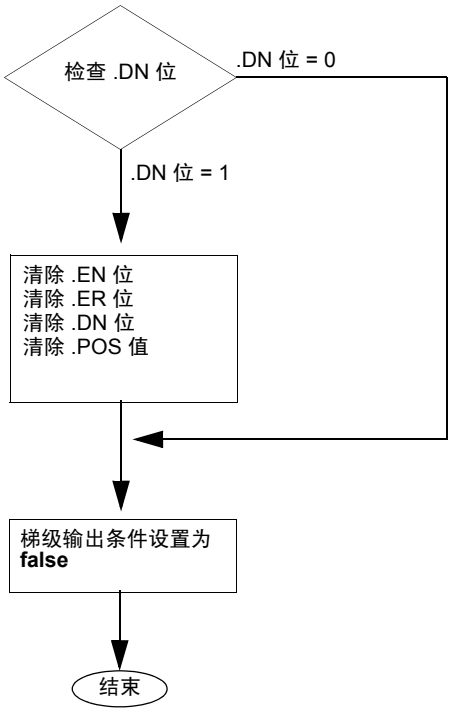
- 在梯形图中，每次应执行指令时都将梯级输入条件由清零状态切换为置位状态。
- 在结构化文本中，限制指令只对转换执行。请参见“附录 C”。

算术状态标志： 算术状态标志受影响。

错误条件：

出现严重错误的条件：	错误类型：	错误代码：
.POS < 0 或 .LEN < 0	4	21
对于指定的数组， Dimension to vary 不存在	4	20
指令试图访问数组范围以外的数据	4	20

执行：

条件：	梯形图操作：	结构化文本操作：
预扫描	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。 梯级输出条件设置为 false。	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。
梯级输入条件为 false		不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
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 false。	不采取任何操作。

示例 1： 对 *int* _array 进行排序，它是 DINT[4,5]。

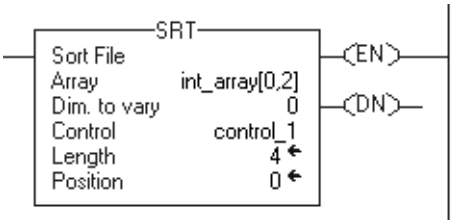
排序前

维度 0 \ 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 \ 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： 对 *int* _array 进行排序，它是 DINT[4,5]。

排序前

维度 1

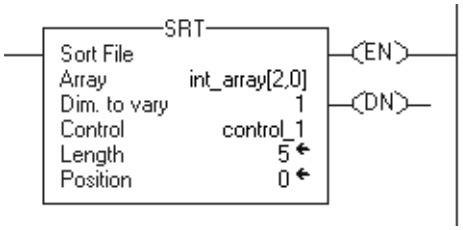
维度 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

排序后

维度 1

维度 0	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;

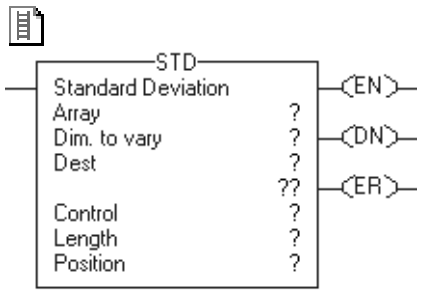
SRT(int_array[2,0],1,control_1);
```

文件标准偏差 (File Standard Deviation, STD)

STD 指令计算数组的某一维度中的值集合的标准偏差，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** SINT 或 INT 标记通过符号扩展转换为 DINT 值。	数组标记	获取此数组中的值的标准偏差 指定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
Dimension to vary	DINT	立即数 (0, 1, 2)	使用哪个维度 取决于维数，顺序是 array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Destination	REAL	标记	运算结果
CONTROL	CONTROL	标记	控制运算的结构
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	启用位指示 STD 指令已启用。
.DN	BOOL	当计算完成时，完成位将处于置位状态。
.ER	BOOL	当该指令产生溢出时，错误位将处于置位状态。该指令停止执行过程，直到程序清除 .ER 位为止。导致溢出的元素的位置存储在 .POS 值中。
.LEN	DINT	长度指定该指令所操作的数组元素的数目。
.POS	DINT	位置包含该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，但是您可以使用 SIZE 指令和 FOR...DO 或其他循环结构实现相同的结果。

```
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的语法的信息，请参见 “附录 C”。

说明： 标准偏差根据此公式计算：

$$\text{标准偏差} = \sqrt{\frac{\left(\sum_{i=1}^{N}\left[\langle X_{(start+i)} - AVE\rangle^{2}\right]\right)}{(N-1)}}$$

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 = $\dfrac{\left(\sum_{i=1}^{N} X_{(start+i)}\right)}{N}$

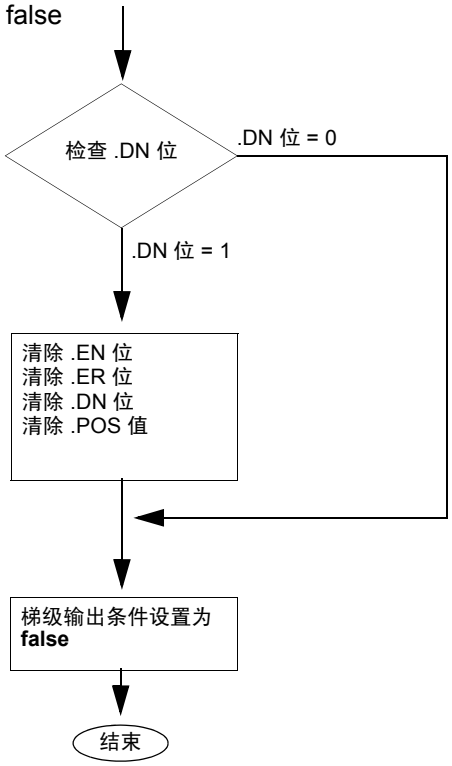
重要事项 确保长度不会导致该指令超出指定的 Dimension to vary。如果发生此情况， Destination 将不正确。

算术状态标志： 算术状态标志受影响。

错误条件：

出现严重错误的条件：	错误类型：	错误代码：
.POS < 0 或 .LEN < 0	4	21
对于指定的数组，Dimension to vary 不存在	4	20

执行：

条件：	梯形图操作：
预扫描	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。 梯级输出条件设置为 false。
梯级输入条件为 false	
梯级输入条件为 true	STD 指令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 = ALL 有关 FAL 指令如何执行的详细信息，请参见第 第 7 章 -9 页。
后扫描	梯级输出条件设置为 false。

示例 1： 计算 *dint_array* 的标准偏差，它是 DINT[4,5]。

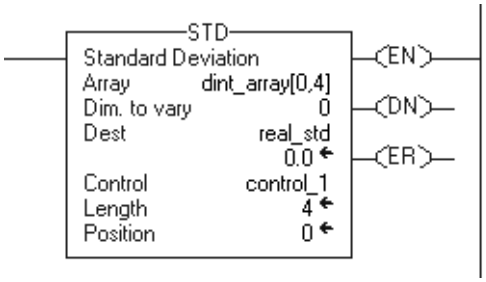
维度 0 \ 维度 1	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$AVE = \frac{16 + 11 + 6 + 1}{4} = \frac{34}{4} = 8.5$$

$$STD = \sqrt{\frac{\langle 16 - 8.5 \rangle^2 + \langle 11 - 8.5 \rangle^2 + \langle 6 - 8.5 \rangle^2 + \langle 1 - 8.5 \rangle^2}{\langle 4 - 1 \rangle}} = 6.454972$$

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，它是 DINT[4,5]。

维度 1

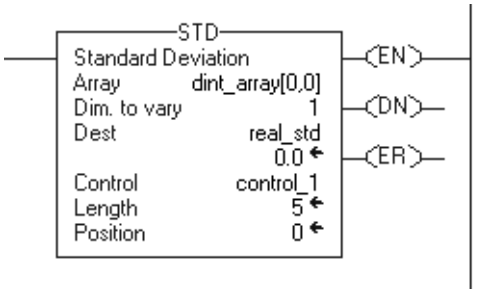
维度 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$AVE = \frac{20 + 19 + 18 + 17 + 16}{5} = \frac{90}{5} = 18$$

$$STD = \sqrt{\frac{\langle 20 - 18\rangle^2 + \langle 19 - 18\rangle^2 + \langle 18 - 18\rangle^2 + \langle 17 - 18\rangle^2 + \langle 16 - 18\rangle^2}{\langle 5 - 1\rangle}} = 1.581139$$

real_std = 1.581139

梯形图



结构化文本

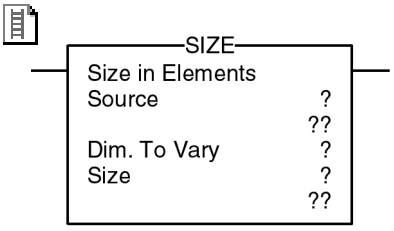
```
SIZE(dint_array,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
    sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
```

元素的大小 (Size In Elements, SIZE)

SIZE 指令获取数组的某一维度的大小。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT REAL 结构 string	数组标记	指令要操作的数组
Dimension to Vary	DINT	立即数 (0, 1, 2)	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	标记	存储指定数组维度的元素数目的标记

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 指令获取 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。

- 该指令获取数组的某一维度的大小。
- 该指令对以下数据类型进行操作：

  衆 t 数组
  衆 t 结构中的数组
  衆 t 作为较大数组的一部分的数组

算术状态标志： 不受影响

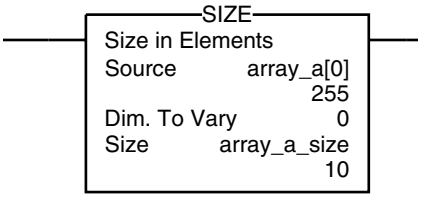
错误条件： 无。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令检查某一维度的大小。	该指令检查某一维度的大小。
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例 1： 获取 *array_a* 的维度 0 （第一维）的元素数目。将大小存储到 *array_a_size*。在此示例中， *array_a* 的维度 0 有 10 个元素。

梯形图

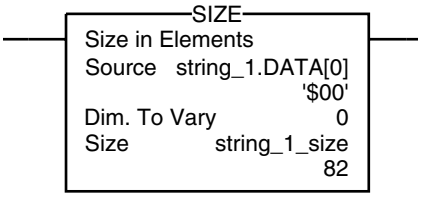


结构化文本

```
SIZE(array_a,0,array_a_size);
```

示例 2： 获取 *string_1* 的 DATA 成员 （它是字符串）的元素数目。将大小存储到 *string_1_size*。在此示例中， *string_1* 的 DATA 成员有 82 个元素。（字符串使用默认的 STRING 数据类型。）由于每个元素包含一个字符， *string_1* 最多可以包含 82 个字符。

梯形图

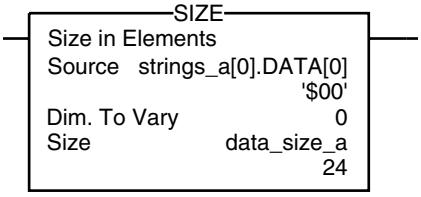


结构化文本

```
SIZE(string_1.DATA[0],0,string_1_size);
```

示例 3： *Strings_a* 是字符串结构组成的数组。SIZE 指令获取字符串结构的 DATA 成员中元素的数目，并将大小存储到 *data_size_a*。在此示例中， DATA 成员有 24 个元素。（该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);
```

说明：

数组（文件）/ 移位指令
（BSL、BSR、FFL、FFU、LFL、LFU）

简介

数组（文件）/ 移位指令用于修改数组中数据的位置。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
每次完成对位数组中一位的加载、移位和卸载操作。	BSL	梯形图	第 8 章 -2
	BSR	梯形图	第 8 章 -5
按相同的顺序加载和卸载值。	FFL	梯形图	第 8 章 -8
	FFU	梯形图	第 8 章 -14
按相反的顺序加载和卸载值。	LFL	梯形图	第 8 章 -20
	LFU	梯形图	第 8 章 -26

您可以混用数据类型，但可能导致发生精度损失和舍入误差。

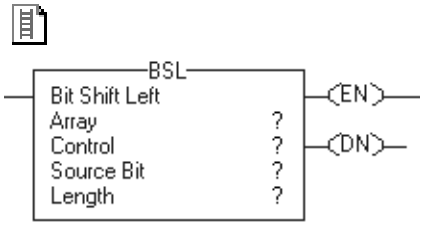
对于梯形图指令，**粗体**数据类型表示最佳数据类型。如果指令的所有操作数都使用同样的最佳数据类型（通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

位左移 (Bit Shift Left, BSL)

BSL 指令将 Array 中的指定位向左移动一个位置。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Array	DINT	数组标记	要修改的数组 指定元素组的第一个元素 **不要**在下标中使用 CONTROL.POS
CONTROL	CONTROL	标记	控制运算的结构
Source 位	BOOL	标记	要移动的位
Length	DINT	立即数	数组中要移动的位数

CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示 BSL 指令已启用。
.DN	BOOL	设置完成位以指示位已经向左移动了一个位置。
.UL	BOOL	卸载位是指令的输出。 .UL 位存储已移出位范围的位的状态。
.ER	BOOL	当 .LEN < 0 时将设置错误位。
.LEN	DINT	长度指定要移动的数组位的数目。

说明： 启用时，该指令将指定位的最高位卸载到 .UL 位，其余的位向左移动一个位置，并将 Source 位加载到数组的位 0。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

BSL 指令对连续内存进行操作。在某些情况下，该指令跨过数组将位移到标记的其他成员。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

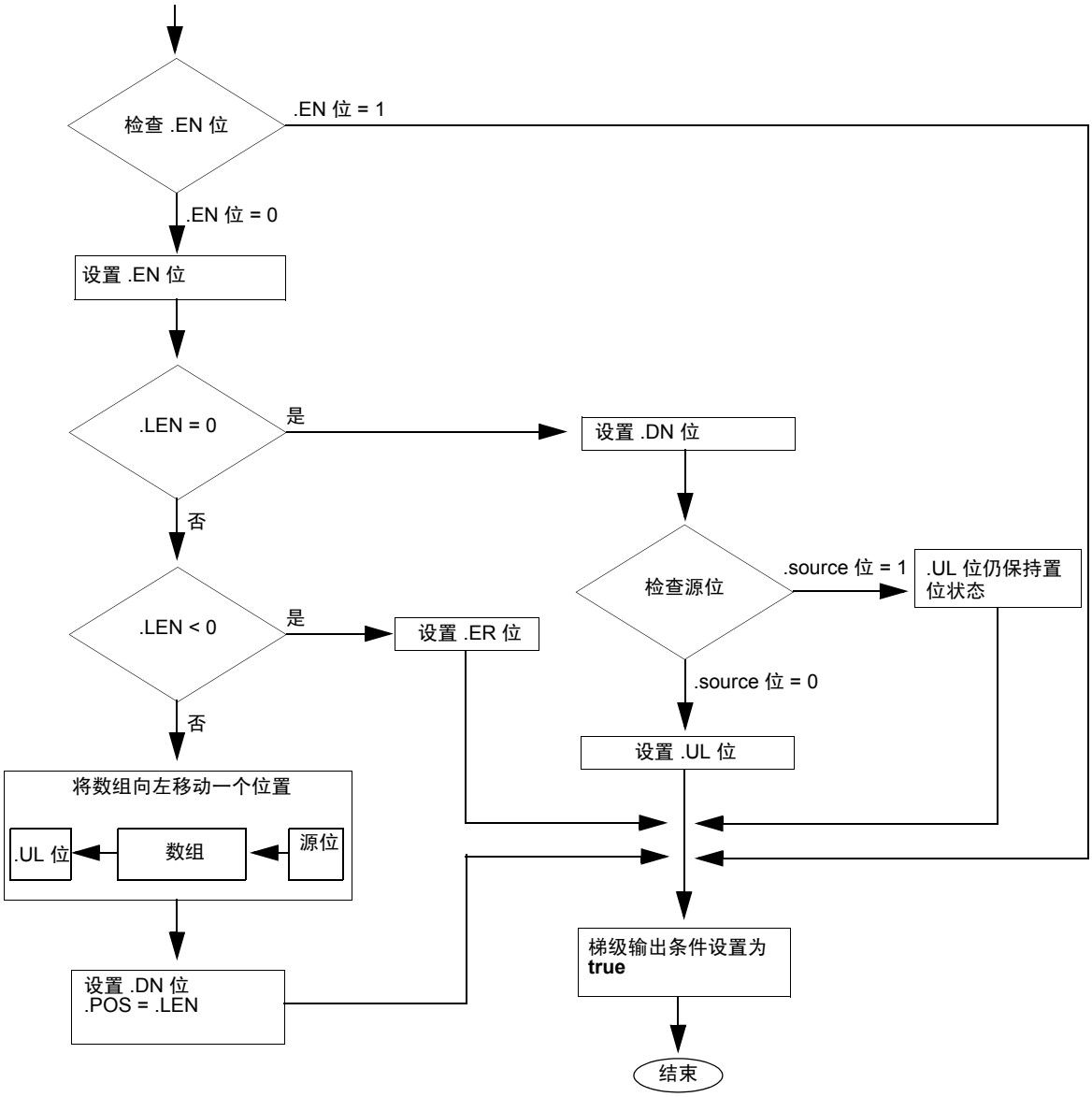
算术状态标志： 不受影响

错误条件： 无

执行：

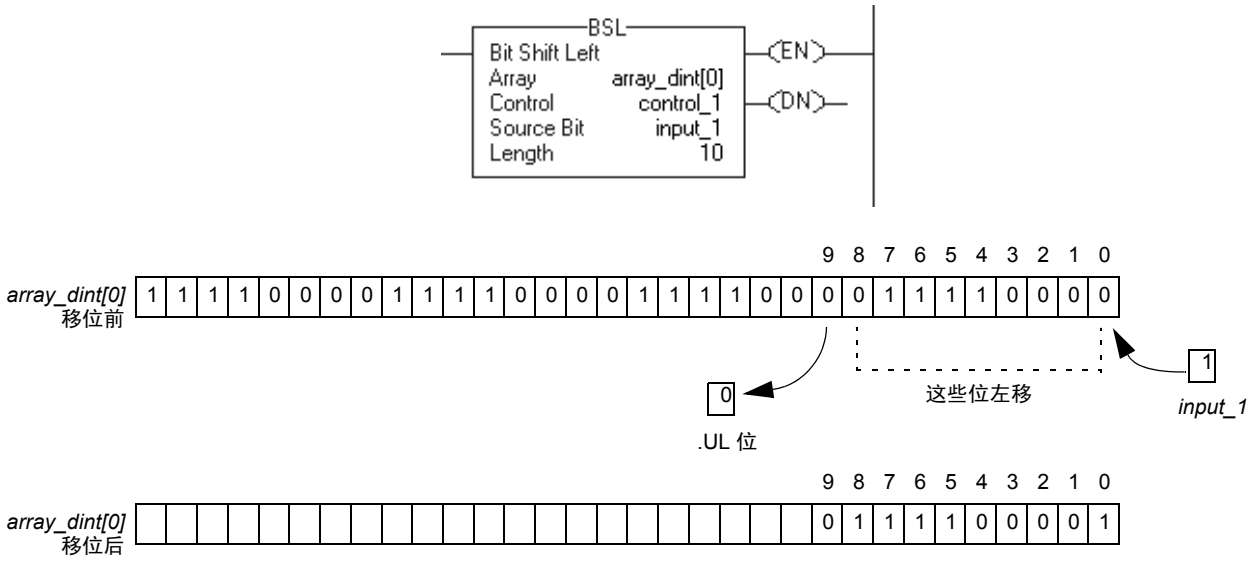
条件：	梯形图操作：
预扫描	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。 清除 .POS 值。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。 清除 .POS 值。 梯级输出条件设置为 false。

梯级输入条件为 true

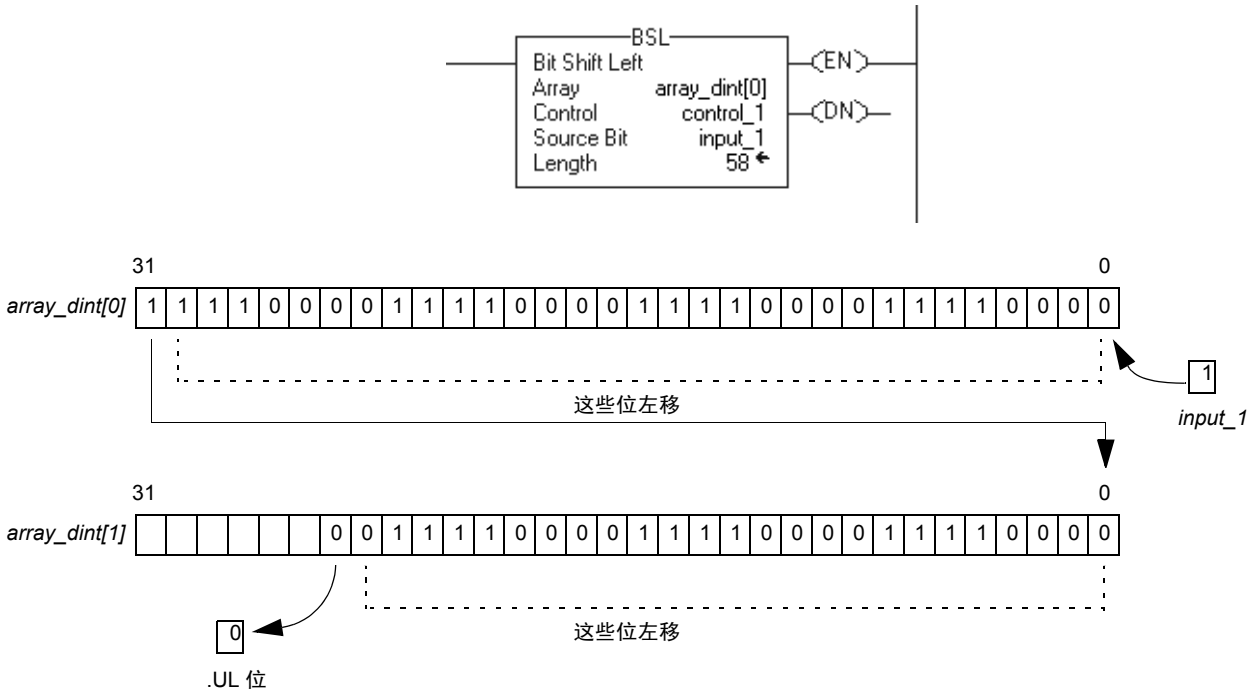


后扫描	梯级输出条件设置为 false。

示例 1： 启用时， BSL 指令起始于 *array_dint[0]* 中的 0 位。指令将 *array_dint[0].9* 卸载到 .UL 位，同时移动其余的位，并将 *input_1* 加载到 *array_dint[0].0*。其余位 (10-31) 中的值无效。



示例 2： 启用时， BSL 指令起始于 *array_dint[0]* 中的 0 位。指令将 *array_dint[1].25* 卸载到 .UL 位，同时移动其余的位，并将 *input_1* 加载到 *array_dint[0].0*。其余位 （*array_dint[1]* 中的位 31-26） 中的值无效。请注意 *array_dint[0].31* 是如何跨字移位到 *array_dint[1].0* 的。

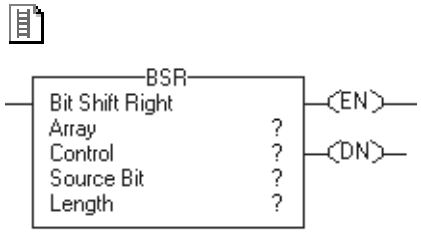


位右移 (Bit Shift Right, BSR)

BSR 指令将 Array 中的指定位向右移动一个位置。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Array	DINT	数组标记	要修改的数组 指定开始移动的元素 **不要**在下标中使用 CONTROL.POS
CONTROL	CONTROL	标记	控制运算的结构
Source 位	BOOL	标记	要移动的位
Length	DINT	立即数	数组中要移动的位数

CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示 BSR 指令已启用。
.DN	BOOL	设置完成位以指示位已经向右移动了一个位置。
.UL	BOOL	卸载位是指令的输出。 .UL 位存储已移出位范围的位的状态。
.ER	BOOL	当 .LEN < 0 时将设置错误位。
.LEN	DINT	长度指定要移动的数组位的数目。

说明： 启用时，指令将 Array 中位 0 的值卸载到 .UL 位，将其余的位向右移动一个位置，并且将 Source 位加载到指定位的最高位。

重要事项

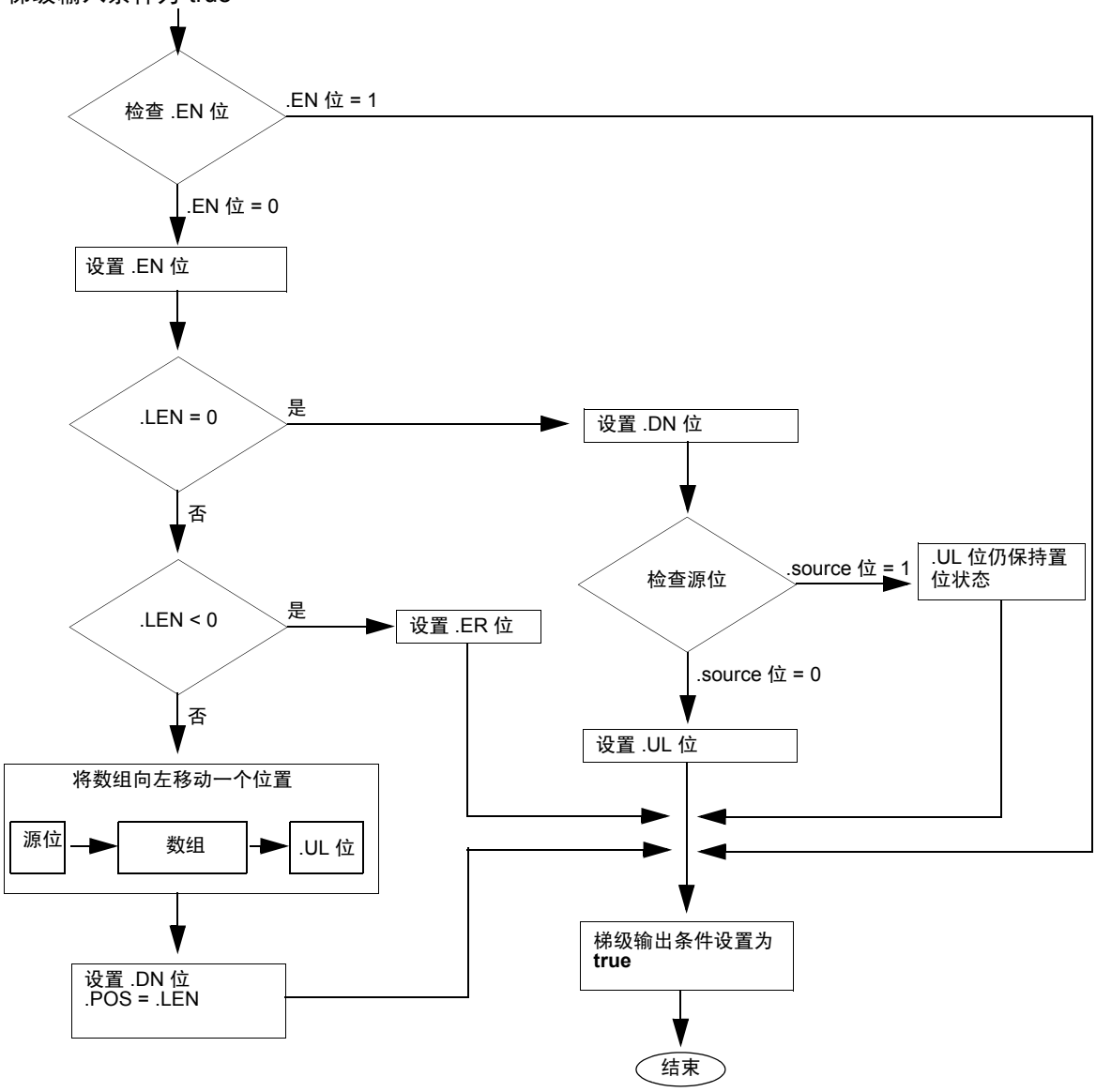
您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

BSR 指令对连续内存进行操作。某些情况下，指令会更改标记的其他成员中的位。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

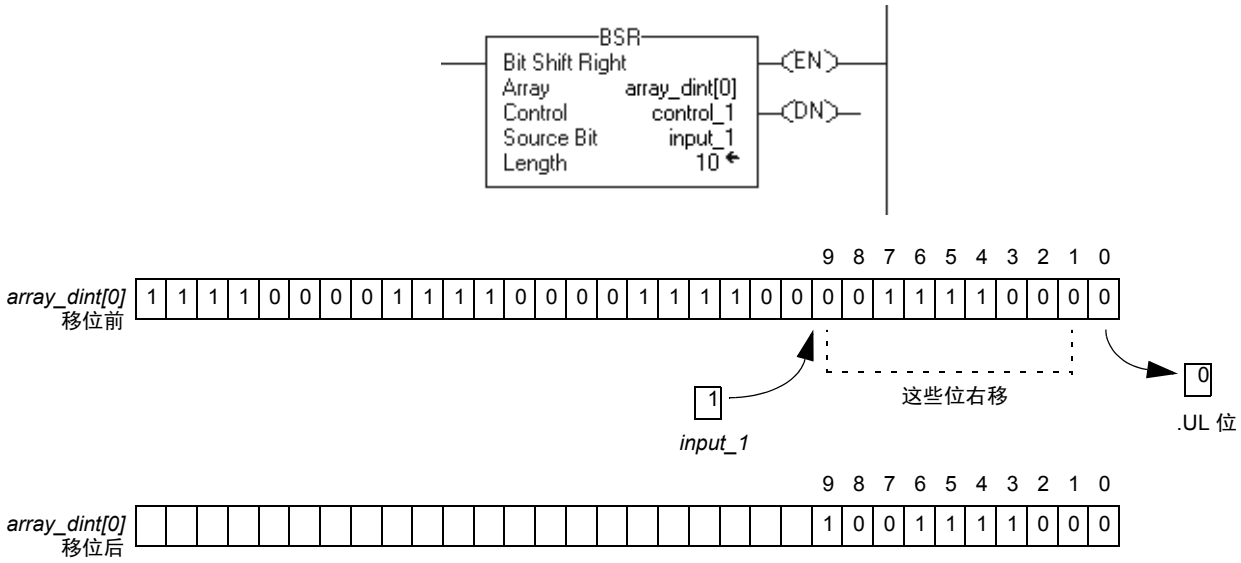
算术状态标志： 不受影响

错误条件： 无

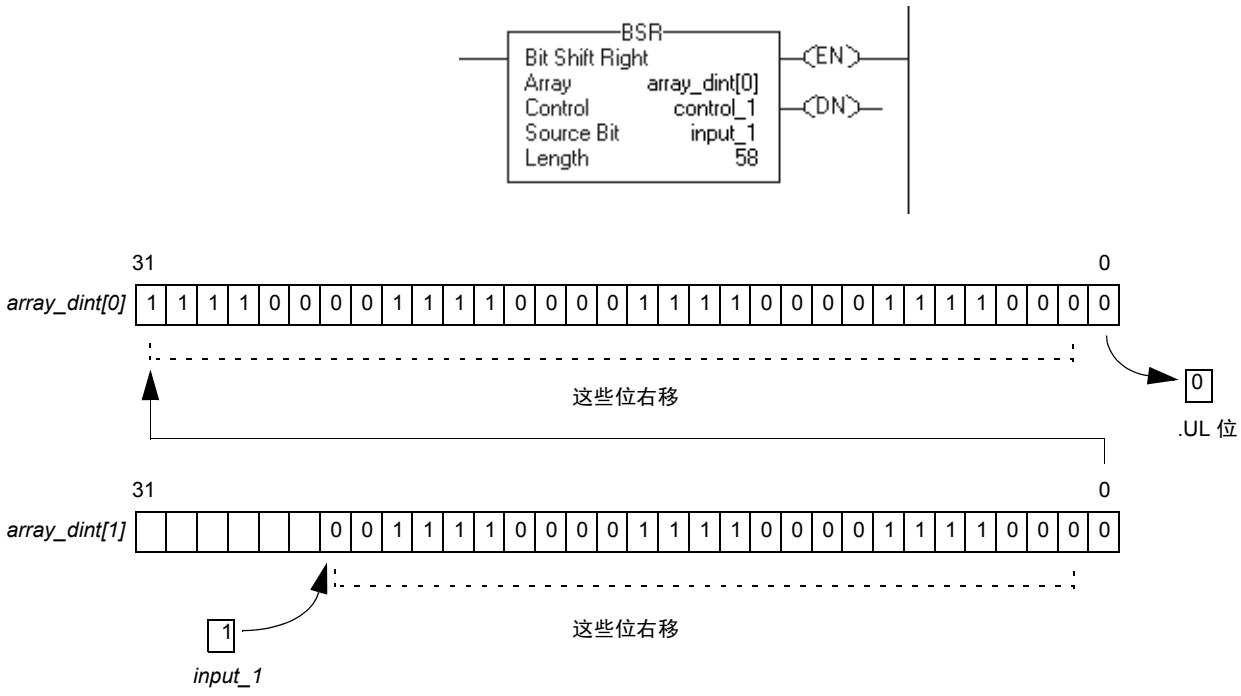
执行：

条件：	梯形图操作：
预扫描	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。 清除 .POS 值。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .EN 位。 清除 .DN 位。 清除 .ER 位。 清除 .POS 值。 梯级输出条件设置为 false。
梯级输入条件为 true	
后扫描	梯级输出条件设置为 false。

示例 1： 启用时， BSR 指令从 *array_dint[0]* 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，并将 *input_1* 加载到 *array_dint[0].9*。其余位 (10-31) 中的值无效。



示例 2： 启用时， BSR 指令从 *array_dint[1]* 的位 25 开始执行。指令将 *array_dint[0].0* 卸载到 .UL 位，右移其余的位，并将 *input_1* 加载到 *array_dint[1].25*。其余位 （*dint_array[1]* 中的位 31-26）中的值无效。请注意 *array_dint[1].0* 是如何跨字移位到 *array_dint[0].31* 的。

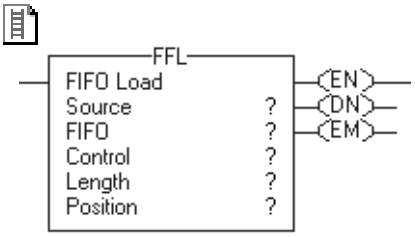


FIFO 加载 (FIFO Load, FFL)

FFL 指令将 Source 值复制到 FIFO。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	**SINT** **INT** **DINT** **REAL** **string** **结构**	立即数 标记	要存入 FIFO 中的数据
	Source 转换为数组标记的数据类型。较小的整数通过符号扩展转换为更大的整数。		
FIFO	**SINT** **INT** **DINT** **REAL** **string** **结构**	数组标记	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	标记	控制运算的结构 通常与相关的 FFU 指令 使用相同的 CONTROL
Length	DINT	立即数	FIFO 一次最多可容纳的元素数
Position	DINT	立即数	在 FIFO 中指令加载数据的下一个位置初始值通常为 0

如果您使用用户自定义的结构作为 Source 或 FIFO 操作数的数据类型，请为这两个操作数使用相同的结构。

CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示 FFL 指令已启用。
.DN	BOOL	设置完成位以指示 FIFO 已满 (.POS = .LEN)。.DN 位禁止加载 FIFO，直到 .POS < .LEN 为止。
.EM	BOOL	空位指示 FIFO 是空的。如果 .LEN ≤ 0 或 .POS < 0，则同时设置 .EM 位和 .DN 位。
.LEN	DINT	长度指定 FIFO 一次最多可以容纳的元素数。
.POS	DINT	位置标识指令将要加载的下一个值在 FIFO 中的位置。

说明： 使用 FFL 指令和 FFU 指令可以按先进 / 先出的顺序存储和取回数据。当成对使用时， FFL 和 FFU 指令建立一个异步移位寄存器。

通常， Source 和 FIFO 属于相同的数据类型。

启用时，FFL 指令将 Source 值加载到 FIFO 中由 .POS 值标识的位置。每次启用该指令时，它加载一个值，直到 FIFO 满为止。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

FFL 指令对连续内存进行操作。有些情况下，该指令将超出数组容量的数据加载到标记的其他成员中。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

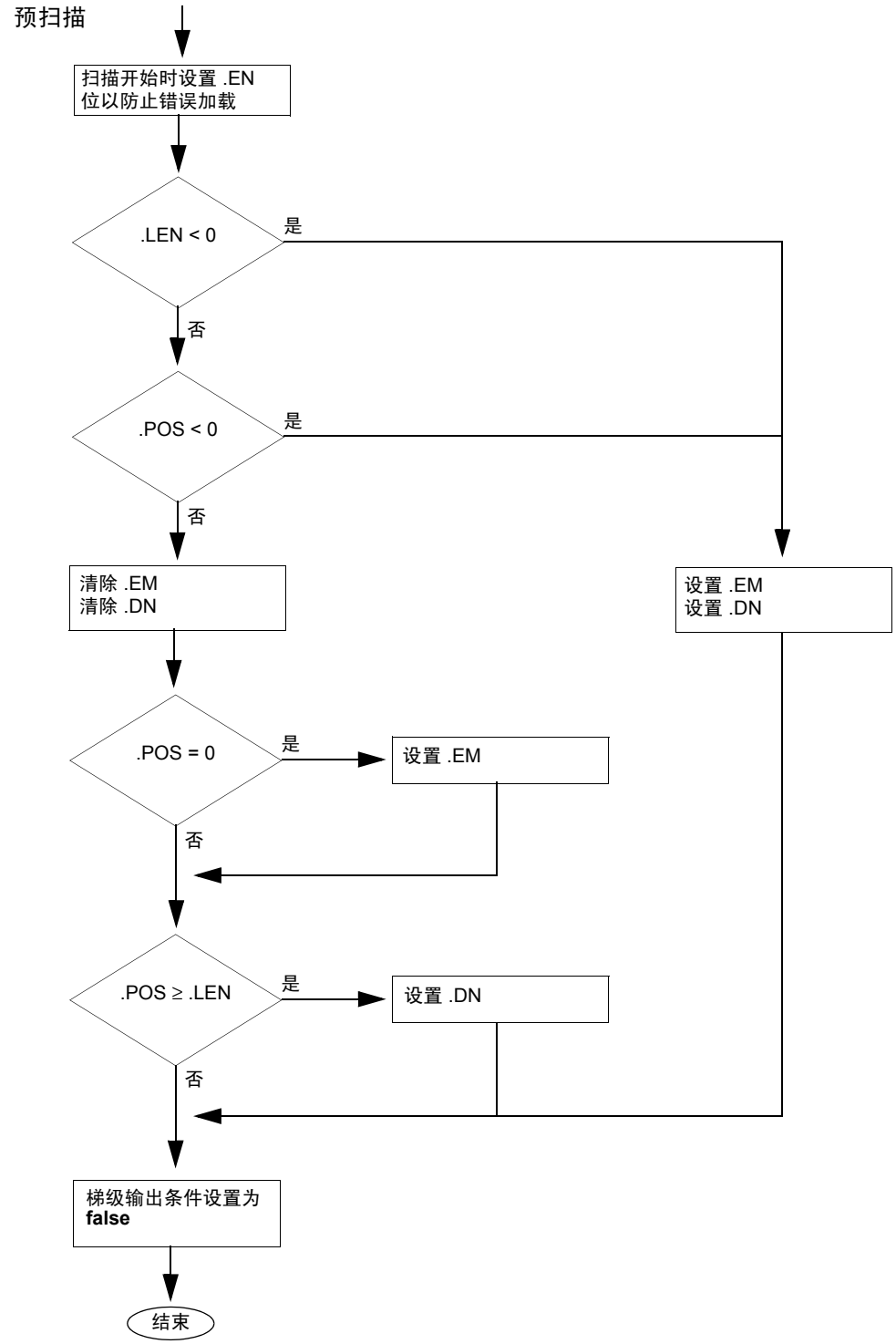
算术状态标志： 不受影响

错误条件：

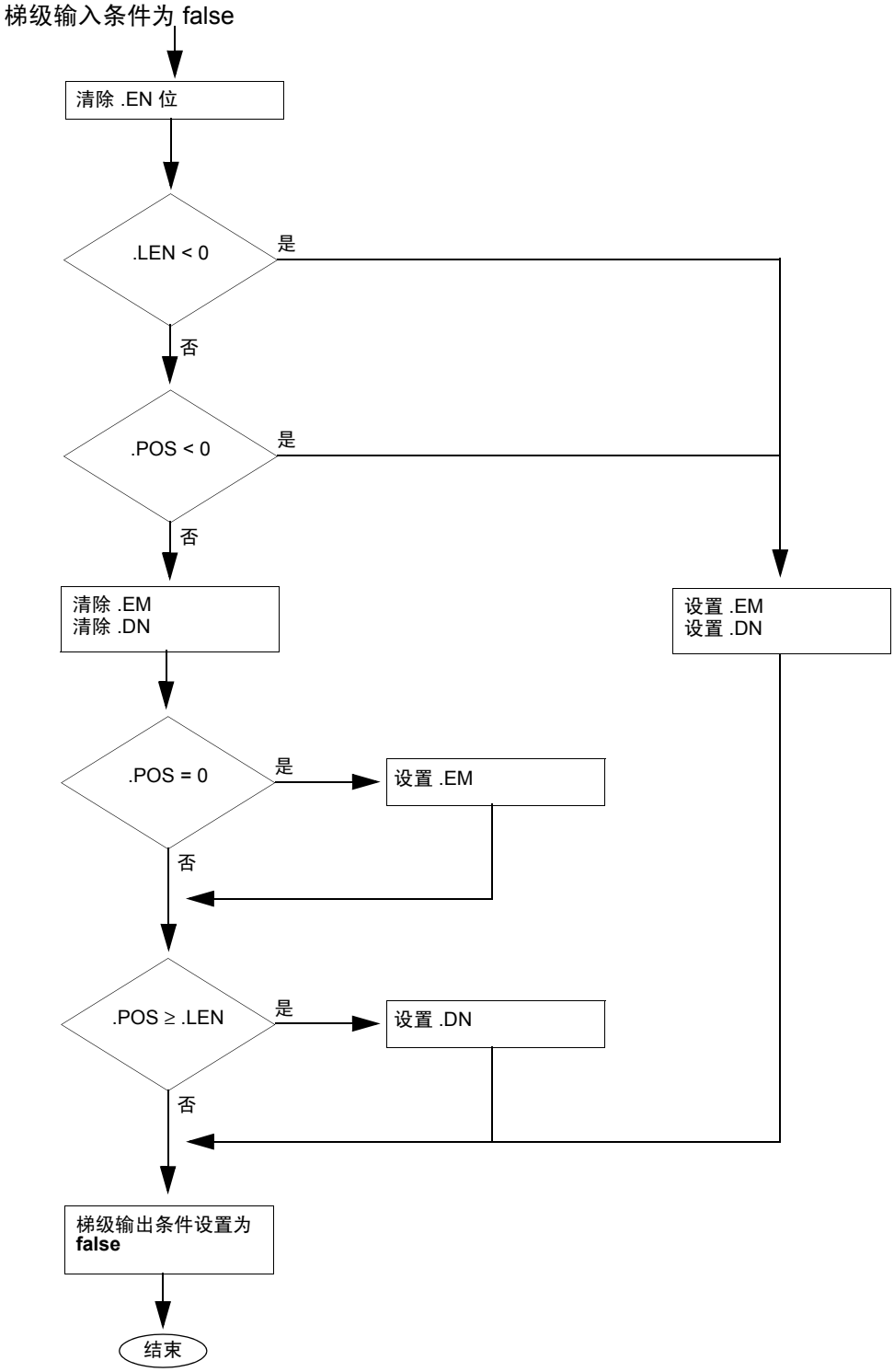
出现严重错误的条件：	错误类型：	错误代码：
（起始元素 + .POS） > FIFO 数组大小	4	20

执行：

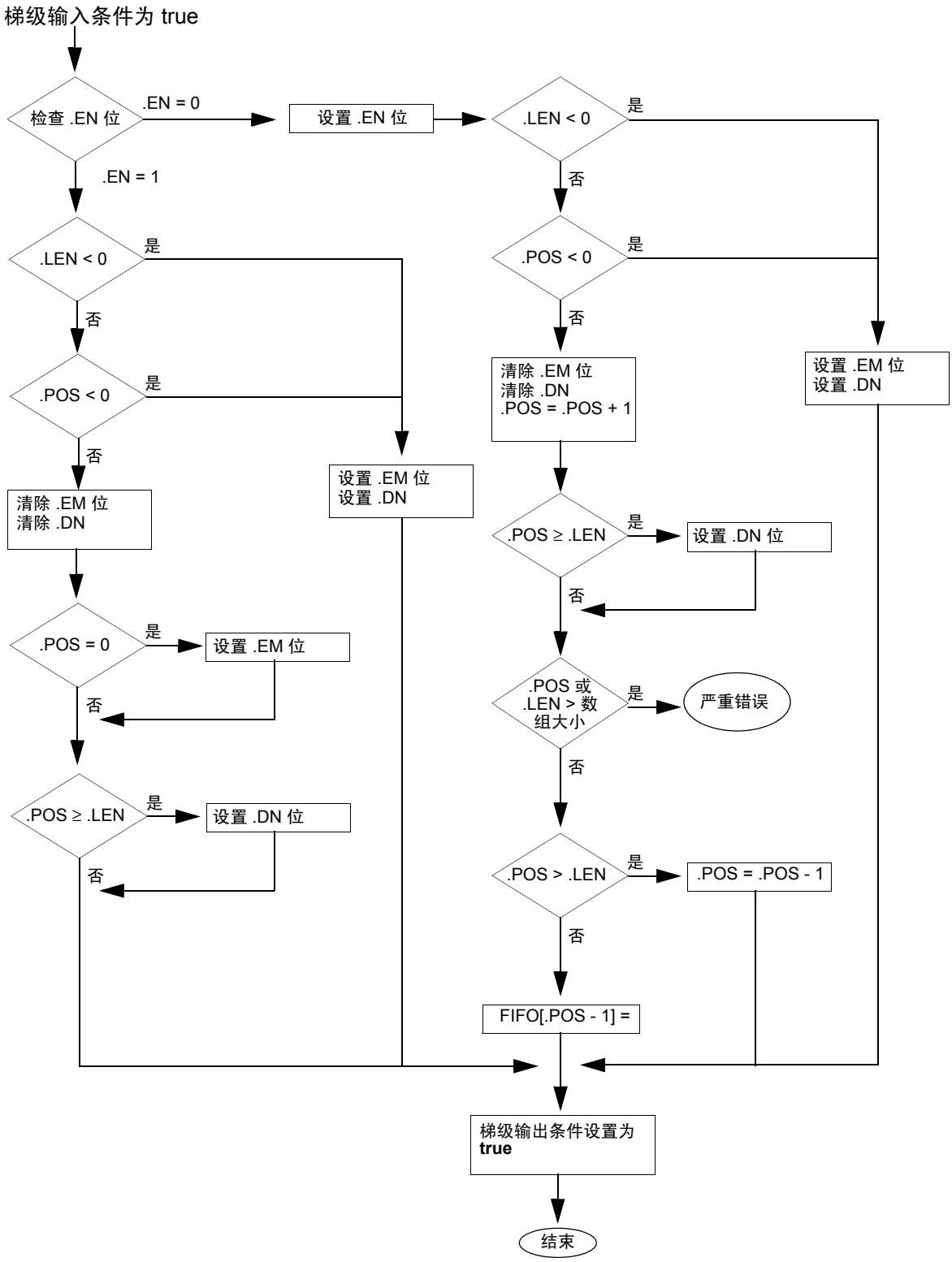
条件：	梯形图操作：


预扫描
扫描开始时设置 .EN 位以防止错误加载
.LEN < 0
是
否
.POS < 0
是
否
清除 .EM
清除 .DN
设置 .EM
设置 .DN
.POS = 0
是
设置 .EM
否
.POS ≥ .LEN
是
设置 .DN
否
梯级输出条件设置为 false
结束

条件：	梯形图操作：

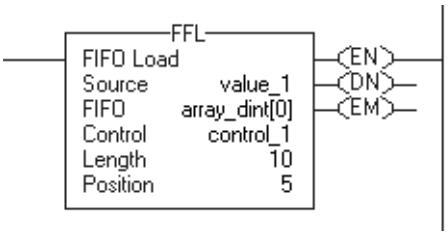

梯级输入条件为 false
清除 .EN 位
.LEN < 0
是
否
.POS < 0
是
否
清除 .EM
清除 .DN
设置 .EM
设置 .DN
.POS = 0
是
设置 .EM
否
.POS ≥ .LEN
是
设置 .DN
否
梯级输出条件设置为 false
结束

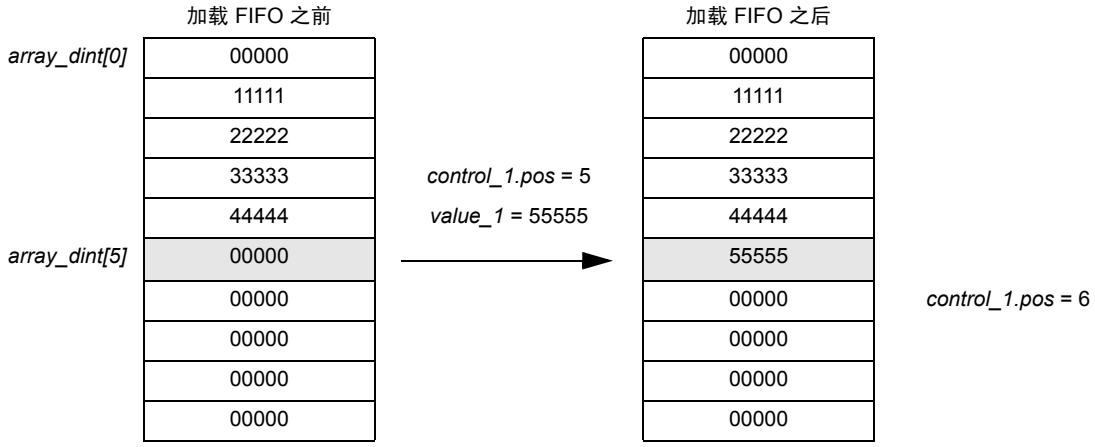
条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， FFL 指令将 *value_1* 加载到 FIFO 中的下一个位置，即本例中的 *array_dint[5]*。



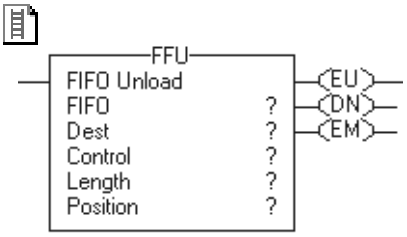


FIFO 卸载 (FIFO Unload, FFU)

FFU 指令卸载 FIFO 位置 0 （第一个位置）的值，并将该值存储到 Destination。 FIFO 中的其余数据向下移动一个位置。

操作数：

梯形图



操作数：	类型：	格式：	说明：
FIFO	**SINT** **INT** **DINT** **REAL** **string** **结构**	数组标记	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** **string** **结构**	标记	从 FIFO 退出的值
	Destination 值转换为 Destination 标记的数据类型。较小的整数通过符号扩展转换为更大的整数。		
CONTROL	CONTROL	标记	控制运算的结构 通常与相关的 FFL 指令 使用相同的 CONTROL
Length	DINT	立即数	FIFO 一次最多可容纳的元素数
Position	DINT	立即数	在 FIFO 中指令卸载数据的下一个位置初始值通常为 0

如果您使用用户自定义的结构作为 FIFO 或 Destination 操作数的数据类型，请为这两个操作数使用相同的结构。

CONTROL 结构

助记符：	数据类型：	说明：
.EU	BOOL	启用卸载位指示 FFU 指令已启用。在程序扫描开始时设置 .EU 位以预设一个错误加载。
.DN	BOOL	设置完成位以指示 FIFO 已满 (.POS = .LEN)。
.EM	BOOL	空位指示 FIFO 是空的。如果 .LEN ≤ 0 或 .POS < 0，则 .EM 位和 .DN 位将处于置位状态。
.LEN	DINT	长度指定 FIFO 中元素的最大数目。
.POS	DINT	位置标识加载到 FIFO 中的数据的末尾。

说明： 使用 FFU 指令和 FFL 指令可以按先进 / 先出的顺序存储和取回数据。

启用时， FFU 指令从 FIFO 的第一个元素卸载数据，并将该值放置到 Destination。每次启用该指令时，它卸载一个值，直到 FIFO 空为止。如果 FIFO 为空， FFU 向 Destination 返回 0。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

FFU 指令对连续内存进行操作。该指令有时从标记的其他成员卸载数据。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

算术状态标志： 不受影响

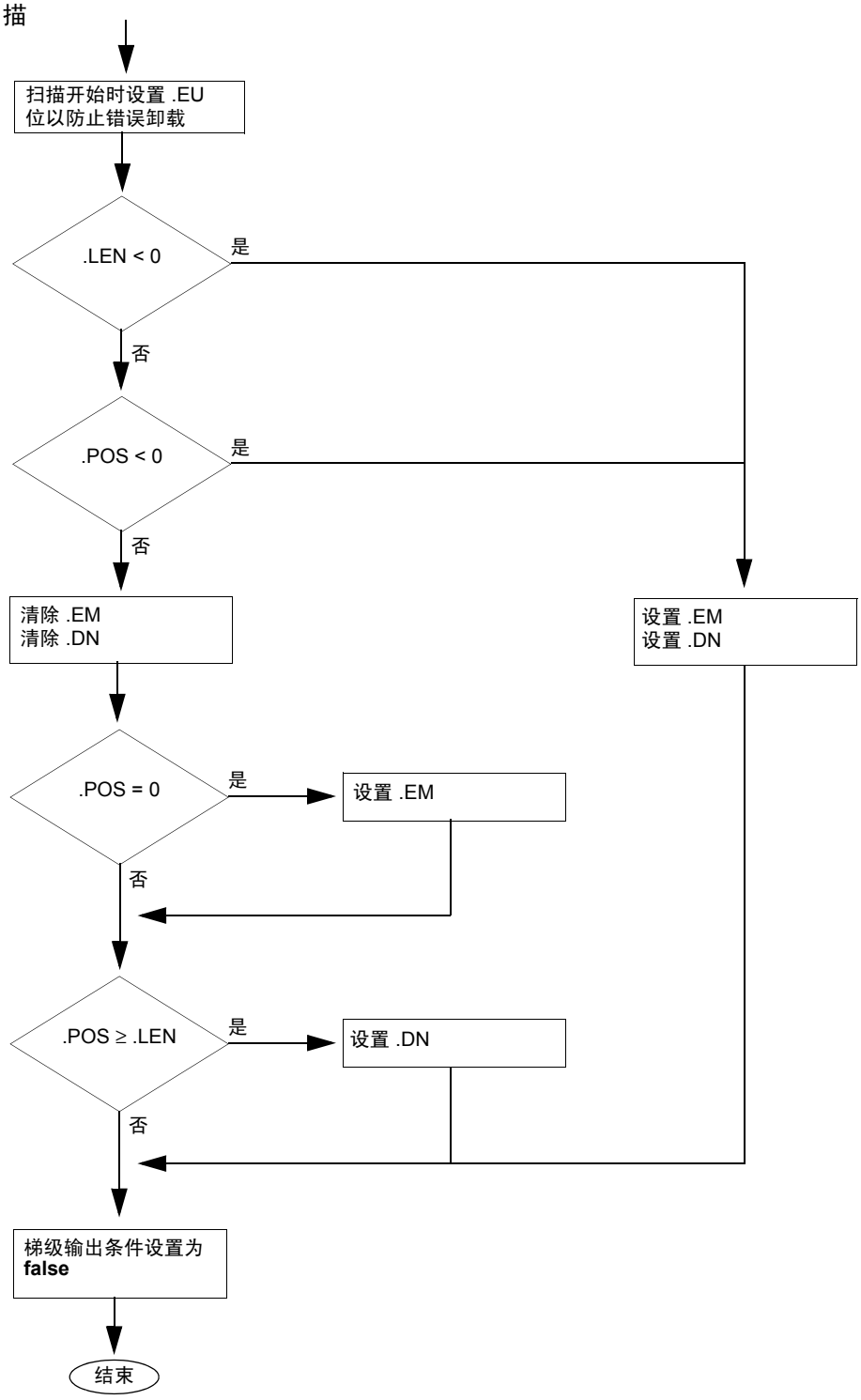
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
Length > FIFO 数组大小	4	20

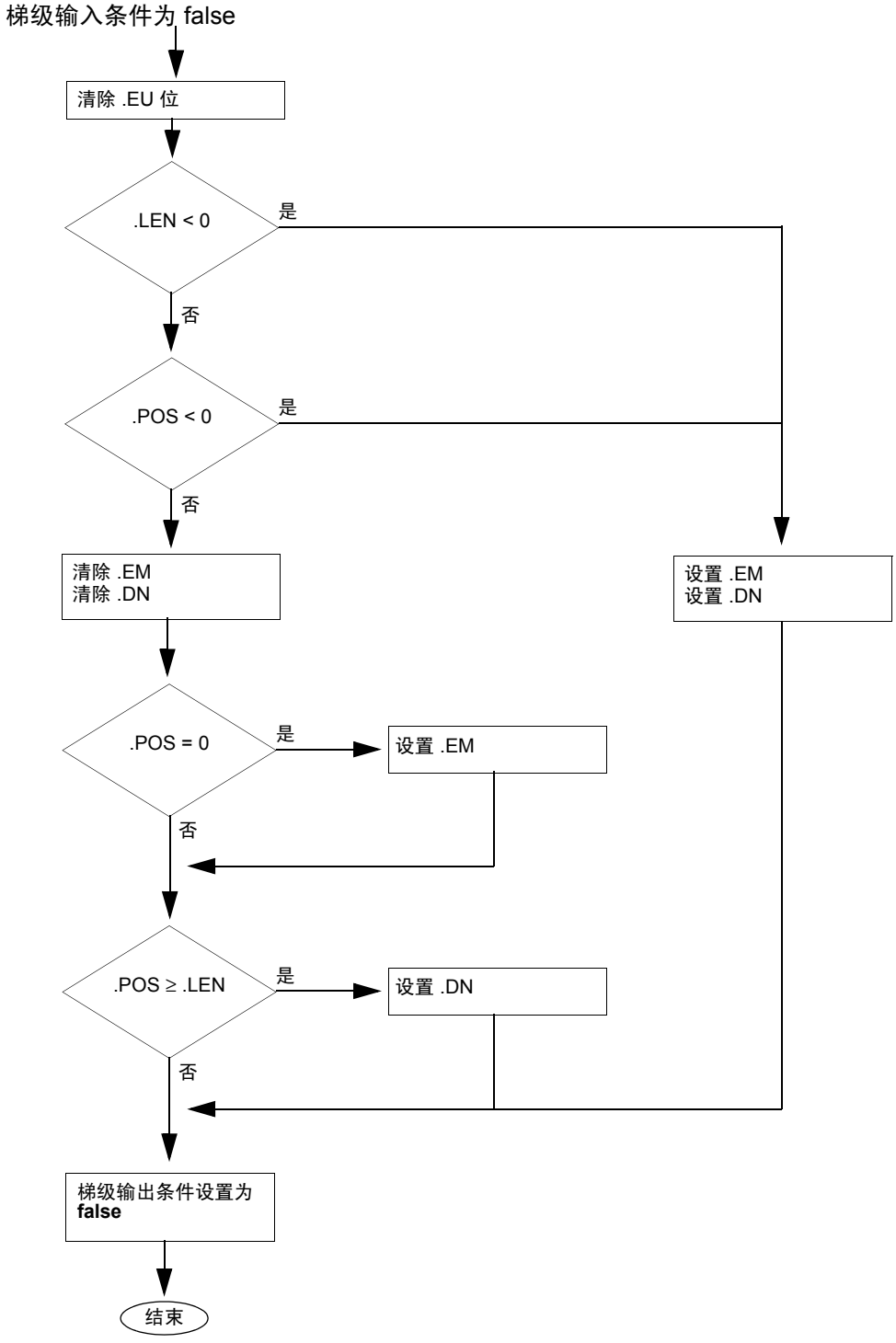
执行：

条件：	梯形图操作：

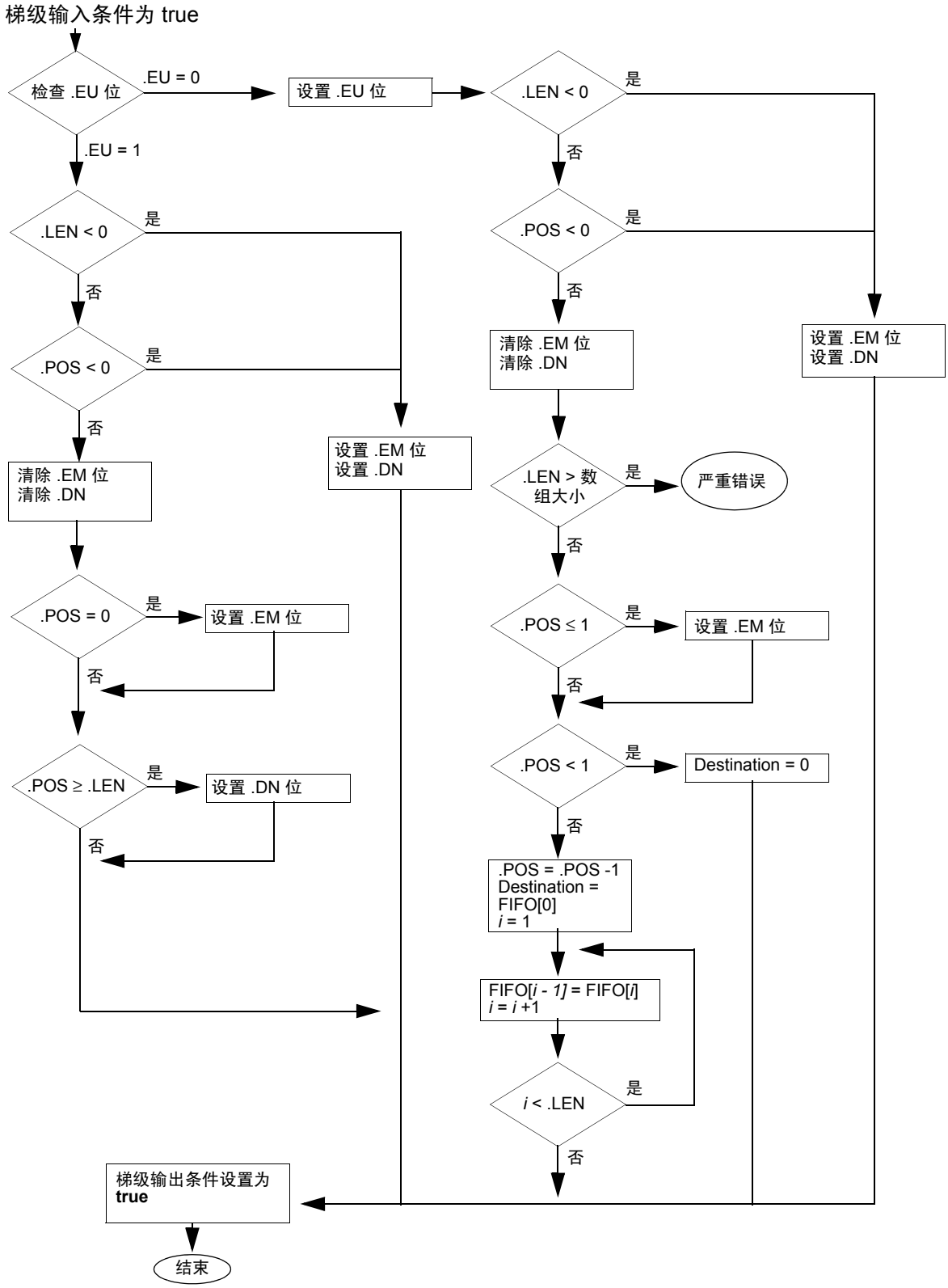
预扫描



条件：	梯形图操作：

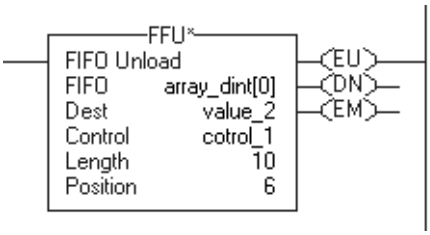

梯级输入条件为 false
清除 .EU 位
.LEN < 0
是
否
.POS < 0
是
否
清除 .EM
清除 .DN
设置 .EM
设置 .DN
.POS = 0
是
设置 .EM
否
.POS ≥ .LEN
是
设置 .DN
否
梯级输出条件设置为
false
结束

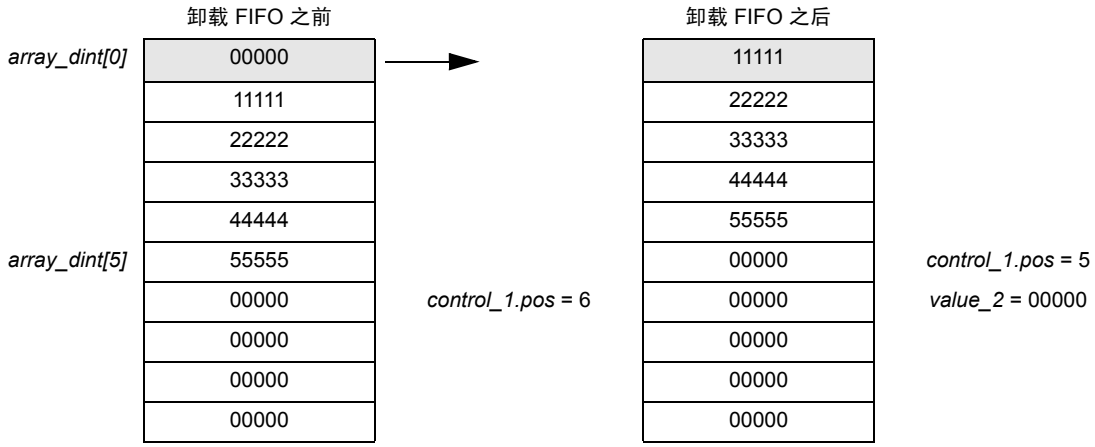
条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， FFU 指令将 *array_dint[0]* 卸载到 *value_2* 并移动 *array_dint* 中的其余元素。



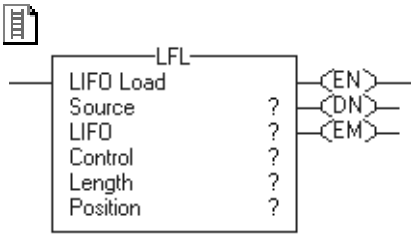


LIFO 加载 (LIFO Load, LFL)

LFL 指令将 Source 值复制到 LIFO。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	**SINT** **INT** **DINT** **REAL** **string** **结构**	立即数 标记	要存储到 LIFO 中的数据
	Source 转换为数组标记的数据类型。较小的整数通过符号扩展转换为更大的整数。		
LIFO	**SINT** **INT** **DINT** **REAL** **string** **结构**	数组标记	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	标记	控制运算的结构 通常与相关的 LFU 指令 使用相同的 CONTROL
Length	DINT	立即数	LIFO 一次最多可容纳的元素数
Position	DINT	立即数	在 LIFO 中指令加载数据的下一个位置初始值通常为 0

如果您使用用户自定义的结构作为 Source 或 LIFO 操作数的数据类型，请为这两个操作数使用相同的结构。

CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示 LFL 指令已启用。
.DN	BOOL	设置完成位以指示 LIFO 已满 (.POS = .LEN)。 .DN 位禁止加载 LIFO，直到 .POS < .LEN。
.EM	BOOL	空位指示 LIFO 是空的。如果 .LEN ≤ 0 或 .POS < 0，则同时设置 .EM 位和 .DN 位。
.LEN	DINT	长度指定 LIFO 一次最多可容纳的元素数。
.POS	DINT	位置标识指令将要加载的下一个值在 LIFO 中的位置。

说明： 使用 LFL 指令和 LFU 指令可以按后进 / 先出的顺序存储和取回数据。当成对使用时， LFL 和 LFU 指令建立一个异步移位寄存器。

通常， Source 和 LIFO 属于相同的数据类型。

启用时，LFL 指令将 Source 值加载到 LIFO 中由 .POS 值标识的位置。每次启用该指令时，它加载一个值，直到 LIFO 满为止。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

LFL 指令对连续内存进行操作。有些情况下，该指令将超出数组容量的数据加载到标记的其他成员中。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

算术状态标志： 不受影响

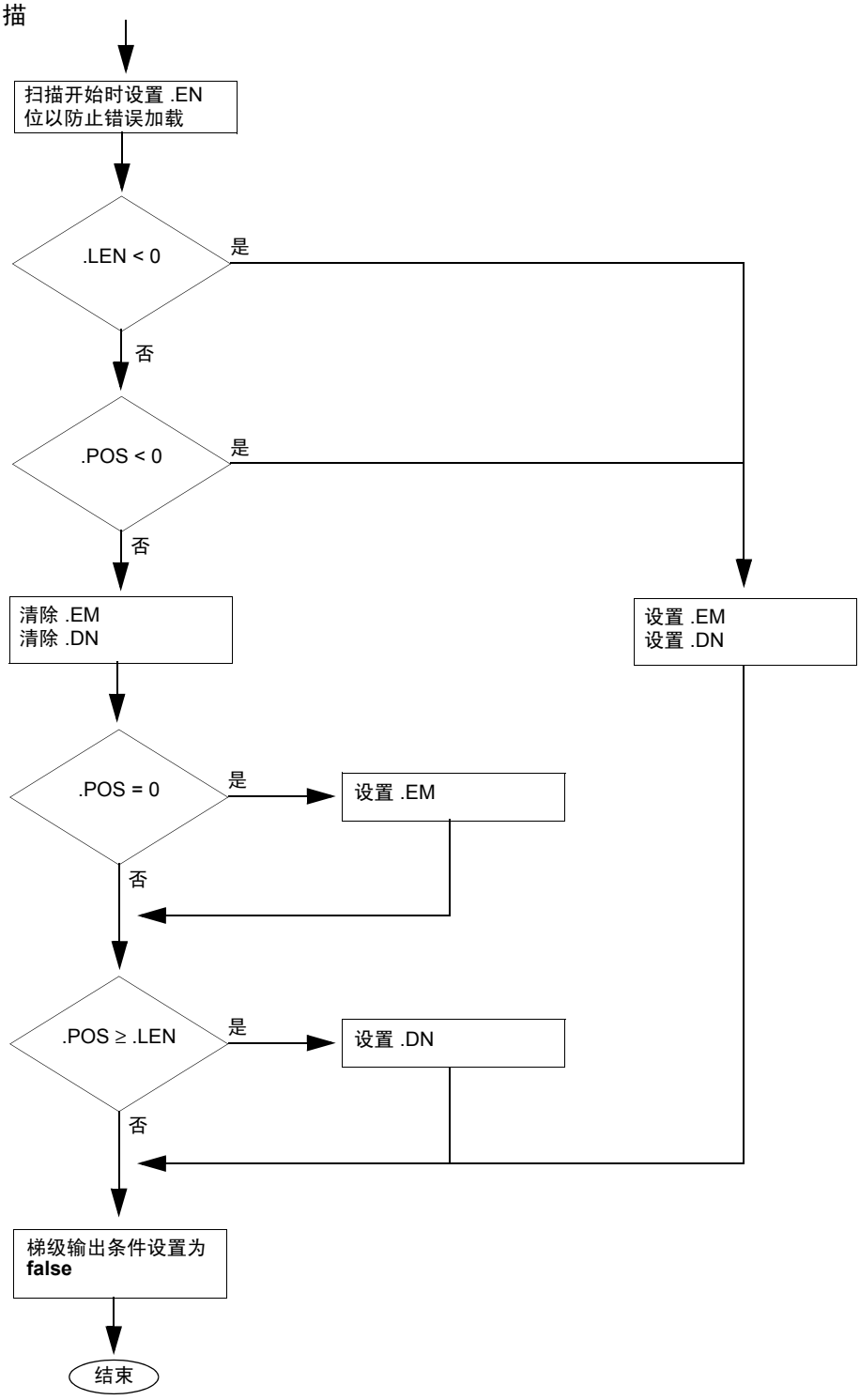
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
（起始元素 + .POS） > LIFO 数组大小	4	20

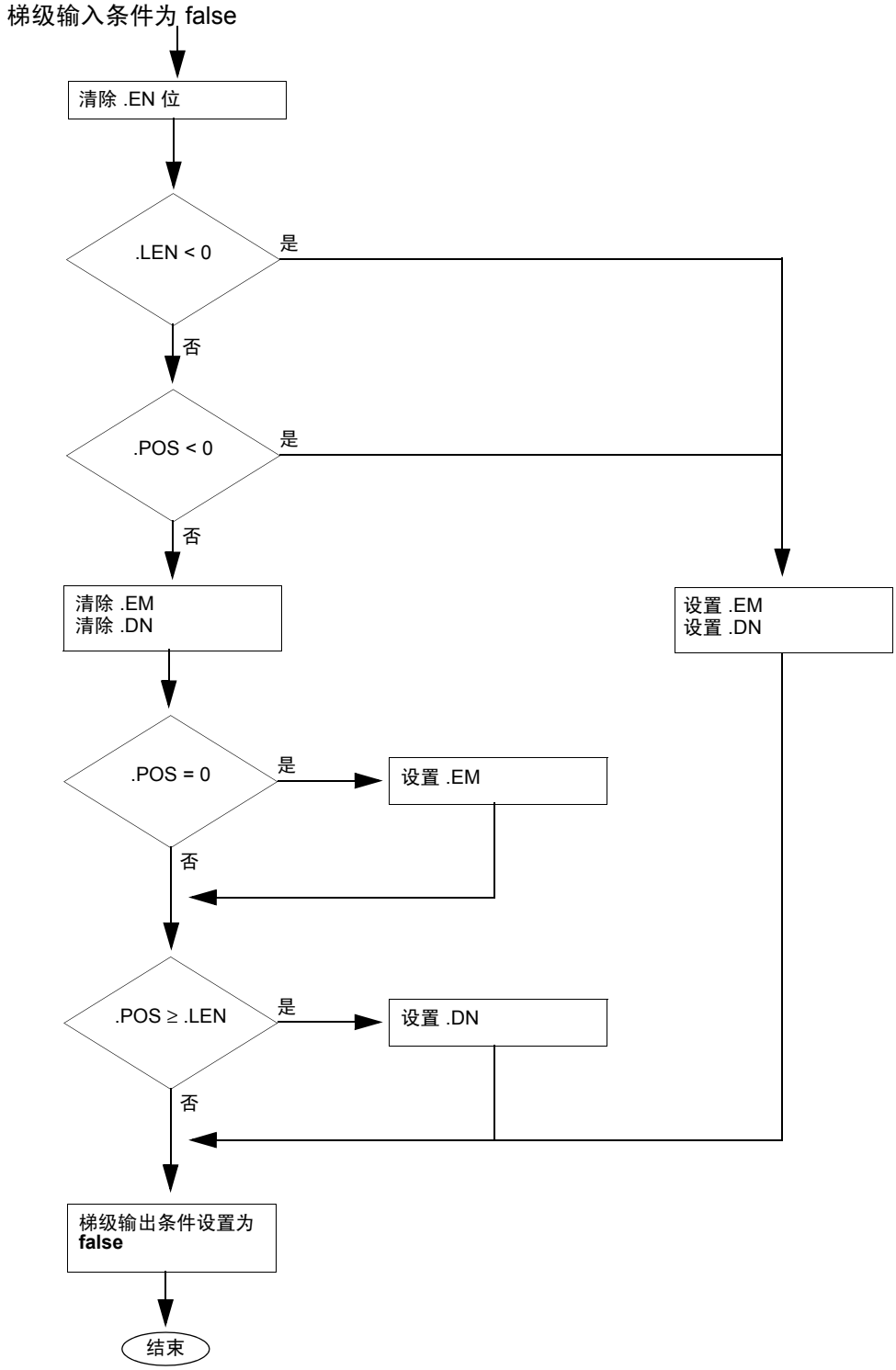
执行：

条件：	梯形图操作：

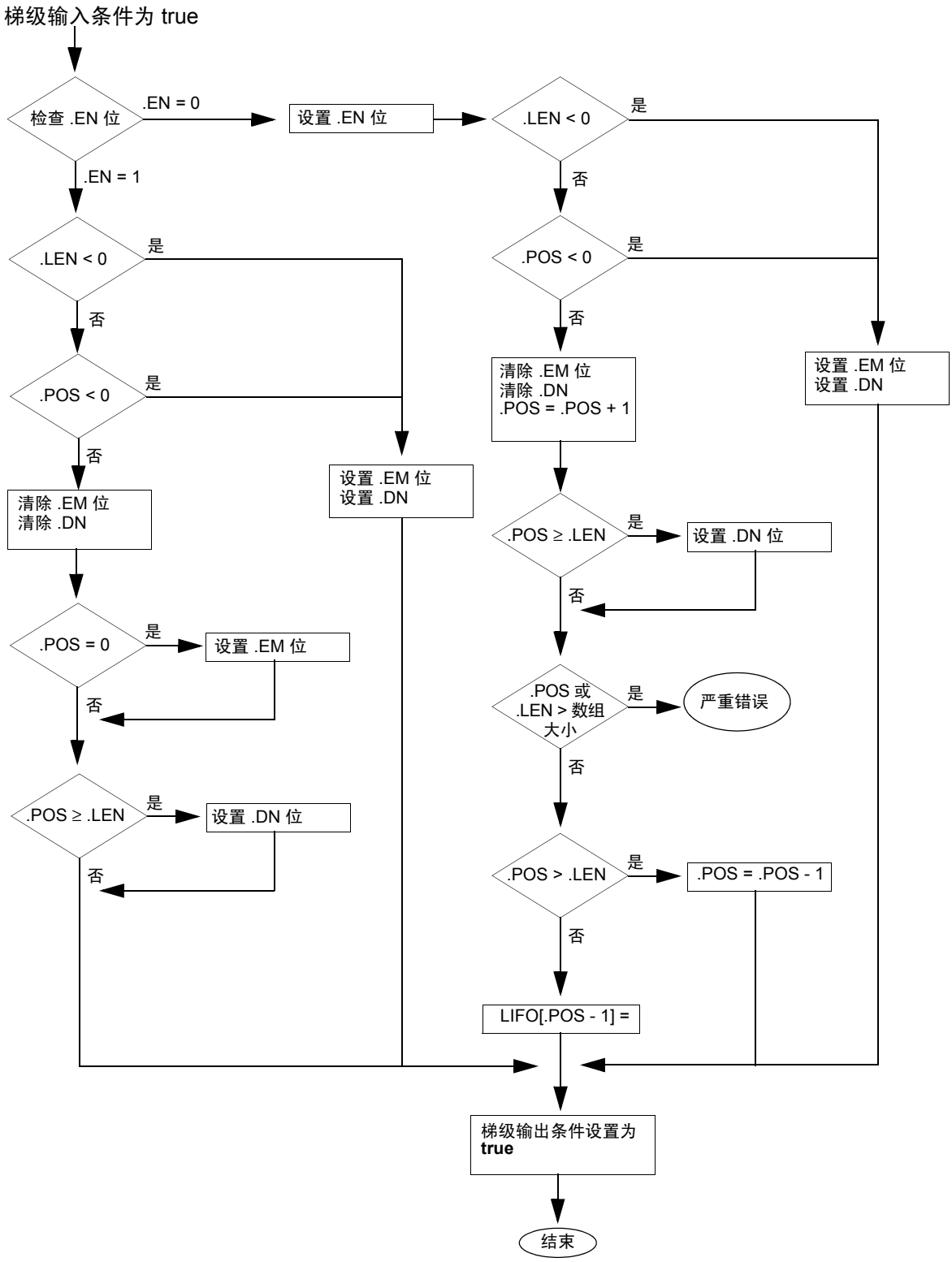
预扫描



条件：	梯形图操作：

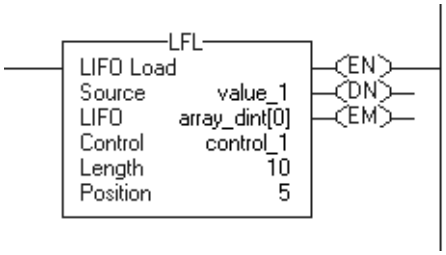

梯级输入条件为 false
清除 .EN 位
.LEN < 0
是
否
.POS < 0
是
否
清除 .EM
清除 .DN
设置 .EM
设置 .DN
.POS = 0
是
设置 .EM
否
.POS ≥ .LEN
是
设置 .DN
否
梯级输出条件设置为
false
结束

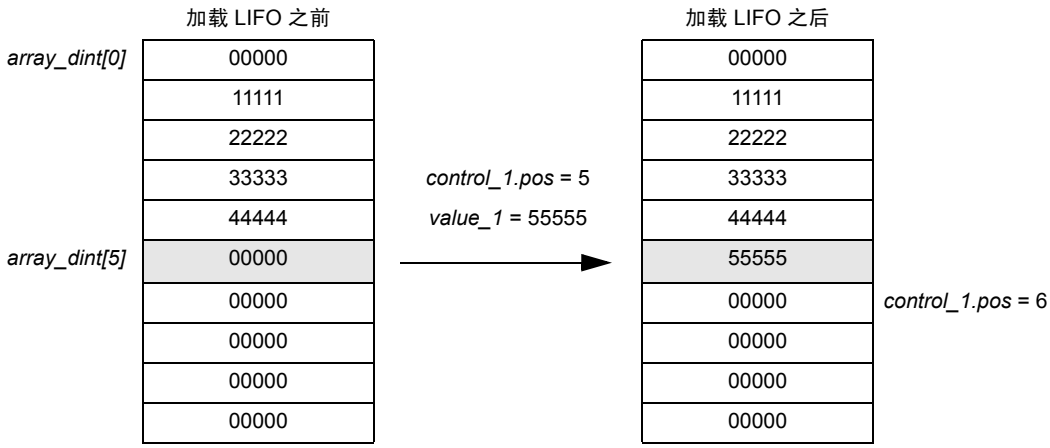
条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， LFL 指令将 *value_1* 加载到 LIFO 中的下一个位置，即本例中的 *array_dint[5]*。



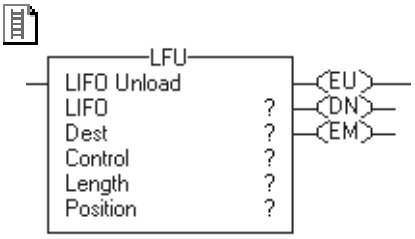


LIFO 卸载 (LIFO Unload, LFU)

LFU 指令卸载 LIFO 中 .POS 处的值，并将 0 存储到该位置。

操作数：

梯形图



操作数：	类型：	格式：	说明：
LIFO	**SINT** **INT** **DINT** **REAL** **string** **结构**	数组标记	要修改的 LIFO 指定 LIFO 的第一个元素 **不要** 在下标中使用 CONTROL.POS
Destination	**SINT** **INT** **DINT** **REAL** **string** **结构**	标记	从 LIFO 退出的值
	Destination 值转换为 Destination 标记的数据类型。较小的整数通过符号扩展转换为更大的整数。		
CONTROL	CONTROL	标记	控制运算的结构 通常与相关的 LFL 指令使用相同的 CONTROL
Length	DINT	立即数	LIFO 一次最多可容纳的元素数
Position	DINT	立即数	在 LIFO 中指令卸载数据的下一个位置初始值通常为 0

如果您使用用户自定义的结构作为 LIFO 或 Destination 操作数的数据类型，请为这两个操作数使用相同的结构。

CONTROL 结构

助记符：	数据类型：	说明：
.EU	BOOL	启用卸载位指示 LFU 指令已启用。在程序扫描开始时设置 .EU 位以预设一个错误加载。
.DN	BOOL	设置完成位以指示 LIFO 已满 (.POS = .LEN)。
.EM	BOOL	空位指示 LIFO 是空的。如果 .LEN ≤ 0 或 .POS < 0，则同时设置 .EM 位和 .DN 位。
.LEN	DINT	长度指定 LIFO 一次最多可容纳的元素数。
.POS	DINT	位置标识加载到 LIFO 中的数据的末尾。

说明： 使用 LFU 指令和 LFL 指令可以按后进 / 先出的顺序存储和取回数据。

启用时， LFU 指令卸载 LIFO 中 .POS 处的值，并将该值放置到 Destination。每次启用该指令时，它卸载一个值并用 0 替换该值，直到 LIFO 空为止。如果 LIFO 为空，则 LFU 向 Destination 返回 0。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

LFU 指令对连续内存进行操作。该指令有时从标记的其他成员卸载数据。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

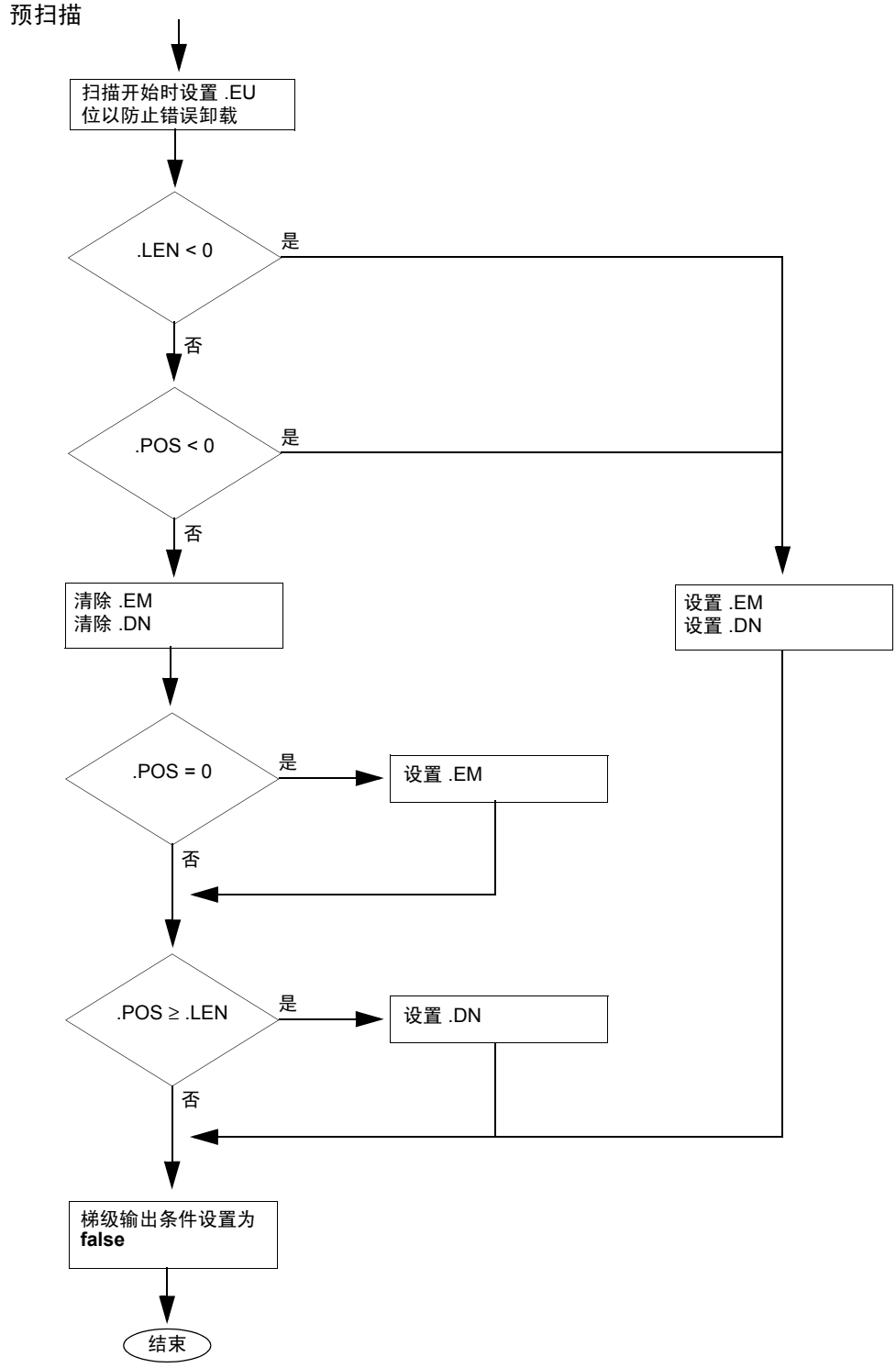
算术状态标志： 不受影响

错误条件：

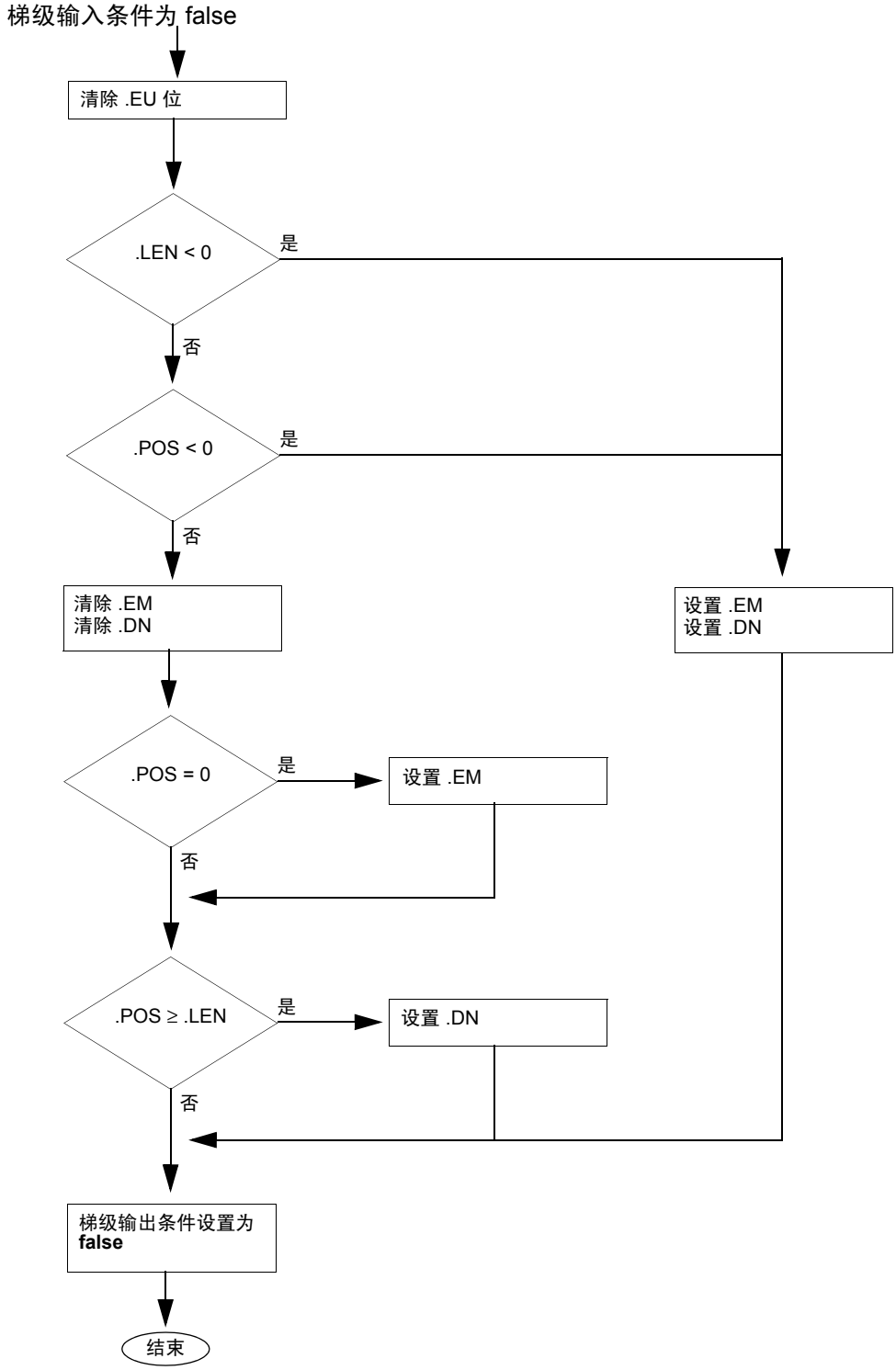
出现严重错误的条件：	错误类型：	错误代码：
Length > LIFO 数组大小	4	20

执行：

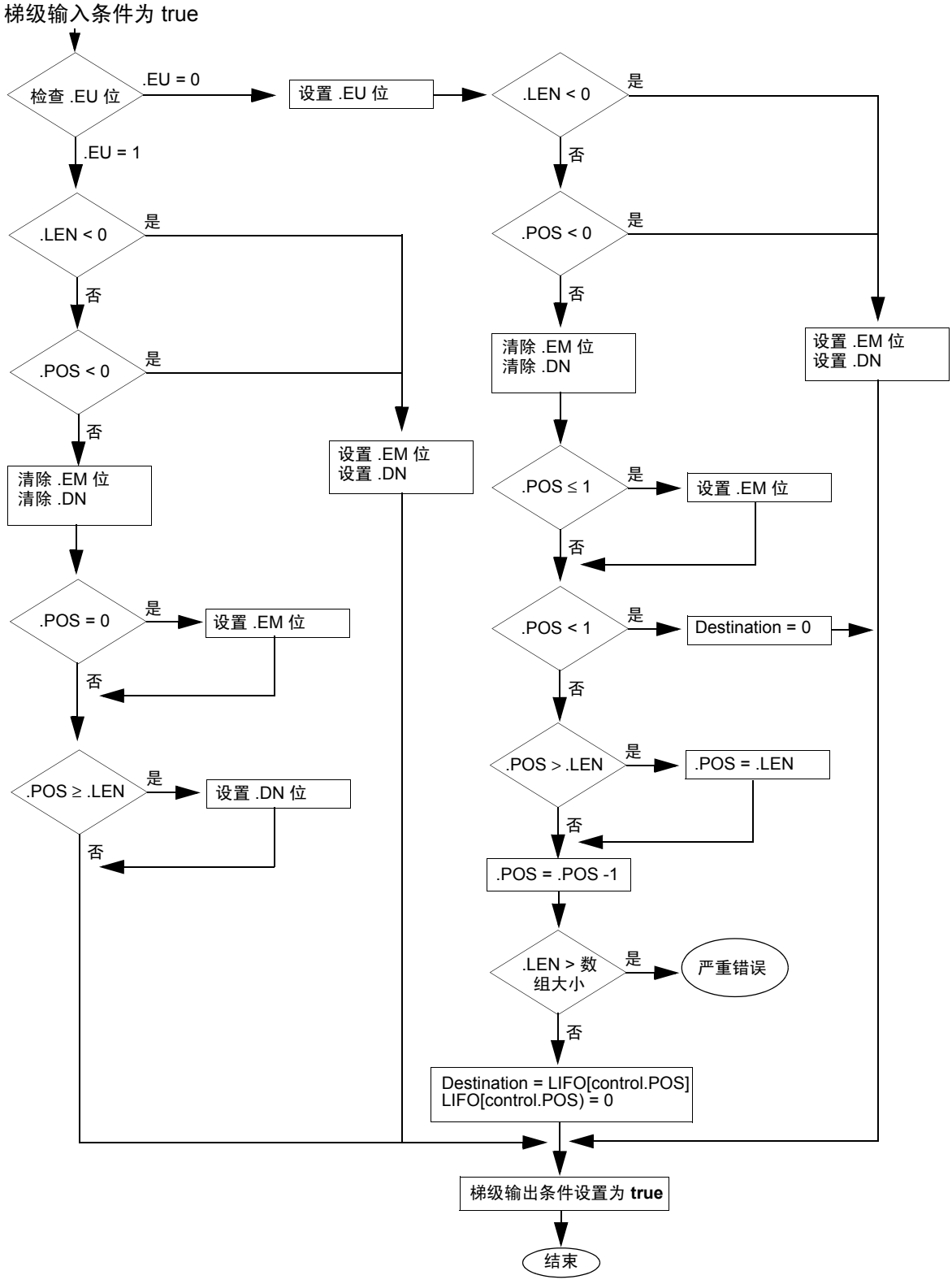
条件：	梯形图操作：


预扫描
扫描开始时设置 .EU 位以防止错误卸载
.LEN < 0
是
否
.POS < 0
是
否
清除 .EM
清除 .DN
设置 .EM
设置 .DN
.POS = 0
是
设置 .EM
否
.POS ≥ .LEN
是
设置 .DN
否
梯级输出条件设置为 false
结束

条件：	梯形图操作：

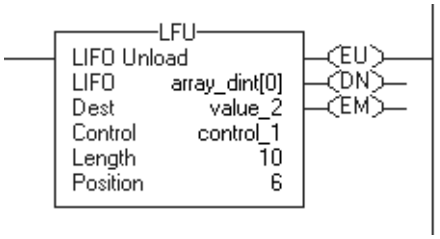

梯级输入条件为 false
清除 .EU 位
.LEN < 0
是
否
.POS < 0
是
否
清除 .EM
清除 .DN
设置 .EM
设置 .DN
.POS = 0
是
设置 .EM
否
.POS ≥ .LEN
是
设置 .DN
否
梯级输出条件设置为 false
结束

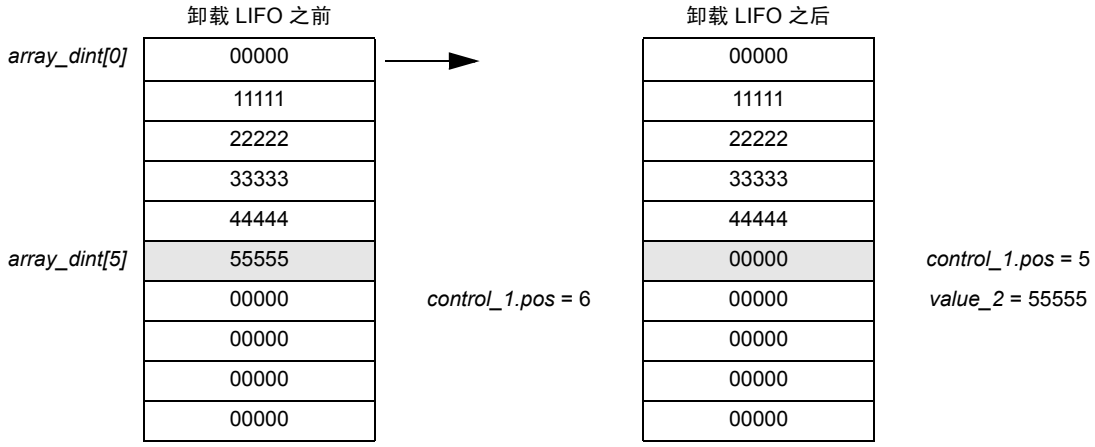
条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， LFU 指令将 *array_dint[5]* 卸载到 *value_2*。





说明：

定序程序指令
（SQI、SQO、SQL）

简介

不执行任何操作。定序程序指令监视一致且可重复的操作。

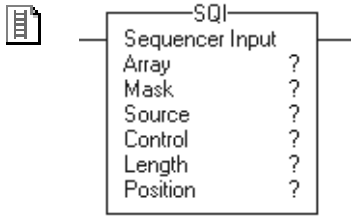
如果需要：	请使用指令：	在以下语言中可用：	请参见页：
检测一步何时完成。	SQI	梯形图	第 9 章 -2
为下一步设置输出条件。	SQO	梯形图	第 9 章 -6
向定序程序数组加载参考条件	SQL	梯形图	第 9 章 -10

对于梯形图指令，**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样的最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

定序程序输入 (Sequencer Input, SQI)

SQI 指令检测 SQO/SQI 指令顺序对何时完成一步。

操作数：



梯形图

操作数：	类型：	格式：	说明：
Array	DINT	数组标记	定序程序数组 指定定序程序数组的第一个元素 **不要** 在下标中使用 CONTROL.POS
Mask	SINT INT **DINT**	标记 立即数	要屏蔽或传递的位
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
Source	SINT INT DINT	标记	定序程序数组的输入数据
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
CONTROL	CONTROL	标记	控制运算的结构 通常使用与 SQO 和 SQL 指令相同的 CONTROL
Length	DINT	立即数	用于比较的 Array （定序程序表）中元素的数目
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 时设置错误位。
.LEN	DINT	长度指定定序程序数组中的步骤数。
.POS	DINT	位置标识指令当前正比较的元素的位置。

说明： 启用时， SQI 指令通过 Mask 来比较 Source 元素与 Array 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。

输入立即数掩码值

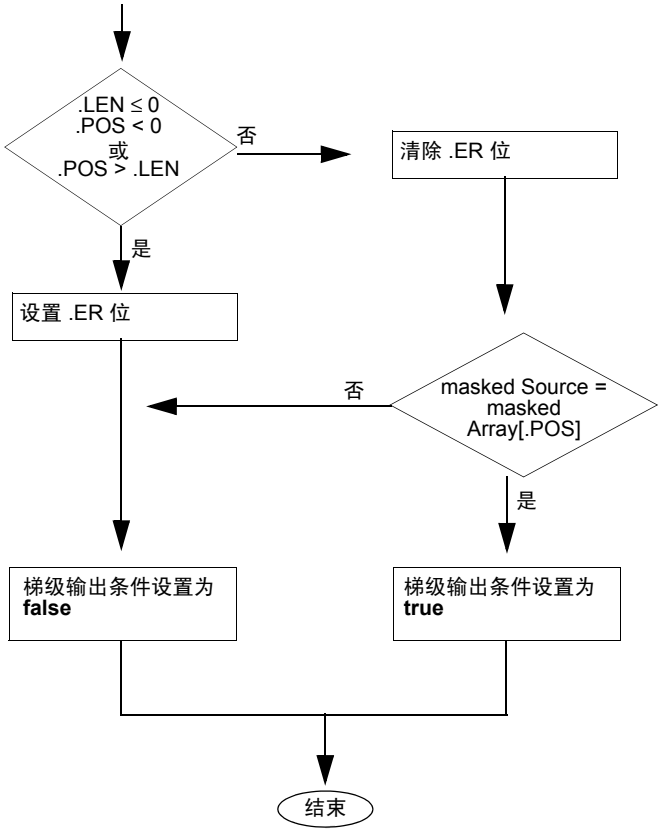
输入掩码时，编程软件默认为十进制值。如果您想用其他格式输入掩码，请在值前加上正确的前缀。

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

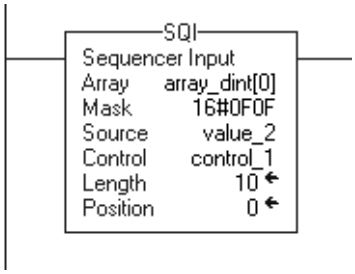
算术状态标志： 不受影响

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	
后扫描	梯级输出条件设置为 false。

示例： 启用时， SQI 指令通过掩码传递 *value_2*，以确定结果是否等于 *array_dint* 中的当前元素。因为掩码比较结果为 true，所以梯级输出条件变为 true。



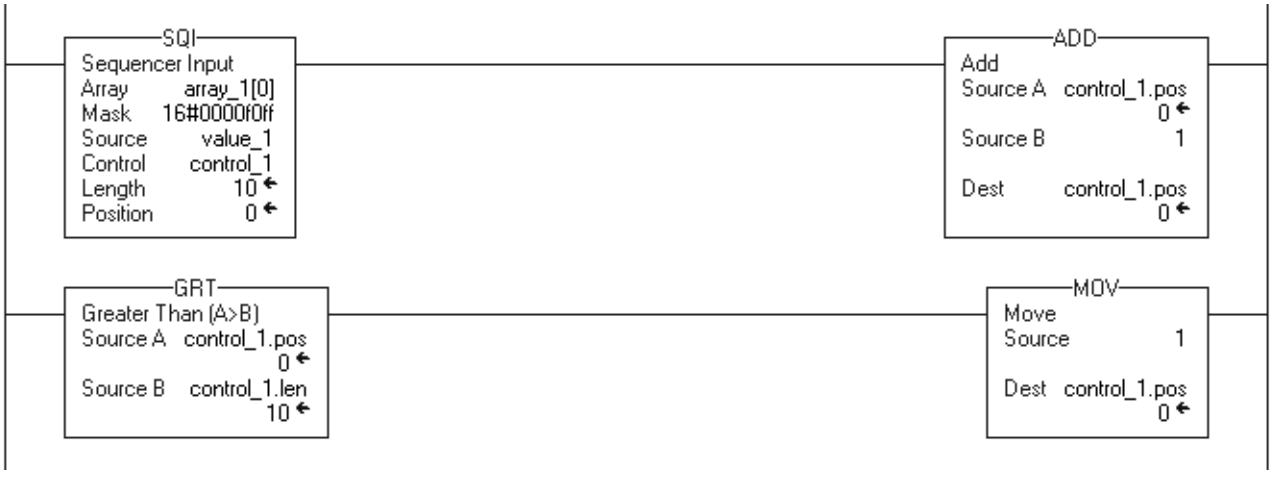
SQI 操作数：	本例的值 （以二进制显示的 DINT）：
Source	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010
Mask	00000000 00000000 00001111 00001111
Array	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010

掩码中的 0 表示不比较相应位 （本示例中由 xxxx 指定）。

只用 SQI 不用 SQO

如果您只用 SQI 指令而不用成对的 SQO 指令，您必须在外部增加定序程序数组。

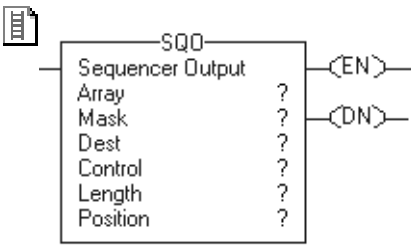
SQI 指令对源值进行比较。 ADD 指令增加定序程序数组。 GRT 确定定序程序数组中是否存在另一个可以检查的值。在一次从头到尾完成定序程序数组的所有步骤后， MOV 指令复位位置值。



定序程序输出 (Sequencer Output, SQO)

SQO 指令设置 SQO/SQI 指令顺序对的下一步输出条件。

操作数：



梯形图

操作数：	类型：	格式：	说明：
Array	DINT	数组标记	定序程序数组 指定定序程序数组的第一个元素 **不要** 在下标中使用 CONTROL.POS
Mask	SINT INT **DINT** SINT 或 INT 标记通过符号扩展转换为 DINT 值。	标记 立即数	要屏蔽或传递的位
Destination	DINT	标记	从定序程序数组输出的数据
CONTROL	CONTROL	标记	控制运算的结构 通常使用与 SQI 和 SQL 指令相同的 CONTROL
Length	DINT	立即数	Array （定序程序表）中输出元素的数目
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	启用位指示 SQO 指令已启用。
.DN	BOOL	当所有指定的元素都被移入 Destination 时设置完成位。
.ER	BOOL	当 .LEN ≤ 0、 .POS < 0 或 .POS > .LEN 时设置错误位。
.LEN	DINT	长度指定定序程序数组中的步骤数。
.POS	DINT	位置标识控制器当前操作的元素。

说明： 启用时， SQO 指令增加位置值，通过 Mask 移动位置中的数据，并将结果存储到 Destination。如果 .POS > .LEN，则指令返回到定序程序数组的开始处，并从 .POS = 1 处继续执行。

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。

输入立即数掩码值

输入掩码时，编程软件默认为十进制值。如果您想用其他格式输入掩码，请在值前加上正确的前缀。

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

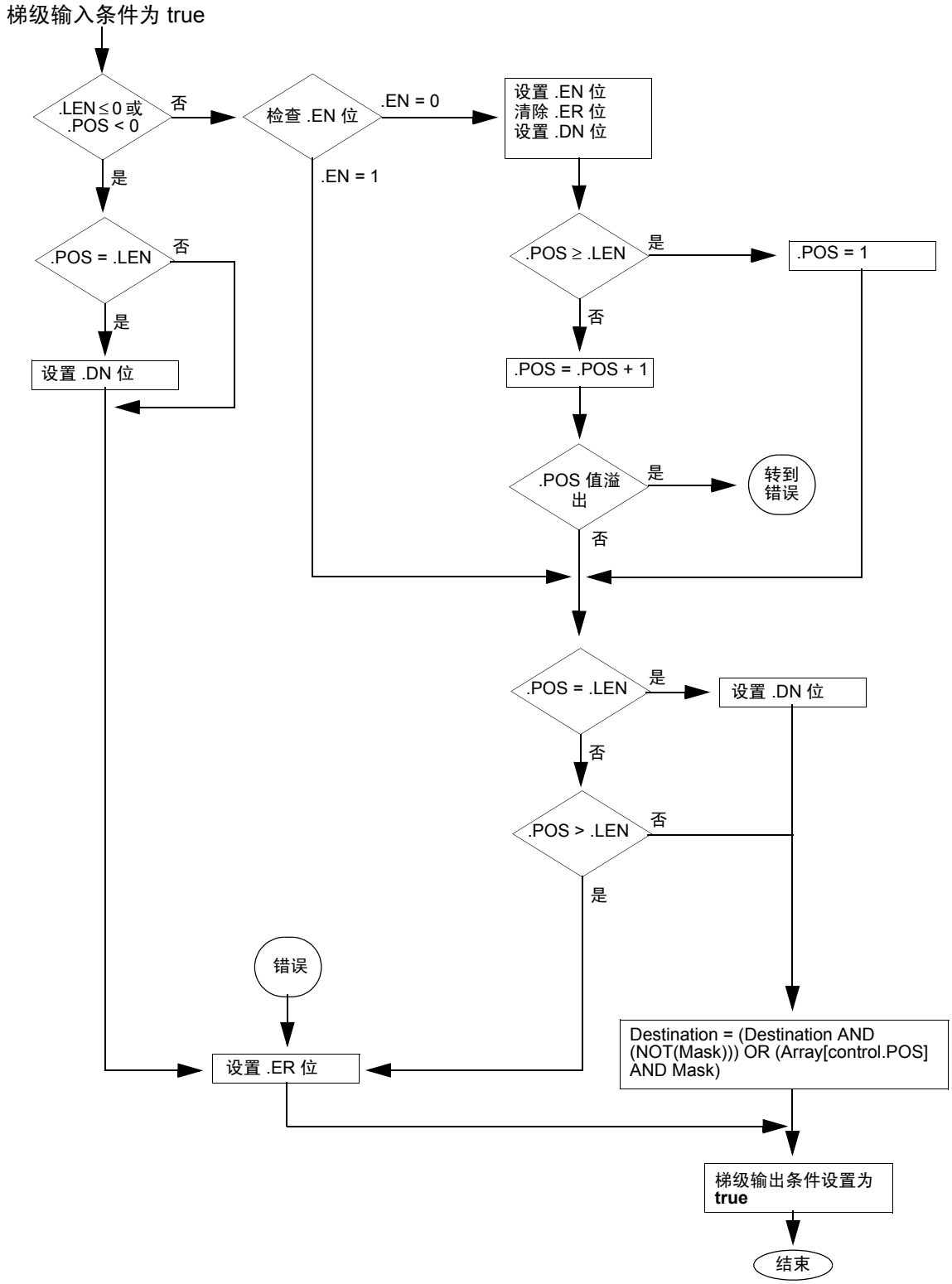
算术状态标志： 不受影响

错误条件： 无

执行：

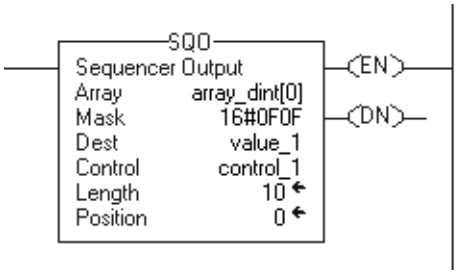
条件：	梯形图操作：
预扫描	程序扫描开始时设置 .EN 位以防止错误加载。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .EN 位。 梯级输出条件设置为 false。

条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， SQO 指令增加位置，通过掩码传递 *array_dint* 中该位置的数据，并将结果存储到 *value_1*。



SQO 操作数：	本例的值 （以二进制显示的 INT）：
Array	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010
Mask	00000000 00000000 00001111 00001111
Destination	xxxxxxxx xxxxxxxx xxxx0101 xxxx1010

掩码中的 0 表示不比较相应位 （本示例中由 xxxx 指定）。

SQI 与 SQO 一起使用

如果您成对使用 SQI 指令和 SQO 指令，请确保这两条指令使用相同的 Control、 Length 和 Position 值。



复位 SQO 的位置

每当控制器从程序模式进入运行模式时， SQO 指令都将 .POS 值清零（初始化）。要将 .POS 复位为初始值 (.POS = 0)，请使用 RES 指令将位置值清零。本示例用首次扫描位的状态来将 .POS 值清零。

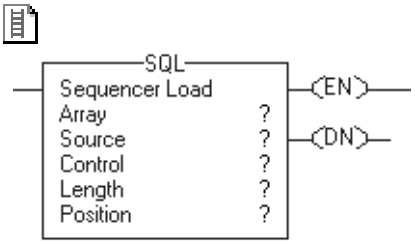


定序程序加载 (Sequencer Load, SQL)

SQL 指令将参考条件加载到定序程序数组中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Array	DINT	数组标记	定序程序数组 指定定序程序数组的第一个元素 **不要** 在下标中使用 CONTROL.POS
Source	SINT INT **DINT**	标记 立即数	要加载到定序程序数组中的输入数据
	SINT 或 INT 标记通过符号扩展转换为 DINT 值。		
CONTROL	CONTROL	标记	控制运算的结构 通常使用与 SQI 和 SQO 指令相同的 CONTROL
Length	DINT	立即数	要加载到 Array （定序程序表）中的元素的数目
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	启用位指示 SQL 指令已启用。
.DN	BOOL	当所有指定元素都被加载到 Array 中时设置完成位。
.ER	BOOL	当 .LEN ≤ 0、 .POS < 0 或 .POS > .LEN 时设置错误位。
.LEN	DINT	长度指定定序程序数组中的步骤数。
.POS	DINT	位置标识控制器当前操作的元素。

说明： 启用时，SQL 指令增加位置值到定序程序数组中的下一个位置，并将 Source 值加载到该位置。如果 .DN 位处于置位状态，或者如果 .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

您 **《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。有些情况下，该指令将超出数组容量的数据加载到标记的其他成员中。如果长度过大并且标记是用户自定义的数据类型，则会发生这种情况。

算术状态标志： 不受影响

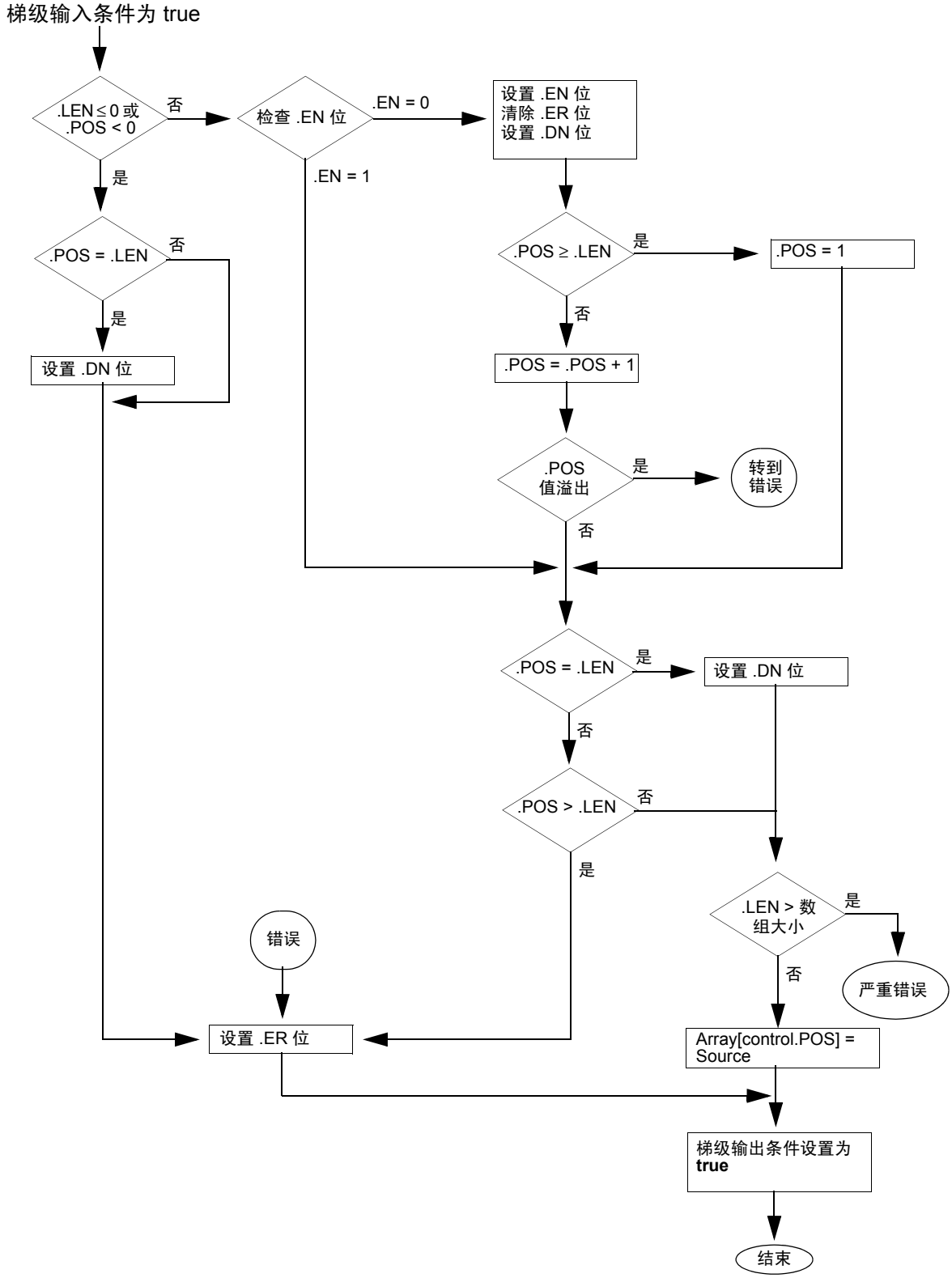
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
Length > Array 的大小	4	20

执行：

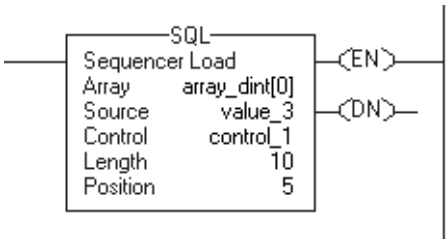
条件：	梯形图操作：
预扫描	程序扫描开始时设置 .EN 位以防止错误加载。 梯级输出条件设置为 false。
梯级输入条件为 false	清除 .EN 位。 梯级输出条件设置为 false。

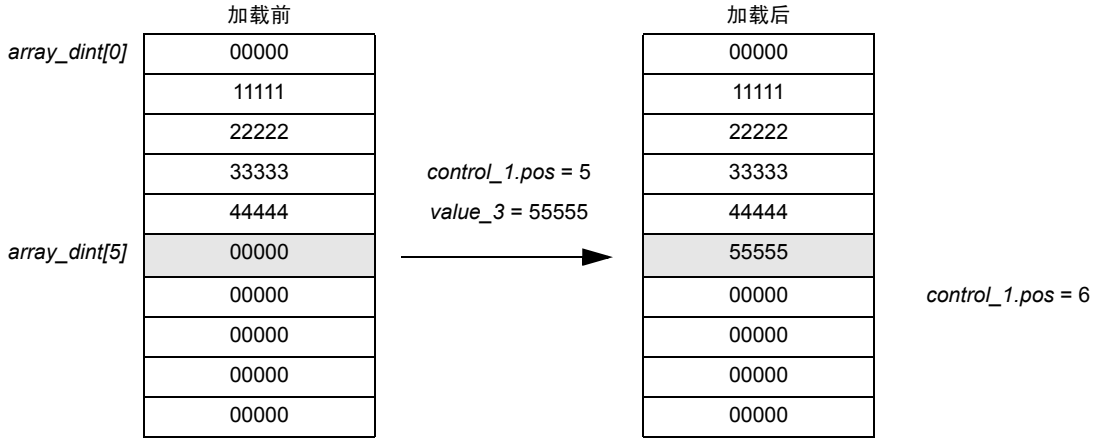
条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， SQL 指令将 *value_3* 加载到定序程序数组的下一个位置，即本例中的 *array_dint[5]*。





说明：

第 10 章

程序控制指令
（JMP、LBL、JSR、RET、SBR、JXR、TND、MCR、UID、UIE、AFI、NOP、EOT、SFP、SFR、EVENT）

简介

使用程序控制指令更改逻辑执行顺序。

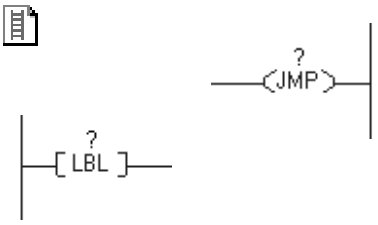
如果需要：	请使用指令：	在以下语言中可用：	请参见页：
跳过不需要始终执行的逻辑段。	JMP LBL	梯形图	第 10 章 -2
跳转到单独的例程，向例程传递数据，执行例程并返回结果。	JSR SBR RET	梯形图 功能块 结构化文本	第 10 章 -4
跳转到外部例程 （仅限 SoftLogix5800 控制器）	JXR	梯形图	第 10 章 -14
标记暂停例程执行的临时结束。	TND	梯形图 结构化文本	第 10 章 -17
禁用逻辑段中的所有梯级。	MCR	梯形图	第 10 章 -19
禁用用户任务。	UID	梯形图 结构化文本	第 10 章 -21
启用用户任务。	UIE	梯形图 结构化文本	第 10 章 -21
禁用梯级。	AFI	梯形图	第 10 章 -23
在逻辑中插入占位符。	NOP	梯形图	第 10 章 -24
结束顺序功能图的转换。	EOT	梯形图 结构化文本	第 10 章 -25
暂停顺序功能图。	SFP	梯形图 结构化文本	第 10 章 -27
复位顺序功能图。	SFR	梯形图 结构化文本	第 10 章 -29
触发事件任务的执行。	EVENT	梯形图 结构化文本	第 10 章 -31

跳转至标号 (Jump to Label, JMP) 标号 (Label, LBL)

JMP 和 LBL 指令跳过部分梯形逻辑。

操作数：

梯形图



操作数：	类型：	格式：	说明：
JMP 指令			
标号名称		标号名称	输入关联的 LBL 指令的名称
LBL 指令			
标号名称		标号名称	执行跳转到具有所引用标号名称的 LBL 指令

说明： 启用时， JMP 指令跳转到所引用的 LBL 指令，控制器从该处继续执行。禁用时， JMP 指令不会影响梯形执行。

JMP 指令可以将梯形执行前移或后移。向前跳转至标号会忽略不需要的逻辑片段，从而有助于节省程序扫描时间。向后跳转使控制器可重复逻辑迭代。

注意，向后跳转的次数不能过多。由于控制器永远不会到达逻辑的末尾，因此监视计时器可能超时，从而导致控制器出现故障。

注意



不扫描跳过的逻辑。应将关键逻辑放在跳过的区域外。

LBL 指令是具有相同标号名称的 JMP 指令的目标。**确保 LBL 指令是所在梯级的第一条指令。**

标号名称在例程中必须唯一。该名称：

- 最多可以有 40 个字符
- 可以包含字母、数字和下划线 (_)

算术状态标志： 不受影响

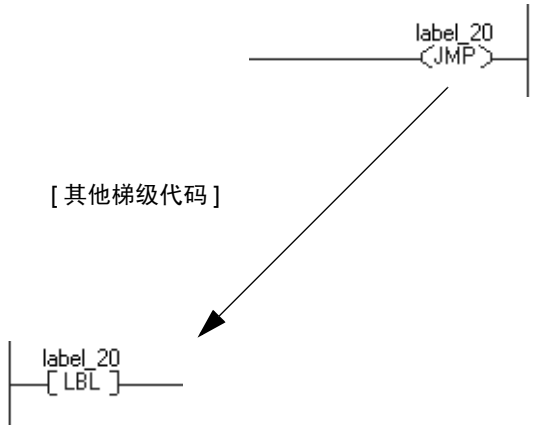
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	梯级输出条件设置为 true。 执行跳转到具有所引用标号名称的 LBL 指令所在的梯级。
后期扫描	梯级输出条件设置为 false。

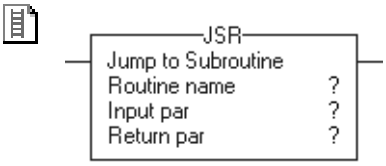
示例： 启用 JMP 指令时，执行跳过逻辑的连续梯级，直到到达包含具有 *label_20* 的 LBL 指令的梯级。



跳转到子例程 (Jump to Subroutine, JSR)
子例程 (Subroutine, SBR)
返回 (Return, RET)

JSR 指令使执行跳转到其他例程。SBR 指令和 RET 指令为可选指令，它们与 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	名称	要执行的例程 （即子例程）
输入参数	BOOL SINT INT DINT REAL 结构	立即数 标记 数组标记	此例程中要向子例程中的标记复制的数据 • 输入参数是可选的。 • 如果需要，输入多个输入参数。
返回参数	BOOL SINT INT DINT REAL 结构	标记 数组标记	此例程中的要向其中复制子例程结果的标记 • 返回参数是可选的。 • 如果需要，输入多个返回参数。

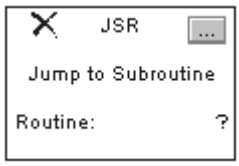
结构化文本

```
JSR(RoutineName,InputCount,
InputPar,ReturnPar);
```

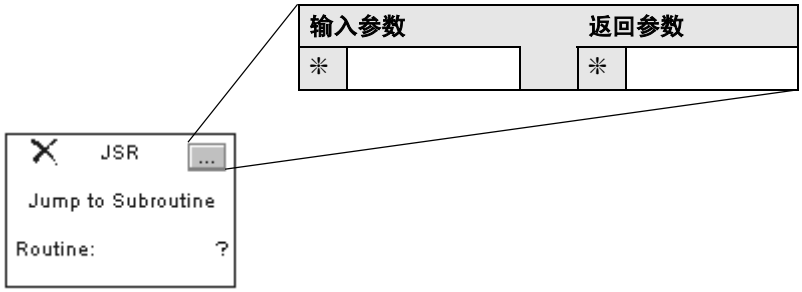
操作数：	类型：	格式：	说明：
例程名称	ROUTINE	名称	要执行的例程 （即子例程）
输入计数	SINT INT DINT REAL	立即数	输入参数的数目
输入参数	BOOL SINT INT DINT REAL 结构	立即数 标记 数组标记	此例程中要向子例程中的标记复制的数据 • 输入参数是可选的。 • 如果需要，输入多个输入参数。
返回参数	BOOL SINT INT DINT REAL 结构	标记 数组标记	此例程中的要向其中复制子例程结果的标记 • 返回参数是可选的。 • 如果需要，输入多个返回参数。

（请继续查看下一页的 JSR 操作数）

JSR 操作数 （续）



功能块



操作数与梯形图 JSR 指令的操作数相同。

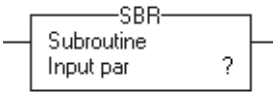
注意



对于 SBR 指令或 RET 指令中的每个参数，请使用相同的数据类型 （包括任何数组维数）作为 JSR 指令中的对应参数。使用不同的数据类型可能产生意外的结果。

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



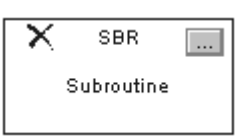
操作数：	类型：	格式：	说明：
输入参数	BOOL SINT INT DINT REAL 结构	标记 数组标记	此例程中的标记， JSR 指令中的对应输入参数将复制到该标记中

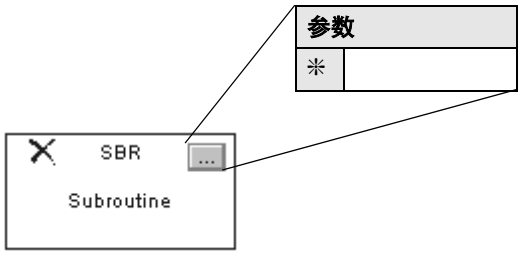
结构化文本

```
SBR(InputPar);
```

操作数与梯形图 SBR 指令的操作数一样。

功能块

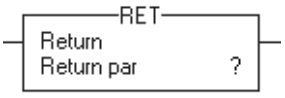




操作数与梯形图 SBR 指令的操作数一样。

RET 操作数：

梯形图



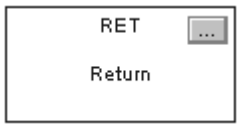
操作数：	类型：	格式：	说明：
返回参数	BOOL SINT INT DINT REAL 结构	立即数 标记 数组标记	此例程中要复制到 JSR 指令中的对应返回参数的数据

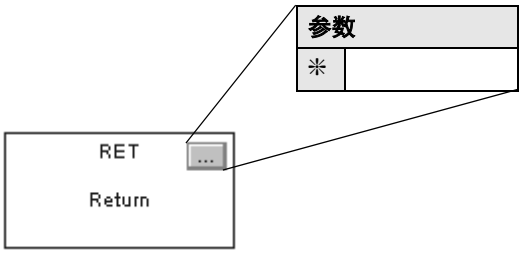
结构化文本

```
RET(ReturnPar);
```

操作数与梯形图 SET 指令的操作数一样。

功能块





操作数与梯形图 SET 指令的操作数一样。

说明： JSR 指令启动指定例程的执行，该例程作为子例程被引用：

- 子例程执行一次。
- 子例程执行后，逻辑执行返回到包含 JSR 指令的例程。

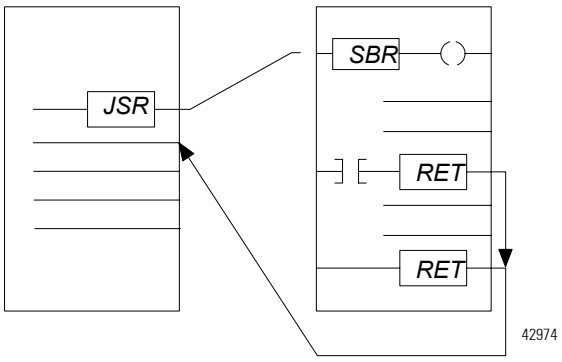
要编写到子例程的跳转，请遵循以下准则：

重要事项

不要使用 JSR 指令调用 （执行）主例程。

- 可以将 JSR 指令放入主例程或其他任何例程中。
- 如果使用 JSR 指令调用主例程，然后将 RET 指令放入主例程中，会出现严重错误 （类型 4，代码 31）。

下图解释指令的操作方式。



JSR

1. 如果要向子例程中的标记复制数据，则输入输入参数。
2. 如果要向此例程中的标记复制子例程的结果，则输入返回参数。
3. 根据需要输入任意数目的输入参数和返回参数。

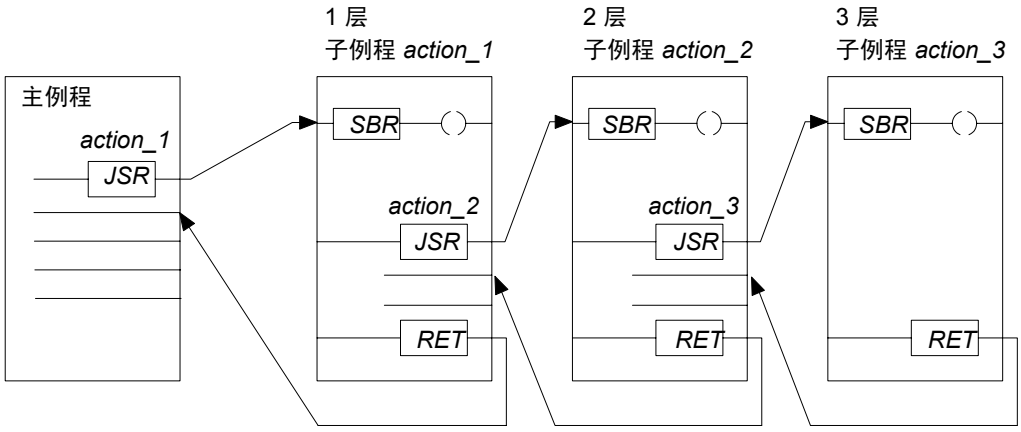
SBR

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 指令中的各输入参数，输入要向其中复制数据的标记。

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 指令中的各返回参数，输入要发送到 JSR 指令的返回参数。
4. 在梯形例程中，如有必要，可根据不同的输入条件放置其他 RET 指令以退出子例程。（功能块例程仅允许一个 RET 指令。）

对于您可以使用的嵌套例程的数目或可传递或返回的参数的数目，除受控制器内存限制外，不受其他因素的限制。



算术状态标志： 算术状态标志受影响。

错误条件：

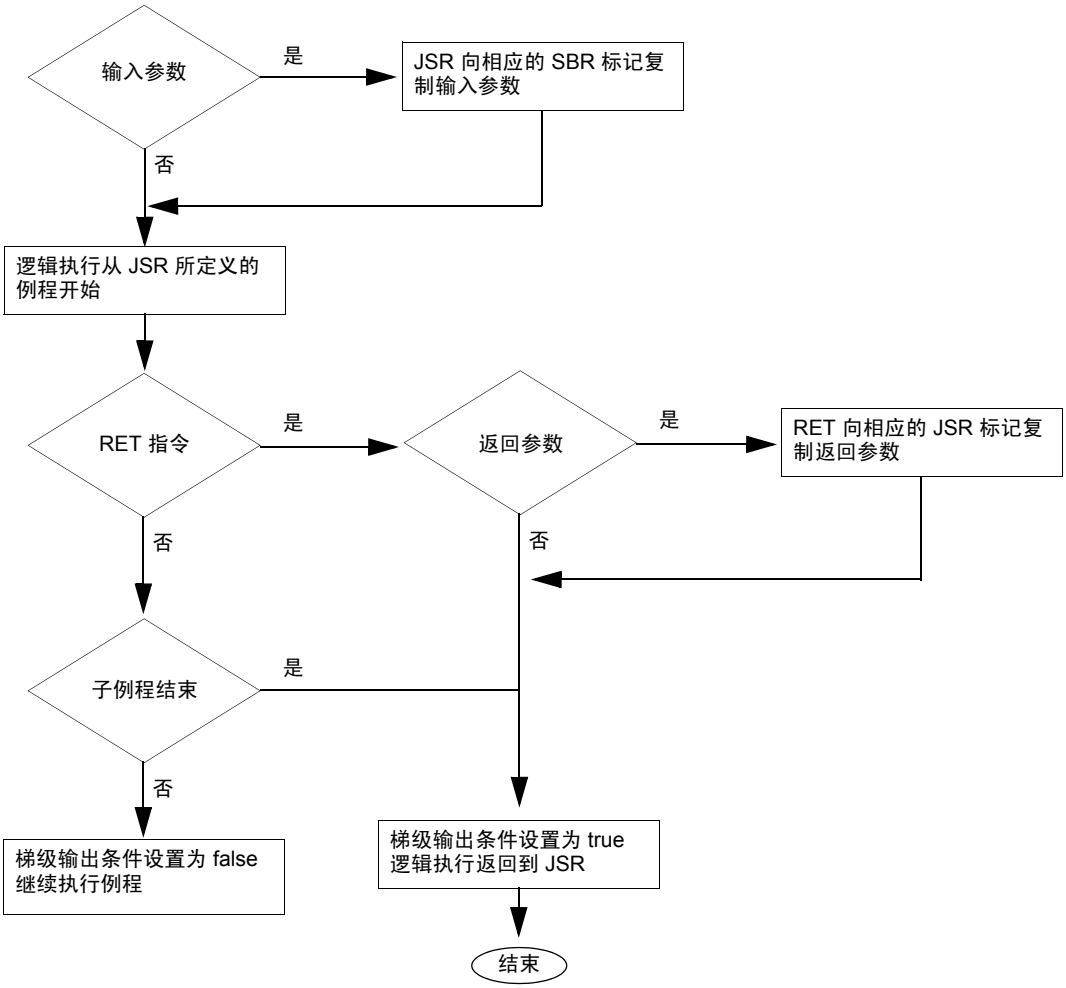
出现严重错误的条件：	错误类型：	错误代码：
JSR 指令的输入参数比 SBR 指令的输入参数少	4	31
JSR 指令跳转到了错误的例程	4 或用户提供	0 或用户提供
RET 指令的返回参数比 JSR 指令的返回参数少	4	31
主例程包含一个 RET 指令	4	31

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	控制器执行所有子例程，而不考虑梯级条件。为了确保预扫描子例程中的所有梯级，控制器忽略 RET 指令。（即， RET 指令不退出子例程。） • 版本 6.x 和更早版本，传递输入参数和返回参数。 • 版本 7.x 和更高版本，不传递输入参数和返回参数。 如果存在对同一子例程的递归调用，则仅在第一次预扫描该子例程。如果存在对同一子例程的多个调用 （非递归），则每次都会预扫描该子例程。 梯级输出条件设置为 false （仅限梯形图）。	
对于 JSR 指令，梯级输入条件为 false	子例程 《不》执行。 子例程中的输出保持上次的状态。 梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。

条件：	梯形图操作：	结构化文本操作：
指令执行		
后期扫描	与上述预扫描执行相同的操作。	与上述预扫描执行相同的操作。



功能块

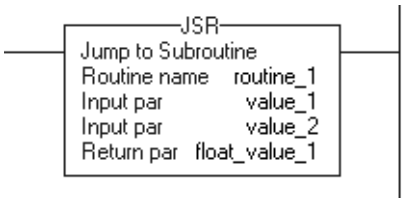
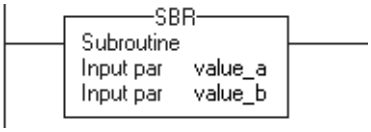
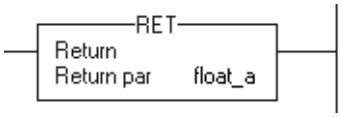
条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
正常执行	1. 如果例程包含 SBR 指令，控制器将首先执行该 SBR 指令。 2. 控制器获取 IREF 中的所有数据值。 3. 控制器按照功能块的编写顺序执行其他功能块。其中包括其他 JSR 指令。 4. 控制器将输出写入 OREF。 5. 如果例程包含 RET 指令，控制器将最后执行该 RET 指令。
后期扫描	调用子例程。 如果该例程为 SFC 例程，则与预扫描过程中一样执行初始化。

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。

RET 指令将 *float_a* 发送给 JSR 指令。 JSR 指令接收 *float_a*，并将该值复制到 *float_value_1*。继续执行 JSR 指令后面的下一条指令。

梯形图

例程：	程序：
主例程	
子例程	 [其他梯级代码] 

结构化文本

例程：	程序：
主例程	`JSR(routine_1,2,value_1,value_2,float_value_1);`

子例程	`SBR(value_a,value_b);`
	
	`<语句>;`
	
	`RET(float_a);`

示例 2：

梯形图

MainRoutine

当 *abc* 处于打开状态时， *subroutine_1* 执行，计算 Cookie 的数目，并将值放入 *cookies_1* 中。

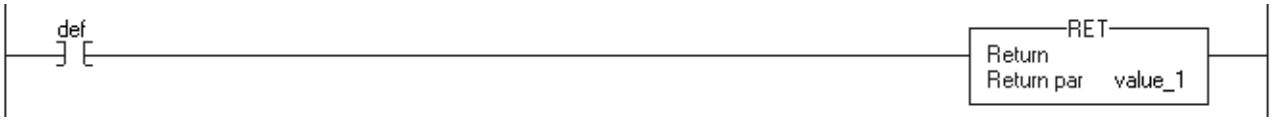


将 *cookies_1* 中的值与 *cookies_2* 相加，并将结果存储在 *total_cookies* 中。

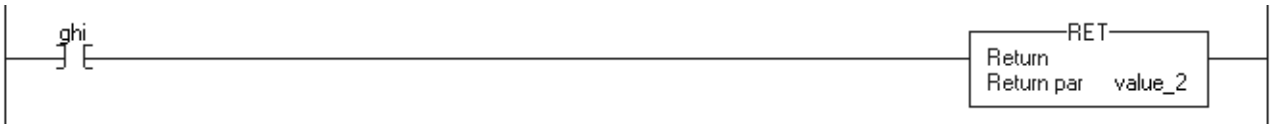


Subroutine_1

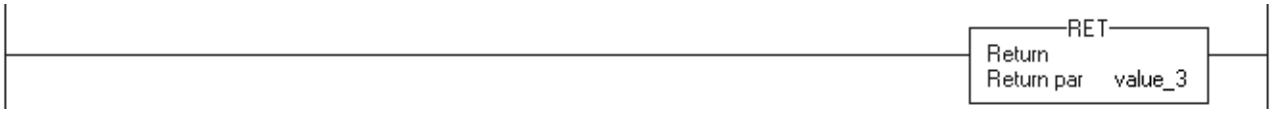
当 *def* 处于打开状态时， RET 指令将 *value_1* 返回给 JSR *cookies_1* 参数，不扫描子例程的其余部分。



当 *def* 处于关闭状态 （前一个梯级）而 *ghi* 处于打开状态， RET 指令将 *value_2* 返回给 JSR *cookies_1* 参数，不扫描子例程的其余部分。



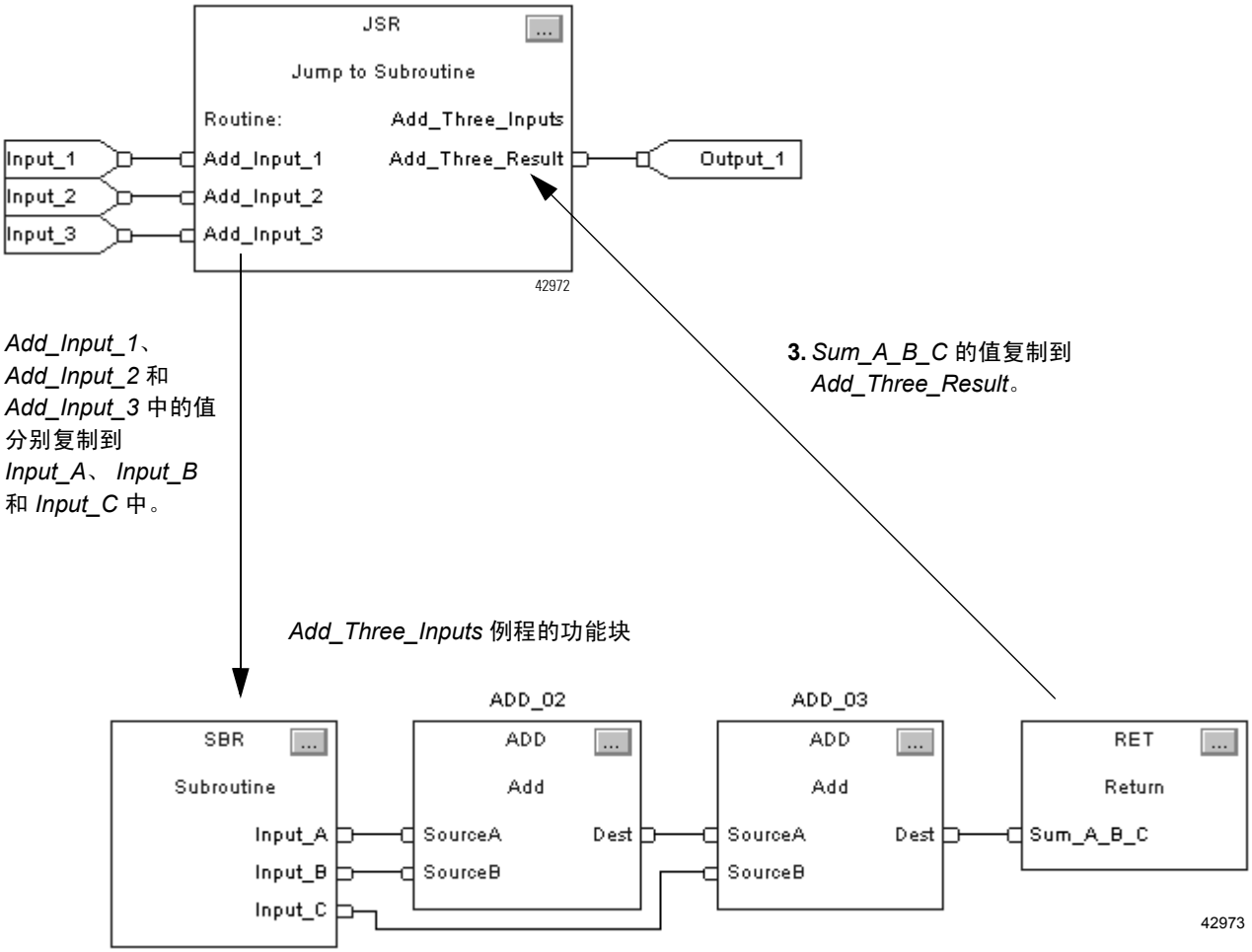
当 *def* 和 *ghi* 都处于关闭状态 （前面的梯级）时， RET 指令将 *value_3* 返回给 JSR *cookies_1* 参数。



示例 3：

功能块

JSR 指令位于 *Routine_A* 中



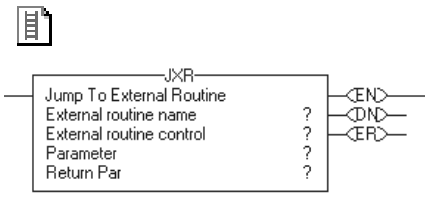
2. ADD 指令添加 *Input_A*、 *Input_B* 和 *Input_C*，并将结果放入 *Sum_A_B_C* 中。

跳转到外部例程 (Jump to External Routine, JXR)

JXR 指令执行外部例程。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	ROUTINE	名称	要执行的外部例程
外部例程控制	EXT_ROUTINE_CONTROL	标记	控制结构 （请参见下一页）
参数	BOOL SINT INT DINT REAL 结构	立即数 标记 数组标记	此例程中要复制到外部例程中的变量的数据 • 参数可选。 • 如果需要，可输入多个参数。 • 最多可以有 10 个参数。
返回参数	BOOL SINT INT DINT REAL	标记	此例程中的标记，将向其中复制外部例程的结果 • 返回参数可选。 • 只能有一个返回参数。

EXT_ROUTINE_CONTROL 结构

助记符：	数据类型：	说明：	实现：
ErrorCode	SINT	如果出现错误，该值将标识该错误。有效值为 0-255。	没有预定义的错误代码。外部例程的开发人员必须提供错误代码。
NumParams	SINT	该值表示与该指令关联的参数的数目。	仅显示 - 此信息源自指令输入。
ParameterDefs	EXT_ROUTINE_PARAMETERS[10]	此数组包含要传递给外部例程的参数的定义。该指令最多可以传递 10 个参数。	仅显示 - 此信息源自指令输入。
ReturnParamDef	EXT_ROUTIN_PARAMETERS	该值包含来自外部例程的返回参数的定义。只有一个返回参数。	仅显示 - 此信息源自指令输入。
EN	BOOL	如果处于设置状态，启用位表示已启用 JXR 指令。	外部例程设置此位。
ReturnsValue	BOOL	如果处于设置状态，该位表示为此指令输入了一个返回参数。如果处于清零状态，该位表示没有为此指令输入返回参数。	仅显示 - 此信息源自指令输入。
DN	BOOL	当外部例程完整地执行一次后，将设置完成位。	外部例程设置此位。
ER	BOOL	如果出现错误，将设置错误位。指令会停止执行，直到程序清零该错误位。	外部例程设置此位。
FirstScan	BOOL	该位标识这次扫描是否是将控制器切换到”运行 “模式后的首次扫描。如果需要，可使用 FirstScan 初始化外部例程。	控制器设置此位以反映扫描状态。
EnableOut	BOOL	启用输出。	外部例程设置此位。
EnableIn	BOOL	启用输入。	控制器设置此位以反映梯级输入条件。无论梯级条件如何，该指令都将执行。外部例程的开发人员应监视此状态并采取相应操作。
User1	BOOL	这些位供用户使用。控制器不初始化这些位。	外部例程或用户程序可设置这些位。
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 后期扫描 （不适用于梯形图程序）	控制器设置这些位以反映扫描状态。
ScanType0	BOOL		

说明： 使用跳转到外部例程 (JXR) 指令从项目中的梯形例程调用外部例程。JXR 指令支持多个参数，因此可以在梯形例程与外部例程之间传递值。

JXR 指令与跳转到子例程 (JSR) 指令类似。 JXR 指令启动指定外部例程的执行：

- 外部例程执行一次。
- 外部例程执行后，逻辑执行返回到包含 JXR 指令的例程。

算术状态标志： 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：
• 外部例程 DLL 中出现异常 • 未能加载 DLL • 在 DLL 中未找到入口点	4	88

执行： 根据 DLL 的实现， JXR 可以是同步的也可以是异步的。 DLL 中的代码还决定了响应扫描状态、梯级输入条件状态和梯级输出条件状态的方式。

有关使用 JXR 指令和创建外部例程的更多信息，请参见 *SoftLogix5800 System User Manual* （SoftLogix5800 系统用户手册），出版号 1789-UM002。

临时结束 (Temporary End, TND)

TND 指令充当一个分界线。

操作数：





梯形图操作数

无



```
TND();
```

结构化文本

无

您必须在指令助记符之后输入括号 ()，即使没有操作数也应如此。

说明： 启用时， TND 指令使控制器仅执行此指令之前的逻辑。

启用时， TND 指令充当例程的结尾。当控制器扫描到 TND 指令时，控制器将移动到当前例程的末尾。如果 TND 指令位于子例程中，控制流将返回到调用例程。如果 TND 指令位于主例程中，控制流将返回到当前任务中的下一个程序。

算术状态标志： 不受影响

错误条件： 无

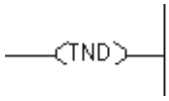
执行：

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	当前例程终止。	当前例程终止。
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 调试或排除故障时可使用 TND 指令，以便执行某个点之前的逻辑。随着调试每个新部分，逐渐在逻辑中移动 TND 指令。

启用 TND 指令时，控制器停止扫描当前例程。

梯形图



结构化文本

```
TND();
```

主控复位 (Master Control Reset, MCR)

MCR 指令成对使用，创建可以禁用 MCR 指令中的所有梯级的程序区域。

操作数：

梯形图



无

说明： 启用 MCR 区域时，将扫描 MCR 区域中的梯级，以查找正常的 true 或 false 条件。禁用 MCR 区域时，控制器仍然扫描 MCR 区域中的梯级，但由于该区域中的非保持输出被禁用，因此扫描时间缩短。对于被禁用的 MCR 区域内部的所有指令，梯级输入条件为 false。

编写 MCR 区域时，请注意：

- 必须以无条件的 MCR 指令结束该区域。
- 不能在一个 MCR 区域中嵌套另一个 MCR 区域。
- 不要跳转到 MCR 区域中。如果区域为 false，跳转到该区域将激活从跳转到的点到区域末尾的区域。
- 如果 MCR 区域持续到例程结束，则不必编写 MCR 指令来结束该区域。

MCR 指令不能取代提供紧急停止功能的硬件连接主控继电器。您仍应安装硬件连接的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。各 MCR 区域必须独立且完整。如果重叠或嵌套，则可能出现不可预测的机器操作并可能造成设备损坏或人员伤害。

应将关键操作放在 MCR 区域外。如果在 MCR 区域中启动诸如计时器这样的指令，则当区域被禁用时，指令执行会停止并且计时器会清零。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。 扫描区域中的指令，但梯级输入条件为 false，且区域中的非保持输出被禁用。
梯级输入条件为 true	梯级输出条件设置为 true。 正常扫描区域中的指令。
后期扫描	梯级输出条件设置为 false。

示例： 当启用第一个 MCR 指令 （已设置 *input_1*、 *input_2* 和 *input_3*）时，控制器执行 MCR 区域 （在两个 MCR 指令之间）中的梯级，并根据输入条件设置或清零输出。

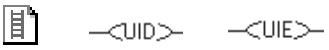
当禁用第一个 MCR 指令 （*input_1*、 *input_2* 和 *input_3* 未全部设置）时，控制器执行 MCR 区域 （在两个 MCR 指令之间）中的梯级，并且无论输入条件如何，对于该 MCR 区域中的所有梯级，梯级输入条件都变为 false。



禁止用户中断 (User Interrupt Disable, UID) 允许用户中断 (User Interrupt Enable, UIE)

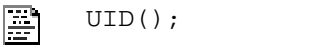
UID 指令和 UIE 指令结合使用，可以防止少数关键梯级被其他任务中断。

操作数：



梯形图

无



```
UID();
UIE();
```

结构化文本

无

您必须在指令助记符之后输入括号 ()，即使没有操作数也应如此。

说明： 当梯级输入条件为 true 时：

- UID 指令防止更高优先级的任务中断当前任务，但 《不》禁用错误例程或控制器错误处理程序的执行。
- UIE 指令允许其他任务中断当前任务。

要防止一系列梯级被中断，请执行下列操作：

1. 尽可能减少 《不》希望中断的梯级的数目。长时间的禁用中断会导致通信的信号损失。
2. 在 《不》希望中断的第一个梯级的上方，输入一个梯级和一个 UID 指令。
3. 在 《不》希望中断的系列中的最后一个梯级之后，输入一个梯级和一个 UIE 指令。
4. 如有必要，可嵌套 UID/UIE 指令对。

算术状态标志： 不受影响

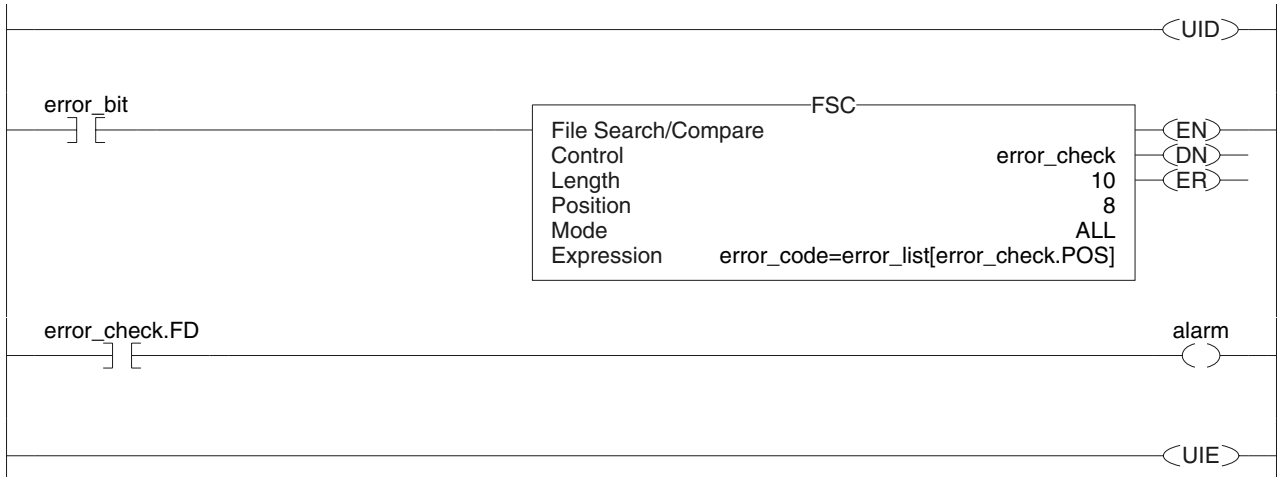
错误条件： 无

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	UID 指令禁止被更高优先级的任务中断。 UIE 指令允许被更高优先级的任务中断。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 出错 （*error_bit* 处于打开状态）时， FSC 指令对照重大错误列表检查错误代码。如果 FSC 指令发现错误为重大错误 （error_check.FD 处于打开状态），将发出警报。 UID 指令和 UIE 指令禁止其他任何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 (Always False Instruction, AFI)

AFI 指令将其梯级输出条件设置为 false。

操作数：

梯形图



无

说明： AFI 指令将其梯级输出条件设置为 false。

算术状态标志： 不受影响

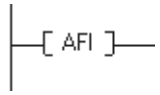
错误条件： 无

执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	梯级输出条件设置为 false。
后期扫描	梯级输出条件设置为 false。

示例： 调试程序时，使用 AFI 指令临时禁用某个梯级。

启用时， AFI 禁用此梯级上的所有指令。



空操作 (No Operation, NOP)

NOP 指令的功能相当于占位符

操作数：

梯形图



无

说明： 可以将 NOP 指令放在梯级上的任何位置。启用时， NOP 指令不执行任何操作。禁用时， NOP 指令也不执行任何操作。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

示例： 当将 NOP 指令放在分支上时，该指令可用于查找无条件分支。

NOP 指令绕过 XIC 指令以启用输出。



转换结束 (End of Transition, EOT)

EOT 指令向 SFC 转换返回一个布尔状态。

操作数：

梯形图



操作数：	类型：	格式：	说明：
数据位	BOOL	标记	转换的状态 （0= 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： 由于 EOT 指令返回一个布尔状态，因此多个 SFC 例程可以共享包含 EOT 指令的同一例程。如果调用例程不是转换， EOT 指令将充当 TND 指令 （请参见第 第 10 章 -17 页）。

EOT 指令的 Logix 实现与 PLC-5 控制器中的 Logix 实现不同。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。在 Logix 控制器中，由于梯级条件在所有 Logix 编程语言中都不可用，因此返回参数会返回转换状态。

算术状态标志： 不受影响

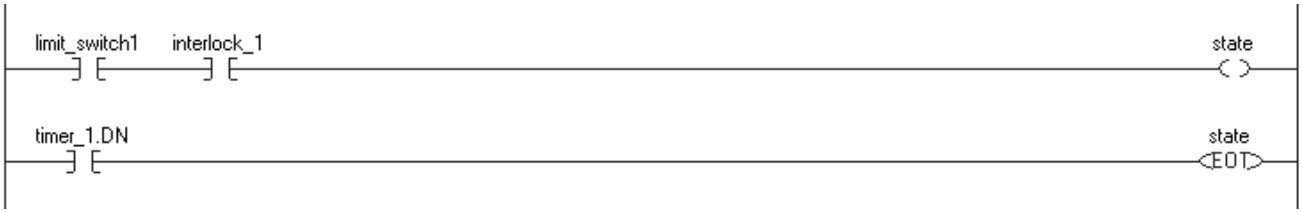
错误条件： 无

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	指令向调用例程返回数据位值。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 当同时设置 *limit_switch1* 和 *interlock_1* 时，将设置状态。 *timer_1* 完成后， EOT 向调用例程返回 *state* 的值。

梯形图



结构化文本

```
state := limit_switch1 AND interlock_1;


IF timer_1.DN THEN

     EOT(state);

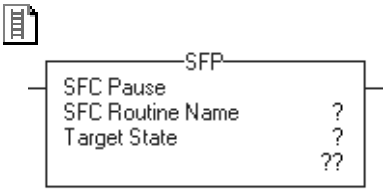
END_IF;
```

SFC 暂停 (SFC Pause, SFP)

SFP 指令暂停 SFC 例程。

操作数：

梯形图



操作数：	类型：	格式：	说明：
SFCRoutine Name	ROUTINE	名称	要暂停的 SFC 例程
TargetState	DINT	立即数 标记	选择一个： 执行 （或输入 0） 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： 使用 SFP 指令可暂停正在执行的 SFC 例程。如果 SFC 例程处于暂停状态，再次使用 SFP 指令将更改状态并继续执行例程。

另外，使用 SFR 指令 （请参见第 第 10 章 -29 页）复位 SFC 例程后，使用 SFP 指令可继续执行 SFC。

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：
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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	该指令暂停或继续指定 SFC 例程的执行。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

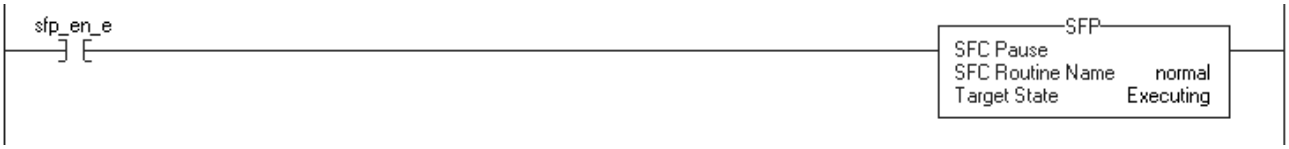
示例： 如果设置 *sfc_en_p*，将暂停名为 *normal* 的 SFC 例程。设置 *sfc_en_e* 时，将重新启动 SFC。

梯形图

暂停 SFC 例程。



继续执行 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;
```

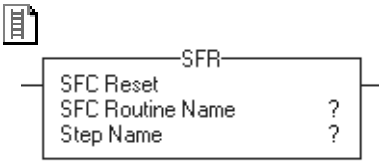
继续执行 SFC 例程：

```
IF (sfp_en_e) THEN
     SFP(normal,executing);
     sfp_en_e := 0;
END_IF;
```

SFC 复位 (SFC Reset, SFR)

SFR 指令在指定步骤复位 SFC 例程的执行。

操作数：



梯形图操作数

操作数：	类型：	格式：	说明：
SFCRoutine Name	ROUTINE	名称	要重置的 SFC 例程
步骤名称	SFC_STEP	标记	目标步骤，将从该步骤开始继续执行

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： 启用 SFR 指令时：

- 在指定的 SFC 例程中，所有存储的操作都停止执行 （重置）。
- SFC 在指定步骤开始执行。

如果目标步骤为 0，图表将重置为初始步骤

SFR 指令在 Logix 中的实现与在 PLC-5 控制器中的实现不同。在 PLC-5 控制器中，当梯级条件为 true 时，会执行 SFR。重置后，在包含 SFR 的梯级变为 false 之前， SFC 将保持暂停状态。这允许重置后延迟执行。PLC-5 SFR 指令的这种暂停 / 取消暂停功能从梯级条件分离出来并移至 SFP 指令中。

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：
例程类型不是 SFC 例程	4	85
SFC 例程中不存在指定的目标步骤	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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	该指令复位指定的 SFC 例程。	该指令复位指定的 SFC 例程。
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 如果出现特定条件 （设置了 *shutdown*），则在步骤 *initialize* 重新启动 SFC。

梯形图



结构化文本

```
IF shutdown THEN

     SFR(mySFC,initialize);

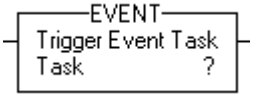
END_IF;
```

触发事件任务 (Trigger Event Task, EVENT)

EVENT 指令触发事件任务的一次执行。

操作数：





梯形图

操作数：	类型：	格式：	说明：
任务	任务	名称	要执行的事件任务 使用该指令可选择其他类型的任务，但它不执行这些任务。



EVENT(*task_name*);

结构化文本

操作数与梯形图 EVENT 指令的操作数相同。

说明： 使用 EVENT 指令以编程的方式执行事件任务：

- 该指令每次执行时都会触发指定的事件任务。
- 在再次触发事件任务之前，请确保给事件任务足够的时间以使其执行完成。否则，将出现重叠。
- 如果在事件任务已经在执行时执行 EVENT 指令，则控制器会增加重叠计数但不会触发该事件任务。

以编程的方式确定 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，可使用获取系统值 (GSV) 指令来监视任务的 Status （状态）属性。

表 10.1 TASK 对象的 Status （状态）属性

属性：	数据类型：	指令：	说明：	
状态	DINT	GSV SSV	提供任务的状态信息。控制器设置一个位后，您必须手动清零该位，以确定是否产生了该类型的另一个错误。	
			要确定是否：	**检查位：**
			EVENT 指令触发任务 （仅限事件任务）。	*0*
			超时触发任务 （仅事件任务）。	1
			任务发生重叠。	2

Status 属性的位一经设置，控制器就不会清零这些位。

- 要将某个位用于新的状态信息，您必须手动清零该位。
- 使用设置系统值 (SSV) 指令将该属性设置为其他值。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	该指令触发指定事件任务的一次执行	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

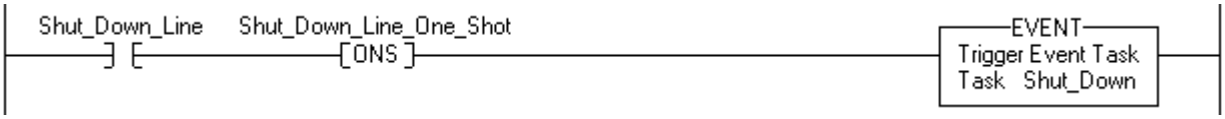
示例 1： 控制器使用多个程序，但使用一个普通的关机程序。各程序都使用一个名为 *Shut_Down_Line* 的程序范围标记，如果程序检测到需要关机的条件，该标记将打开。各程序中的逻辑按以下方式执行：

如果 *Shut_Down_Line* = on （存在需要关机的条件），则

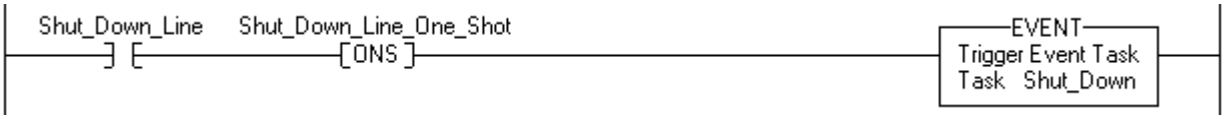
执行一次 *Shut_Down* 任务

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;
```

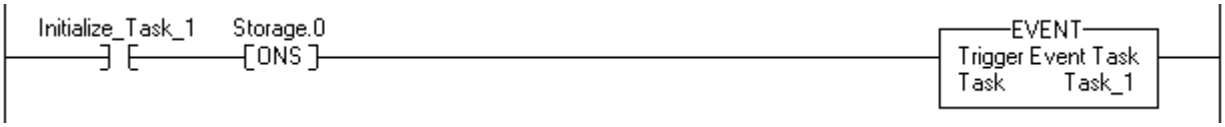
示例 2： 以下示例使用 EVENT 指令初始化事件任务。（其他类型的事件通常触发该事件任务。）

连续任务

如果 *Initialize_Task_1* = 1，则

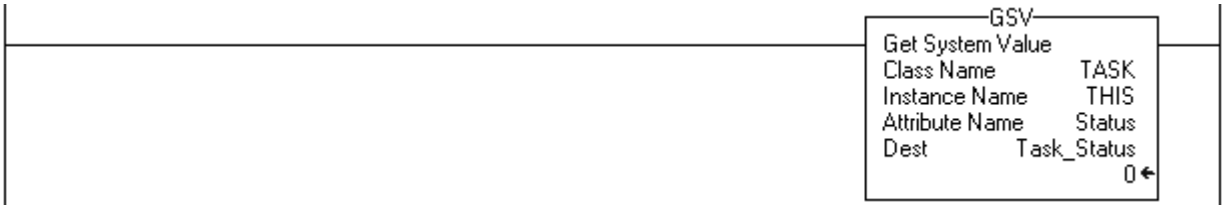
ONS 指令将 EVENT 指令的执行限制为一次扫描。

EVENT 指令触发 *Task_1* （事件任务）的执行。



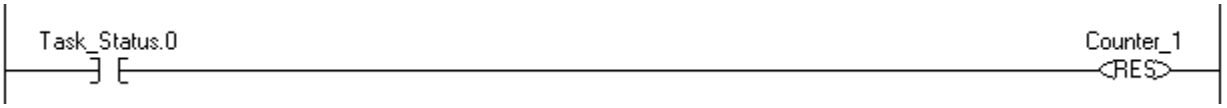
Task_1 （事件任务）

GSV 指令设置 *Task_Status* （DINT 标记）= 事件任务的 Status （状态）属性。在 Instance Name （实例名称）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 EVENT 指令触发事件任务 （即，当持续任务执行其 EVENT 指令以初始化事件任务时）。

RES 指令复位事件任务使用的计数器。

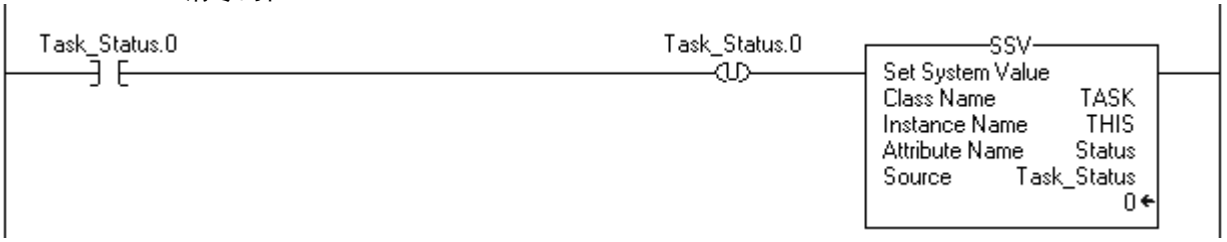


Status 属性的位一经设置，控制器就不会清零这些位。要将某个位用于新的状态信息，您必须手动清零该位。

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 任务的 Status （状态）属性 (*Task_1*) = *Task_Status*。其中包括被清零的位。



第 11 章

For/Break 指令
（FOR、 FOR...DO、 BRK、 EXIT、 RET）

简介

使用 FOR 指令重复调用子例程。使用 BRK 指令中断子例程的执行。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
重复执行例程。	FOR FOR...DO(1)	梯形图 结构化文本	第 11 章 -2
终止例程的重复执行。	BRK EXIT(1)	梯形图 结构化文本	第 11 章 -5
返回到 FOR 指令。	RET	梯形图	第 11 章 -6

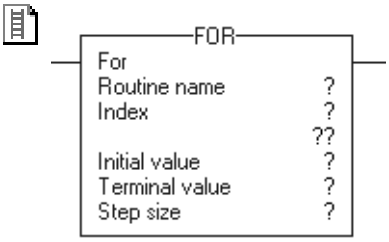
(1) 仅结构化文本。

循环 (For, FOR)

FOR 指令重复执行某个例程。

操作数：

梯形图



操作数：	类型：	格式：	说明：
例程名称	ROUTINE	例程名称	要执行的例程
索引	DINT	标记	计算例程已经执行的次数
Initial value （初始值）	SINT INT DINT	立即数 标记	开始进行索引的值
Terminal value （终止值）	SINT INT DINT	立即数 标记	停止执行例程的值
Step size （步进大小）	SINT INT DINT	立即数 标记	每次 FOR 指令执行例程时要添加到索引的量

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“附录 C”。

说明：

重要事项

不要使用 FOR 指令调用 （执行）主例程。

- 可以在主例程或任何其他例程中使用 FOR 指令。
- 如果使用 FOR 指令调用主例程，然后在主例程中使用 RET 指令，将出现严重错误 （类型 4，代码 31）。

启用时， FOR 指令重复执行例程，直到 Index （索引）值超过 Terminal value （终止值）。此指令不向例程传递参数。

每次 FOR 指令执行例程时，都会将 Step size （步进大小）添加到 Index （索引）中。

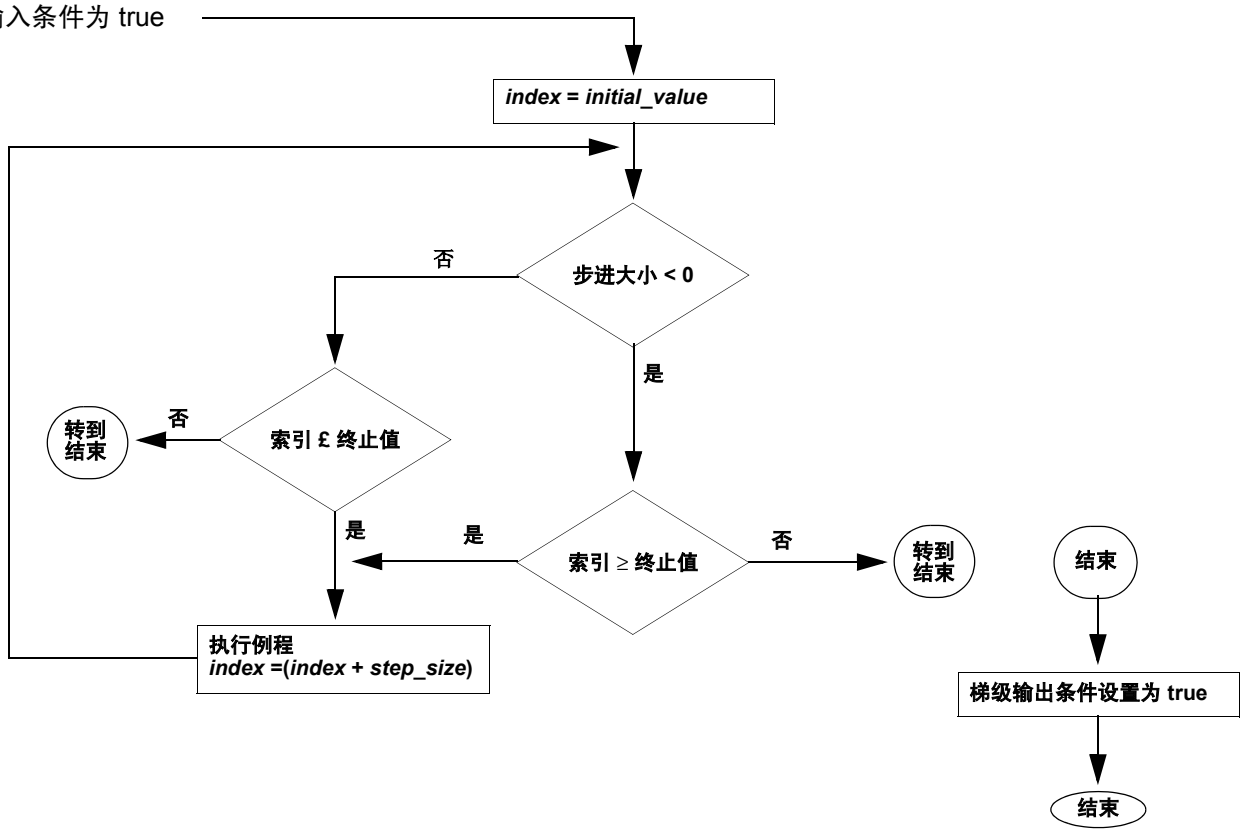
请注意，单次扫描中的循环次数不能过多。重复次数过多可能导致控制器的监视超时，从而导致严重错误。

算术状态标志： 不受影响

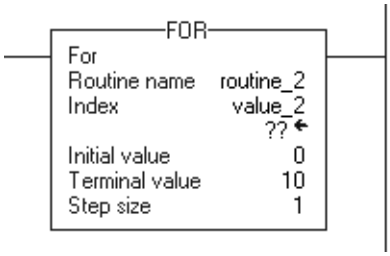
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
主例程包含一条 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 false。 控制器执行一次子例程。 如果对于同一子例程存在递归的 FOR 指令，则仅在第一次扫描时预扫描该子例程。如果对于同一子例程存在多条 FOR 指令 （非递归），则每次扫描时都会预扫描该子例程。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	
后扫描	梯级输出条件设置为 false。

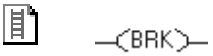
示例： 启用时， FOR 指令重复执行 *routine_2*，并且每次将 *value_2* 加 1。当 *value_2* > 10 或启用 BRK 指令时， FOR 指令不再执行 *routine_2*。



终止 (Break, BRK)

BRK 指令中断被 FOR 指令调用的例程的执行。

操作数：



梯形图

无



结构化文本

在循环结构中使用 EXIT 语句。有关结构化文本结构的信息，请参见结构化文本编程。

说明： 启用时， BRK 指令退出例程并将控制返回给 FOR 后面的指令。

如果存在嵌套的 FOR 指令， BRK 指令将控制返回给最内层的 FOR 指令。

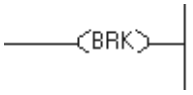
算术状态标志： 不受影响

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	梯级输出条件设置为 true。 执行返回到调用 FOR 指令的下一条指令。
后扫描	梯级输出条件设置为 false。

示例： 启用时， BRK 指令停止执行当前例程并返回到调用 FOR 指令的下一条指令。

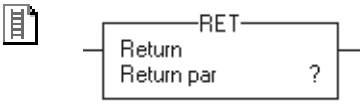


返回 (Return, RET)

RET 指令返回调用 FOR 指令。

操作数：

梯形图



无

说明：

重要事项 不要在主例程中使用 RET 指令。如果在主例程中使用 RET 指令，将出现严重错误 （类型 4，代码 31）。

启用时， RET 指令返回到 FOR 指令。 FOR 指令以 Step size （步进大小）为单位递增索引值，并再次执行子例程。如果 Index （索引）值超过 Terminal value （终止值），则 FOR 指令完成，执行转移到 FOR 指令的下一条指令。

FOR 指令不使用参数。 FOR 指令会忽略您在 RET 指令中输入的所有参数。

也可以使用 TND 指令来结束子例程的执行。

算术状态标志： 不受影响

错误条件：

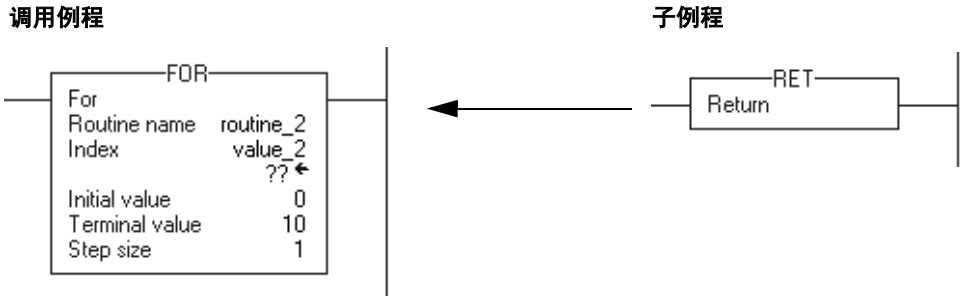
出现严重错误的条件：	错误类型：	错误代码：
主例程包含一条 RET 指令	4	31

执行：

条件：	梯形图操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	将指定参数返回到调用例程。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

示例： FOR 指令重复执行 *routine_2*，并每次将 *value_2* 加 1。当 *value_2* > 10 或启用 BRK 指令时， FOR 指令不再执行 *routine_2*。

RET 指令返回调用 FOR 指令。 FOR 指令要么再次执行子例程并以 Step size （步进大小）为单位递增索引值，要么在 Index （索引）值超过 Terminal value （终止值）时完成，并且将执行转移到 FOR 指令后面的指令。



说明：

第 12 章

特殊指令
（FBC、 DDT、 DTR、 PID）

简介

特殊指令执行应用特定的操作。

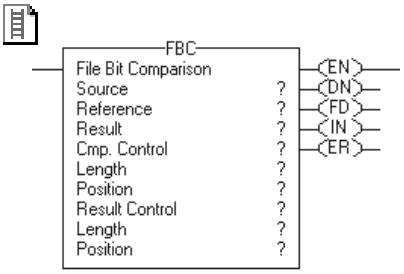
如果需要：	请使用指令：	在以下语言中可用：	请参见页：
与一已知、完整的参考数据相比较，并记录所有不匹配项。	FBC	梯形图	第 12 章 -2
与一已知、完整的参考数据相比较，记录所有不匹配项，并更新参考数据，使其与源匹配。	DDT	梯形图	第 12 章 -10
通过掩码传递源数据，并将结果与参考数据相比较。然后将源写入参考数据，以供下一次比较。	DTR	梯形图	第 12 章 -18
控制 PID 循环。	PID	梯形图 结构化文本	第 12 章 -21

文件位比较 (File Bit Comparison, FBC)

FBC 指令将源数组中的位与参考数组中的位相比较。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source（源）	DINT	数组标记	与参考比较的数组 **不要**在下标中使用 CONTROL.POS
Reference（参考）	DINT	数组标记	与源比较的数组 **不要**在下标中使用 CONTROL.POS
结果	DINT	数组标记	要存储结果的数组 **不要**在下标中使用 CONTROL.POS
Cmp control	CONTROL	结构	用于比较的控制结构
Length	DINT	立即数	要比较的位数
Position	DINT	立即数	源中的当前位置 初始值通常为 0
Result control	CONTROL	结构	结果的控制结构
Length	DINT	立即数	结果中的存储位置数
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记。对二者使用相同的标记可能会导致不可预料的操作、设备损坏和 / 或人员伤害。

COMPARE 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位表示已启用 FBC 指令。
.DN	BOOL	当 FBC 指令比较 Source （源）数组和 Reference （参考）数组中的最后一位时，设置完成位。
.FD	BOOL	每当 FBC 指令记录一个不匹配项 （“每次一个不匹配项”操作）或记录了所有不匹配项 （“每次扫描所有不匹配项”操作）后，设置发现位。
.IN	BOOL	禁止位表示 FBC 搜索模式。 0 = “所有”模式 1 = “每次一个不匹配项”模式
.ER	BOOL	如果 compare .POS < 0， compare .LEN < 0， result .POS < 0 或 result .LEN < 0，则设置该错误位。指令停止执行，直到程序清除 .ER 位。
.LEN	DINT	该长度值标识要比较的位数。
.POS	DINT	该位置值标识当前位。

RESULT 结构

助记符：	数据类型：	说明：
.DN	BOOL	Result 数组已满时设置该完成位。
.LEN	DINT	该长度值标识 Result 数组中的存储位置数。
.POS	DINT	该位置值标识 Result 数组中的当前位置。

说明： 启用时， FBC 指令将 Source （源）数组中的位与 Reference （参考）数组中的位相比较。并在结果数组中记录每个不匹配项的位号。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

FBC 指令在邻近的内存上操作。某些情况下，该指令在搜索或写入时会越过数组而进入标记的其他成员。如果长度过大并且标记为用户自定义的数据类型，则会发生这种情况。

DDT 指令与 FBC 指令的不同之处在于：每当 DDT 指令找到不匹配项时，该指令会更改参考位，使其与源位匹配。 FBC 指令不更改参考位。

选择搜索模式

如果需要检测：	选择模式：
每次一个不匹配项	设置比较 CONTROL 结构中的 .IN 位。 每当梯级输入条件从 false 变为 true 时， FBC 指令都会搜索 Source （源）数组与 Reference （参考）数组之间的下一个不匹配项。找到不匹配项时，指令设置 .FD 位，记录该不匹配项的位置，并停止执行。
所有不匹配项	清除比较 CONTROL 结构中的 .IN 位。 每当梯级输入条件从 false 变为 true 时， FBC 指令都会搜索 Source （源）数组与 Reference （参考）数组之间的所有不匹配项。

算术状态标志： 不受影响

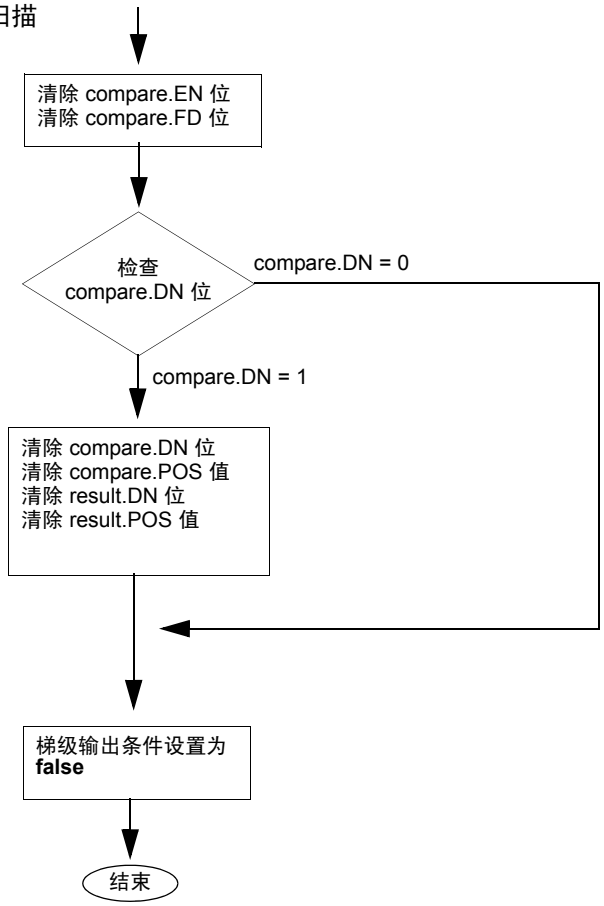
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
Result.POS > Result 数组的大小	4	20

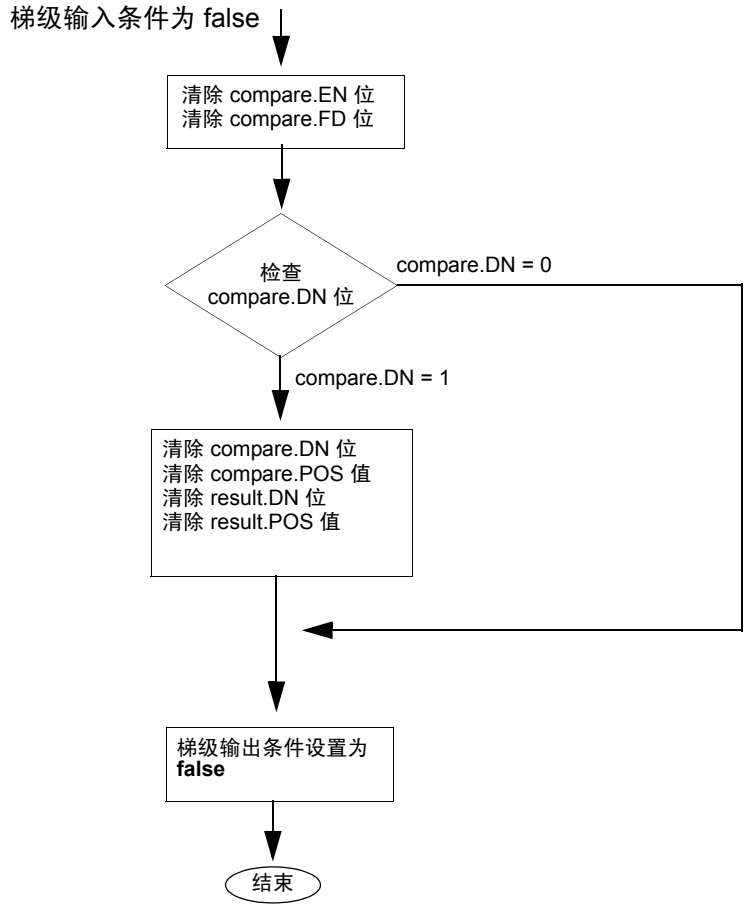
执行：

条件：	梯形图操作：

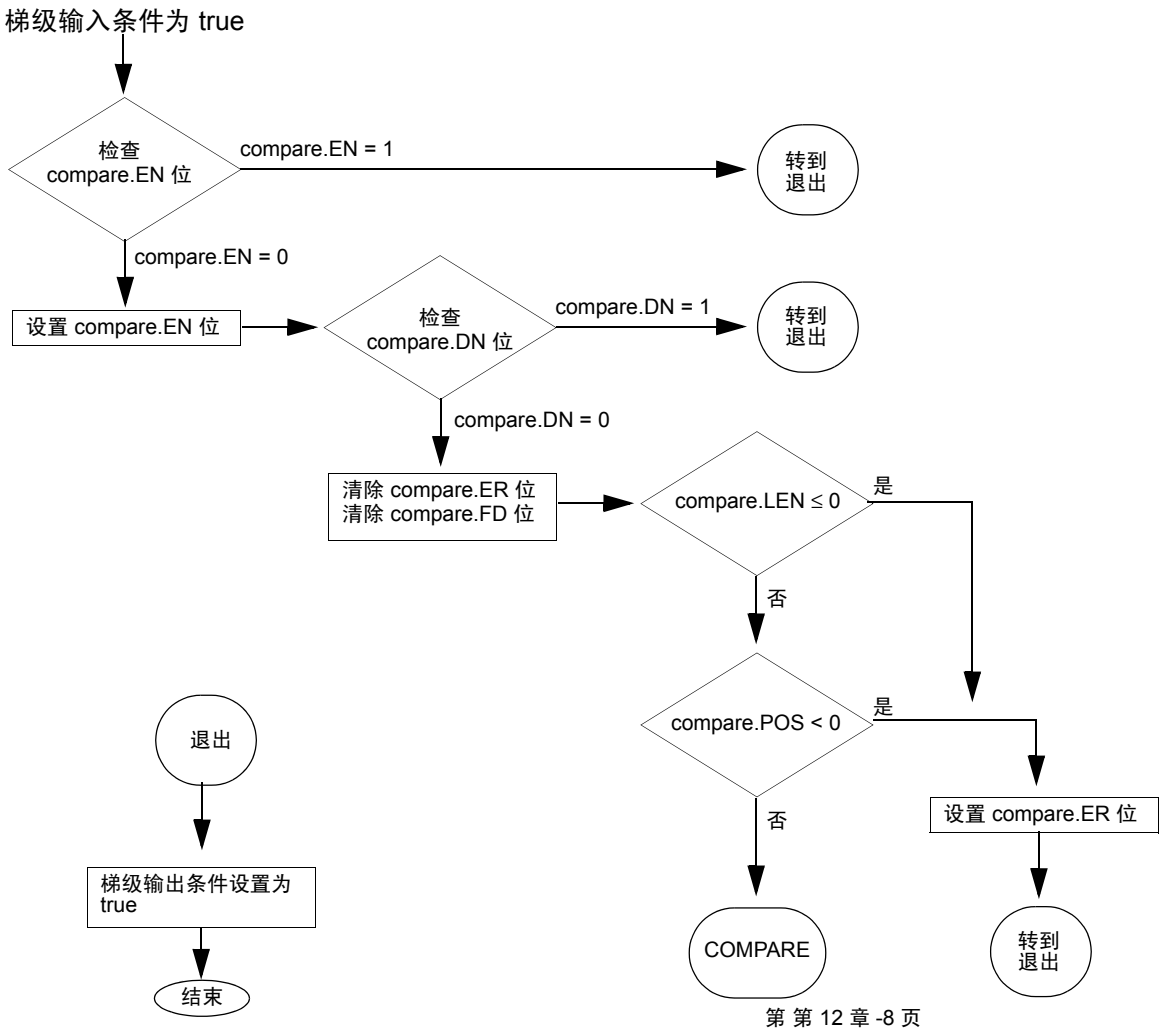
预扫描



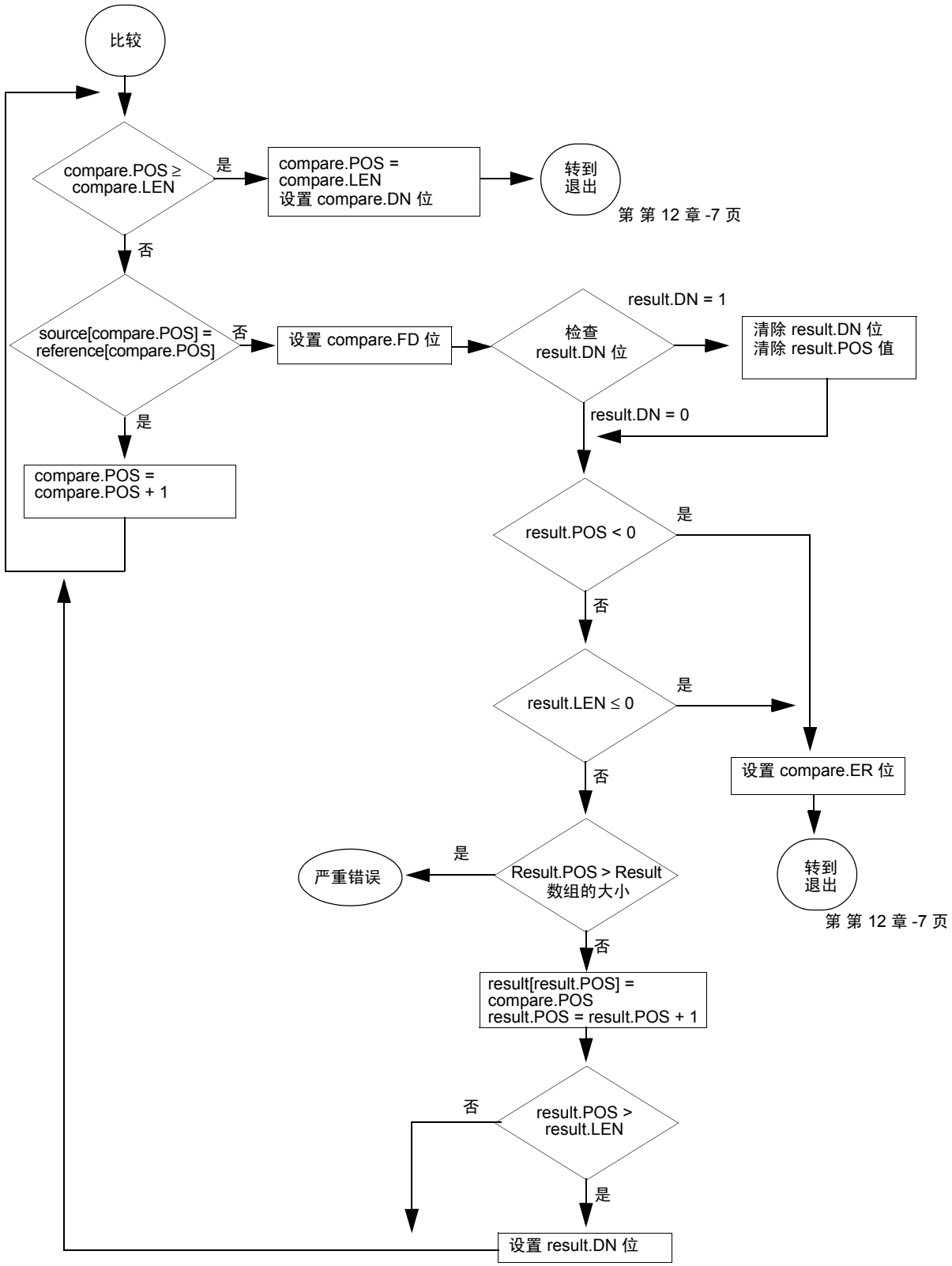
条件：	梯形图操作：


梯级输入条件为 false
清除 compare.EN 位
清除 compare.FD 位
检查
compare.DN 位
compare.DN = 0
compare.DN = 1
清除 compare.DN 位
清除 compare.POS 值
清除 result.DN 位
清除 result.POS 值
梯级输出条件设置为
false
结束

条件：	梯形图操作：

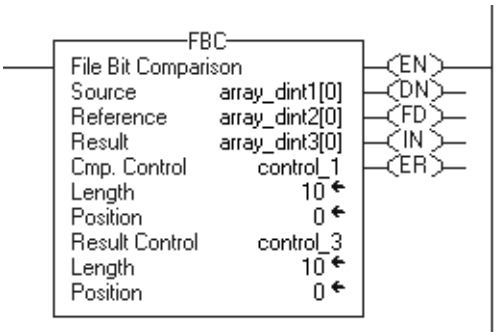

梯级输入条件为 true
检查 compare.EN 位
compare.EN = 1
转到 退出
compare.EN = 0
设置 compare.EN 位
检查 compare.DN 位
compare.DN = 1
转到 退出
compare.DN = 0
清除 compare.ER 位
清除 compare.FD 位
compare.LEN ≤ 0
是
否
compare.POS < 0
是
否
设置 compare.ER 位
退出
梯级输出条件设置为 true
结束
COMPARE
转到 退出
第 第 12 章 -8 页

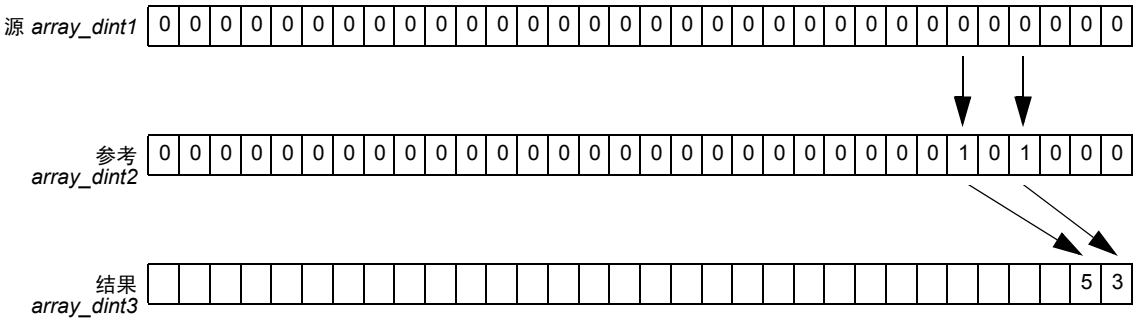
条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， FBC 指令将源 *array_dint1* 与参考 *array_dint2* 相比较，并在结果 *array_dint3* 中存储所有不匹配项的位置。



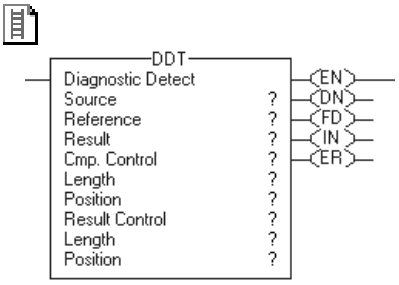


诊断检测 (Diagnostic Detect, DDT)

DDT 指令将 Source （源）数组中的位与 Reference （参考）数组中的位相比较，以确定状态的更改。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	DINT	数组标记	与参考比较的数组 **不要**在下标中使用 CONTROL.POS
Reference	DINT	数组标记	与源比较的数组 **不要**在下标中使用 CONTROL.POS
Result	DINT	数组标记	要存储结果的数组 **不要**在下标中使用 CONTROL.POS
Cmp control	CONTROL	结构	用于比较的控制结构
Length	DINT	立即数	要比较的位数
Position	DINT	立即数	源中的当前位置 初始值通常为 0
Result control	CONTROL	结构	结果的控制结构
Length	DINT	立即数	结果中的存储位置数
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记。对二者使用相同的标记可能会导致不可预料的操作、设备损坏和 / 或人员伤害。

COMPARE 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位表示已启用 DDT 指令。
.DN	BOOL	当 DDT 指令比较 Source （源）数组和 Reference （参考）数组中的最后一位时，设置完成位。
.FD	BOOL	每当 DDT 指令记录一个不匹配项 （“每次一个不匹配项”操作）或记录了所有不匹配项 （“每次扫描所有不匹配项”操作）后，设置发现位。
.IN	BOOL	禁止位表示 DDT 搜索模式。 0 = “所有”模式 1 = “每次一个不匹配项”模式
.ER	BOOL	如果 compare .POS < 0， compare .LEN < 0， result .POS < 0 或 result .LEN < 0，则设置该错误位。指令停止执行，直到程序清除 .ER 位。
.LEN	DINT	该长度值标识要比较的位数。
.POS	DINT	该位置值标识当前位。

RESULT 结构

助记符：	数据类型：	说明：
.DN	BOOL	Result 数组已满时设置该完成位。
.LEN	DINT	该长度值标识 Result 数组中的存储位置数。
.POS	DINT	该位置值标识 Result 数组中的当前位置。

说明： 启用时， DDT 指令将 Source （源）数组中的位与 Reference （参考）数组中的位相比较，在结果数组中记录每个不匹配项的位号，并更改 Reference （参考）位的值，使其与相应 Source （源）位的值匹配。

重要事项

您 **《必须》**测试并确认指令不会更改您不希望更改的数据。

DDT 指令在邻近的内存上操作。某些情况下，该指令在搜索或写入时会越过数组而进入标记的其他成员。如果长度过大并且标记为用户自定义的数据类型，则会发生这种情况。

DDT 指令与 FBC 指令的不同之处在于：每当 DDT 指令找到不匹配项时， DDT 指令都会更改参考位，使其与源位匹配。 FBC 指令不更改参考位。

选择搜索模式

如果需要检测：	选择模式：
每次一个不匹配项	设置比较 CONTROL 结构中的 .IN 位。 每当梯级输入条件从 false 变为 true 时， DDT 指令都会搜索 Source （源）数组与 Reference （参考）数组之间的下一个不匹配项。找到不匹配项时，指令设置 .FD 位，记录该不匹配项的位置，并停止执行。
所有不匹配项	清除比较 CONTROL 结构中的 .IN 位。 每当梯级输入条件从 false 变为 true 时， DDT 指令都会搜索 Source （源）数组与 Reference （参考）数组之间的所有不匹配项。

算术状态标志： 不受影响

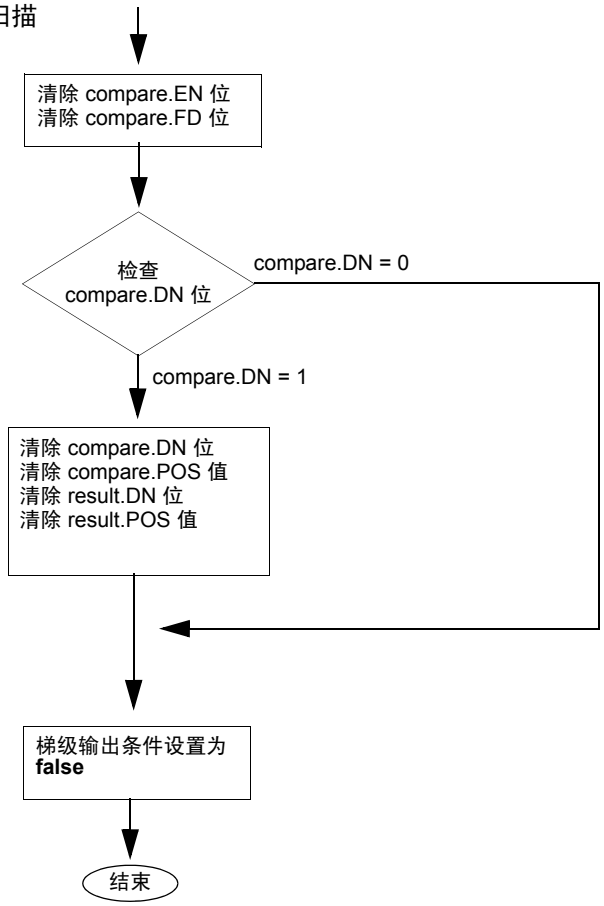
错误条件：

出现严重错误的条件：	错误类型：	错误代码：
Result.POS > Result 数组的大小	4	20

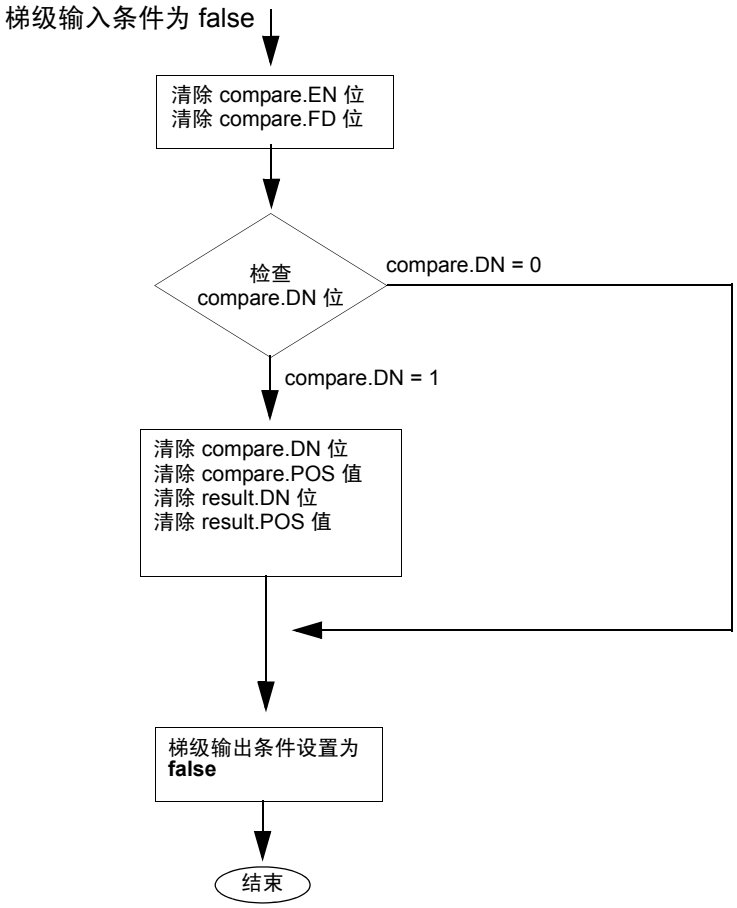
执行：

条件：	梯形图操作：

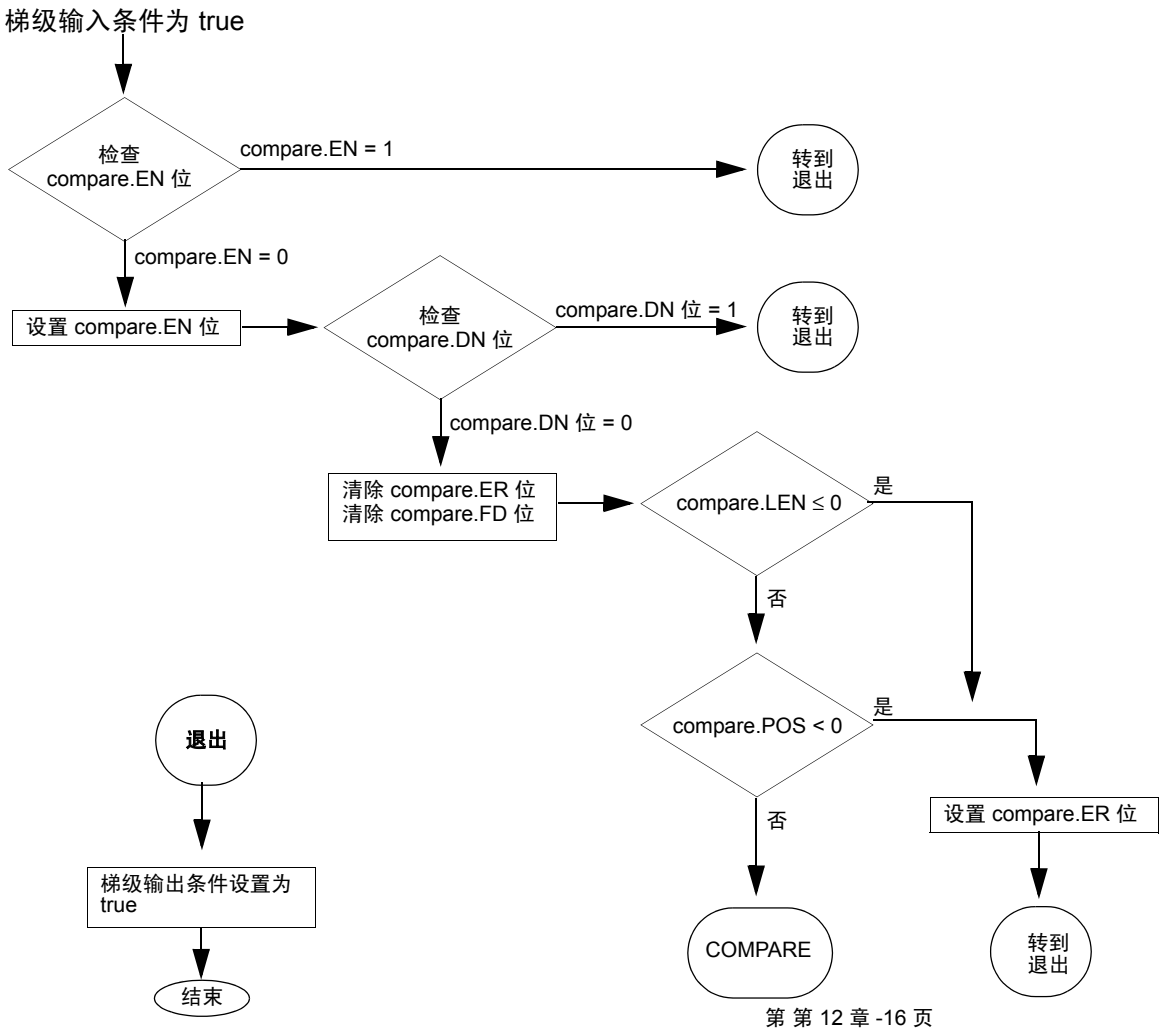
预扫描



条件：	梯形图操作：

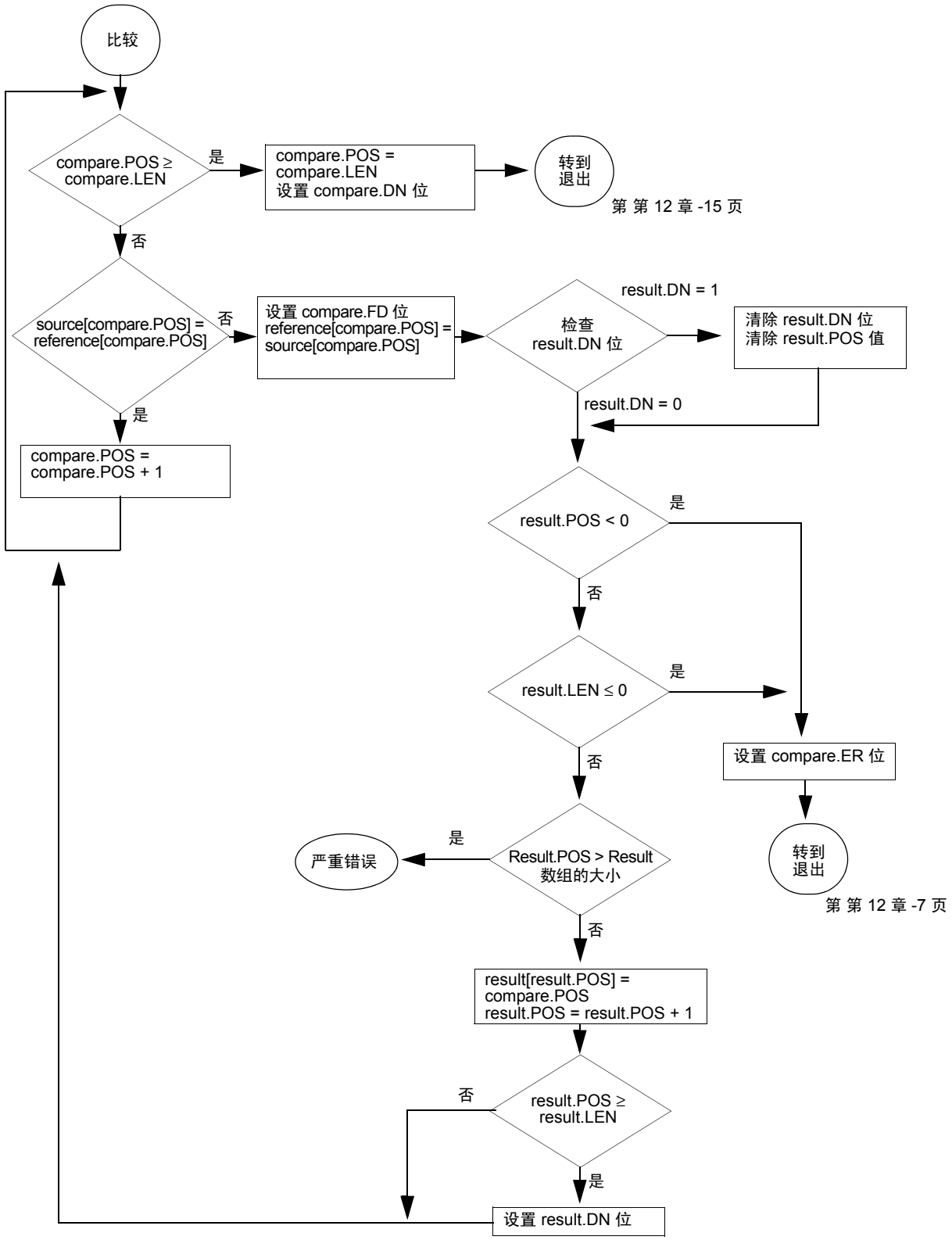

梯级输入条件为 false
清除 compare.EN 位
清除 compare.FD 位
检查
compare.DN 位
compare.DN = 0
compare.DN = 1
清除 compare.DN 位
清除 compare.POS 值
清除 result.DN 位
清除 result.POS 值
梯级输出条件设置为
false
结束

条件：	**梯形图操作：**



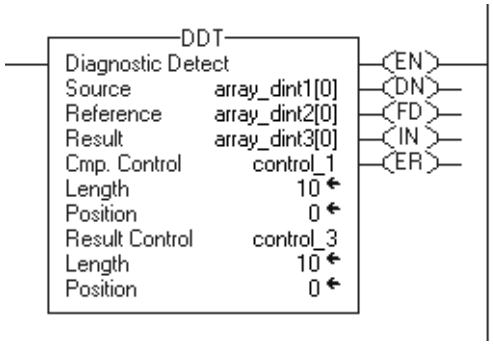
第 第 12 章 -16 页

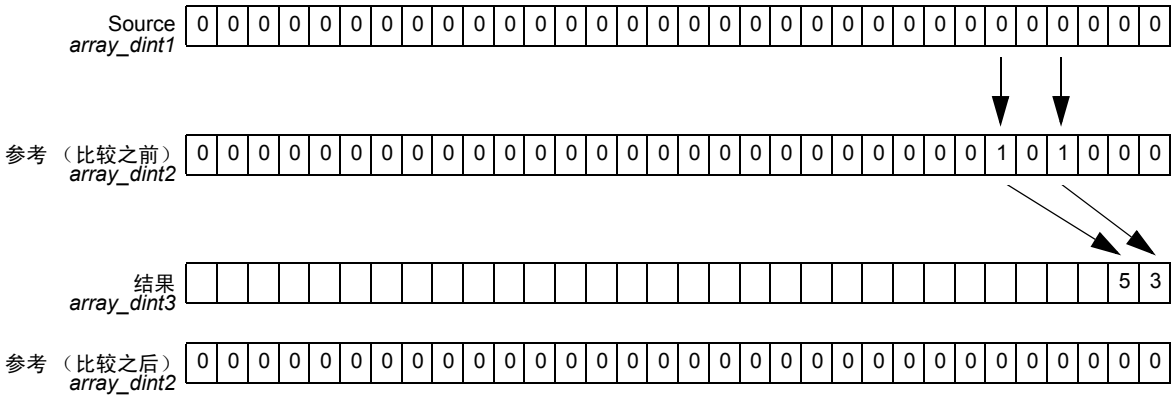
条件：	梯形图操作：



后扫描	梯级输出条件设置为 false。

示例： 启用时， DDT 指令将源 *array_dint1* 与参考 *array_dint2* 相比较，并在结果 *array_dint3* 中存储所有不匹配项的位置。控制器还会更改参考 *array_dint2* 中的不匹配位，使其与 *array_dint1* 匹配。



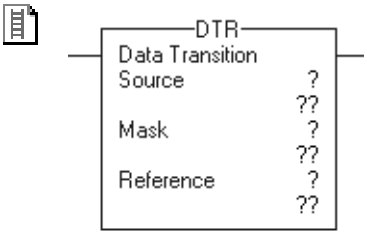


数据传送 (Data Transitional, DTR)

DTR 指令通过一个 Mask 传递 Source 值，并将结果与 Reference 值进行比较。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	DINT	立即数 标记	与参考比较的数组
Mask	DINT	立即数 标记	要屏蔽或传递的位
Reference	DINT	标记	与源比较的数组

说明： DTR 指令通过一个 Mask 传递 Source 值，并将结果与 Reference 值进行比较。 DTR 指令还将掩码 Source （源）值写入 Reference （参考）值，以供下一次比较。 Source 保持不变。

掩码中的 “1” 表示传递该数据位。掩码中的 “0” 表示屏蔽该数据位。

当掩码 Source （源）与 Reference （参考）不同时，梯级输出条件变为 true，以便进行一次扫描。当掩码 Source （源）与 Reference （参考）相同时，梯级输出条件为 false。

注意



使用此指令进行联机编程可能存在危险。如果 Reference （参考）值与 Source （源）值不同，则梯级输出条件变为 true。如果在处理器处于 “运行” 或 “远程运行” 模式时插入此指令，要非常谨慎。

输入立即掩码值

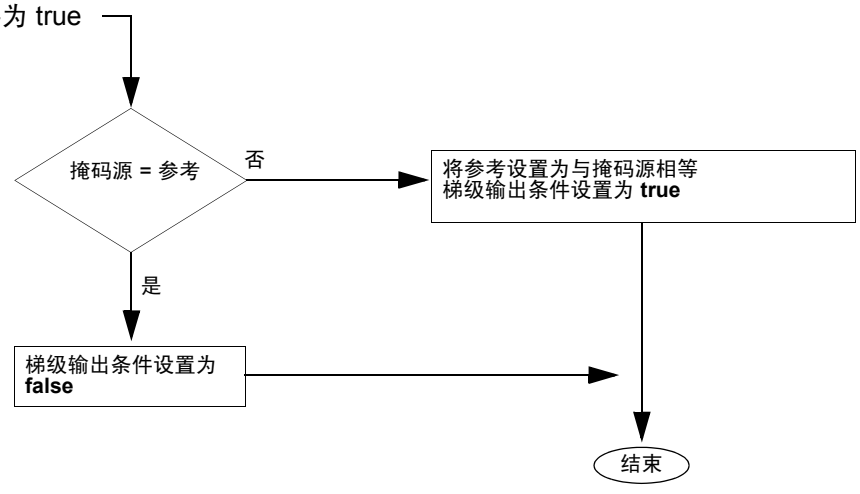
输入掩码时，编程软件默认为十进制值。如果要使用另一种格式输入掩码，请在该值前放置正确的前缀。

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

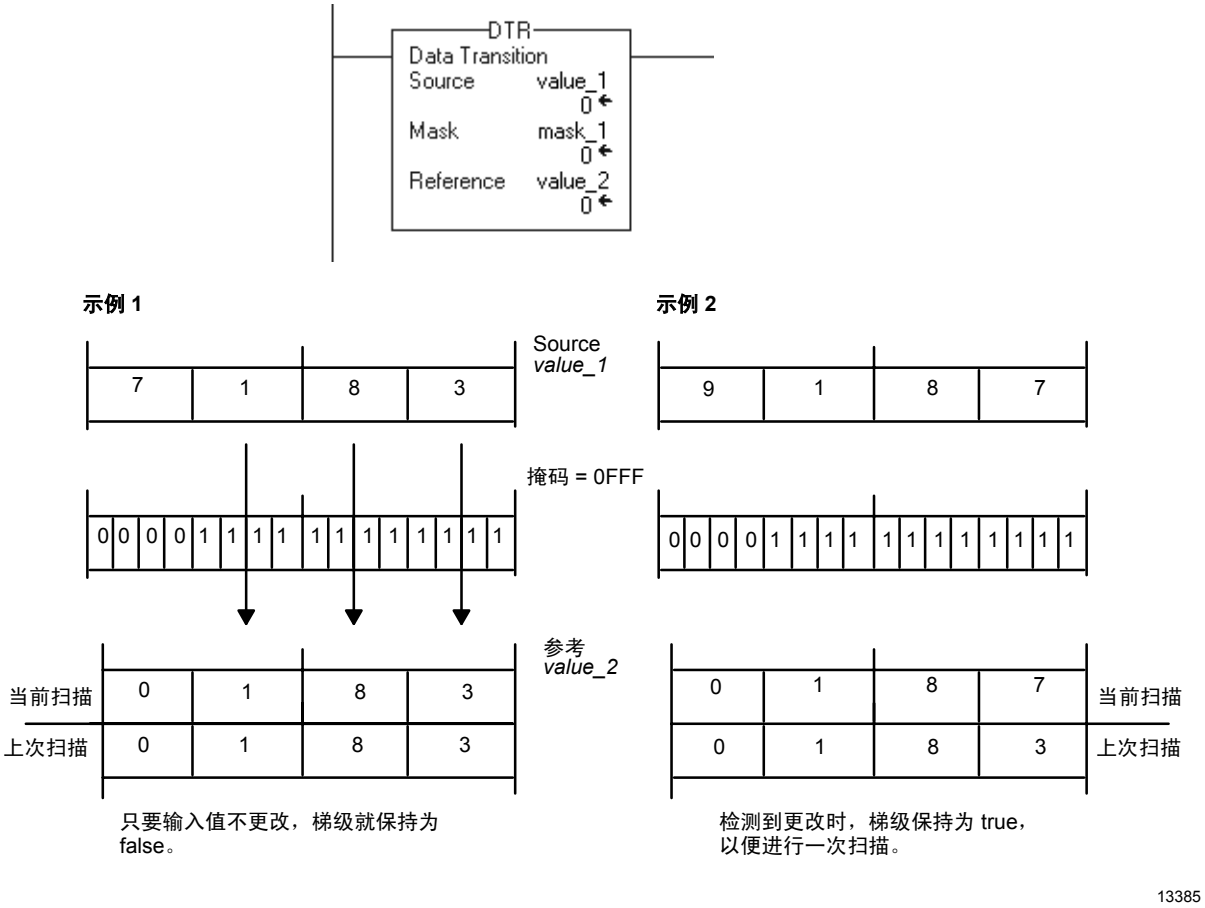
算术状态标志： 不受影响

错误条件： 无

执行：

条件：	梯形图操作：
预扫描	Reference = Source AND Mask。 梯级输出条件设置为 false。
梯级输入条件为 false	Reference = Source AND Mask。 梯级输出条件设置为 false。
梯级输入条件为 true	
后扫描	梯级输出条件设置为 false。

示例： 启用时， DTR 指令屏蔽 *value_1*。如果两个值之间有差异，则梯级输出条件被设置为 true。



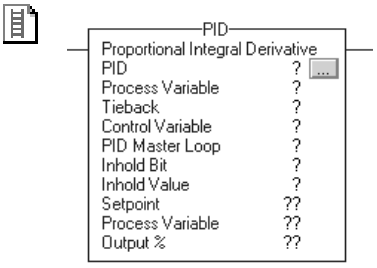
掩码中的 0 使该位保持不变。

比例积分微分 (Proportional Integral Derivative, PID)

PID 指令控制过程量，如流、压力、温度或水平。

操作数：

梯形图



操作数：	类型：	格式：	说明：
PID	PID	结构	PID 结构
过程量	SINT INT DINT REAL	标记	要控制的值
牵引	SINT INT DINT REAL	立即数 标记	（可选）硬件手动 / 自动站点的输出，它绕过控制器的输出 如果不希望使用此参数，请输入 0。
控制量	SINT INT DINT REAL	标记	定位到最终控制设备 （阀、气闸等）的值 如果使用死区控制，控制量必须为 REAL，否则，当误差在死区范围内时，该值将被强制为 0。
PID 主循环	PID	结构	（可选）主 PID 的 PID 标记 如果执行级联控制，且此 PID 为从属循环，请输入主 PID 的名称。如果不希望使用此参数，请输入 0。
保持位	BOOL	标记	（可选）来自 1756 模拟量输出信道的保持位的当前状态，用于支持无冲击重新启动 如果不希望使用此参数，请输入 0。
保持值	SINT INT DINT REAL	标记	（可选）来自 1756 模拟量输出信道的数据读回值，用于支持无冲击重新启动 如果不希望使用此参数，请输入 0。
设置点			显示设置点的当前值
过程量			显示成比例过程量的当前值
输出百分比			显示当前输出百分比值

结构化文本

```
PID(PID,ProcessVariable,
Tieback,ControlVariable,
PIDMasterLoop,InholdBit,
InHoldValue);
```

操作数与梯形图 PID 指令的操作数相同。然而，可以通过访问 PID 结构的 .SP、 .PV 和 .OUT 成员指定设置点、过程量和输出百分比，而不必在操作数列表中提供这些值。

PID 结构

助记符：	数据类型：	说明：	
.CTL	DINT	.CTL 成员提供对一个 32 位字中状态成员（位）的访问。PID 指令设置位 07 -15。	
		位：	**为成员：**
		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：	**为 PID 指令设置的以下成员：**
		15	.INI
		14	.SPOR
		13	.OLL
		12	.OLH
		11	.EWD
		10	.DVNA
		09	.DVPA
		08	.PVLA
		07	.PVHA
.SP	REAL	设置点	
.KP	REAL	独立	比例增益 （无单位）
		非独立	控制器增益 （无单位）
.KI	REAL	独立	积分增益 （1/ 秒）
		非独立	复位时间 （分 / 循环）
.KD	REAL	独立	微分增益 （秒）
		非独立	预调时间 （分）
.BIAS	REAL	前馈或偏置百分比	
.MAXS	REAL	最大执行单位比例值	
.MINS	REAL	最小执行单位比例值	
.DB	REAL	死区执行单位	

助记符：	数据类型：	说明：
.SO	REAL	设置输出百分比
.MAXO	REAL	最大输出界限 （输出百分比）
.MINO	REAL	最小输出界限 （输出百分比）
.UPD	REAL	循环更新时间 （秒）
.PV	REAL	成比例的 PV 值
.ERR	REAL	成比例的误差值
.OUT	REAL	输出百分比
.PVH	REAL	过程量上限警报值
.PVL	REAL	过程量下限警报值
.DVP	REAL	正偏差警报界限
.DVN	REAL	负偏差警报界限
.PVDB	REAL	过程量警报死区
.DVDB	REAL	偏差警报死区
.MAXI	REAL	最大 PV 值 （不成比例的输入）
.MINI	REAL	最小 PV 值 （不成比例的输入）
.TIE	REAL	手动控制的牵引值
.MAXCV	REAL	最大 CV 值 （对应于 100%）
.MINCV	REAL	最小 CV 值 （对应于 0%）
.MINTIE	REAL	最小牵引值 （对应于 100%）
.MAXTIE	REAL	最大牵引值 （对应于 0%）

助记符：	数据类型：	说明：	
.DATA	REAL[17]	.DATA 成员存储：	
		元素：	**说明：**
		.DATA[0]	积分累加值
		.DATA[1]	微分平滑临时值
		.DATA[2]	前一次 .PV 值
		.DATA[3]	前一次 .ERR 值
		.DATA[4]	前一次有效 .SP 值
		.DATA[5]	百分比比例常数
		.DATA[6]	.PV 比例常数
		.DATA[7]	微分比例常数
		.DATA[8]	前一次 .KP 值
		.DATA[9]	前一次 .KI 值
		.DATA[10]	前一次 .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]	前一次 .CV 值
		.DATA[15]	.CV 的缩小比例常数
		.DATA[16]	牵引值的缩小比例常数
.EN	BOOL	已启用	
.CT	BOOL	级联类型 （0= 从属； 1= 主）	
.CL	BOOL	级联循环 （0= 否； 1= 是）	
.PVT	BOOL	过程量跟踪 （0= 否； 1= 是）	
.DOE	BOOL	微分 （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	禁用微分平滑 （0= 启用微分平滑滤波器； 1= 禁用微分平滑滤波器）	
.NOBC	BOOL	禁用偏置反向计算 （0= 启用偏置反向计算； 1= 禁用偏置反向计算）	
.NOZC	BOOL	禁用过零死区 （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	设置点超出范围 （0= 是； 1= 是）	

助记符：	数据类型：	说明：
.OLL	BOOL	CV 低于最小输出界限 （0= 否； 1= 是）
.OLH	BOOL	CV 高于最大输出界限 （0= 否； 1= 是）
.EWD	BOOL	误差在死区范围内 （0= 否； 1= 是）
.DVNA	BOOL	偏差下限警报 （0= 否； 1= 是）
.DVPA	BOOL	偏差上限警报 （0= 否； 1= 是）
.PVLA	BOOL	PV 下限警报 （0= 否； 1= 否）
.PVHA	BOOL	PV 上限警报 （0= 否； 1= 是）

说明： PID 指令通常接收来自模拟量输入模块的过程量 (PV)，并调节模拟量输出模块上的控制量输出 (CV)，以将过程量保持在所需设置点上。

.EN 位表示执行状态。当梯级输入条件从 false 转换为 true 时，将设置 .EN 位。当梯级输入条件变为 false 时，将清除 .EN 位。 PID 指令不使用 .DN 位。只要梯级输入条件为 true， PID 指令就会执行所有扫描。



算术状态标志： 不受影响

错误条件：

重要事项 这些错误在 PLC-5 控制器中是严重错误。

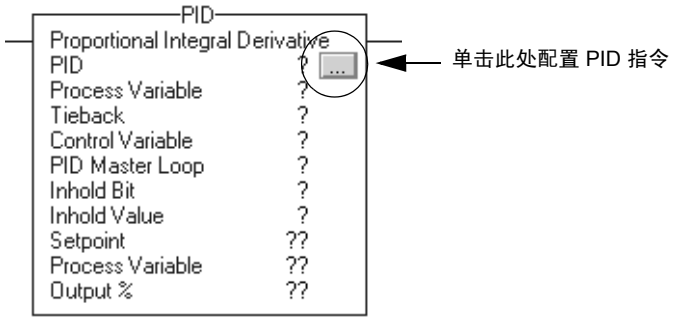
出现轻微错误的条件：	错误类型：	错误代码：
.UPD ≤ 0	4	35
设置点超出范围	4	36

执行：

条件：	操作：	操作：
预扫描	梯级输出条件设置为 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	指令执行 PID 循环。	指令执行 PID 循环。
后扫描	梯级输出条件设置为 false。	不采取任何操作。

配置 PID 指令

在输入 PID 指令并指定 PID 结构后，您就可以使用配置选项卡来指定 PID 指令如何工作。



指定调节

选择 Tuning （调节）选项卡。只要您单击另一个字段，单击 OK （确定），单击 Apply （应用），或按 Enter，更改就会立即生效。

在以下字段中：	指定：
Setpoint （设置点， SP）	输入设置点值 (.SP)。
Set output % （设置输出百分比）	输入设置输出百分比 (.SO)。 在软件手动模式中，此值用于输出。 在自动模式中，此值显示输出百分比。
Output bias （输出偏置）	输入输出偏置百分比 (.BIAS)。
Proportional gain （比例增益， K_p）	输入比例增益 (.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）。
Integral gain （积分增益， K_i）	输入积分增益 (.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）。
Derivative time （微分时间， K_d）	输入微分增益 (.KD)。 对于独立增益，它是微分增益 （秒）。 对于相关增益，它是预调时间 （分）。
Manual mode （手动模式）	选择手动 (.MO) 或软件手动 (.SWM)。 如果同时选择了二者，则手动模式优先于软件手动模式。

指定配置

选择 Configuration （配置）选项卡。必须单击 OK （确定）或 Apply （应用），所做更改才能生效。

在以下字段中：	指定：
PID equation （PID 等式）	选择独立增益或相关增益 (.PE)。 当您希望三个增益 （P、I 和 D）独立操作时，使用独立增益。当您需要一个影响所有这三项 （P、I 和 D）的总控制器增益时，使用相关增益。
Control action （控制操作）	为控制操作 (.CA) 选择 E=PV-SP 或 E=SP-PV。
Derivative of （微分）	选择 PV 或误差 (.DOE)。 使用 PV 的微分消除由设置点变化导致的输出尖峰。当算法可以容忍超调时，使用误差的微分以快速响应设置点的更改。
循环更新时间	输入指令的更新时间 (.UPD)。
CV high limit （CV 上限）	输入控制量的上限 (.MAXO)。

在以下字段中：	指定：
CV low limit （CV 下限）	输入控制量的下限 (.MINO)。
Deadband value （死区值）	输入死区值 (.DB)。
No derivative smoothing （禁用微分平滑）	启用或禁用此选择 (.NDF)。
No bias calculation （禁用偏置计算）	启用或禁用此选择 (.NOBC)。
No zero crossing in deadband （禁用过零死区）	启用或禁用此选择 (.NOZC)。
PV tracking （PV 跟踪）	启用或禁用此选择 (.PVT)。
Cascade loop （级联循环）	启用或禁用此选择 (.CL)。
Cascade type （级联类型）	如果启用级联循环，选择从属循环或主循环 (.CT)。

指定警报

选择 Alarms （警报）选项卡。必须单击 OK （确定）或 Apply （应用），所做更改才能生效。

在以下字段中：	指定：
PV high （PV 上限）	输入 PV 上限警报值 (.PVH)。
PV low （PV 下限）	输入 PV 下限警报值 (.PVL)。
PV deadband （PV 死区）	输入 PV 警报死区值 (.PVDB)。
positive deviation （正偏差）	输入正偏差值 (.DVP)。
negative deviation （负偏差）	输入负偏差值 (.DVN)。
deviation deadband （偏差死区）	输入偏差警报死区值 (.DVDB)。

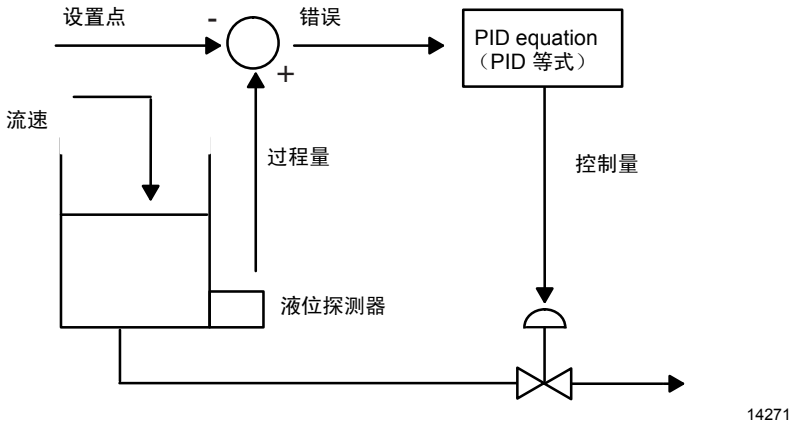
指定比例

选择 Scaling （比例）选项卡。必须单击 OK （确定）或 Apply （应用），所做更改才能生效。

在以下字段中：	指定：
PV unscaled maximum （PV 不成比例最大值）	输入最大 PV 值 (.MAXI)，它等于从 PV 值的模拟量输入信道接收的最大不成比例值。
PV unscaled minimum （PV 不成比例最小值）	输入最小 PV 值 (.MINI)，它等于从 PV 值的模拟量输入信道接收的最小不成比例值。
PV engineering units maximum （PV 执行单位最大值）	输入对应于 .MAXI 的执行单位最大值 (.MAXS)
PV engineering units minimum （PV 执行单位最小值）	输入对应于 .MINI 的执行单位最小值 (.MINS)
CV maximum （CV 最大值）	输入对应于 100% 的 CV 最大值 (.MAXCV)。
CV minimum （CV 最小值）	输入对应于 0% 的 CV 最小值 (.MINCV)。
Tieback maximum （最大牵引值）	输入最大牵引值 (.MAXTIE)，它等于从牵引值的模拟量输入信道接收的最大不成比例值。
Tieback minimum （最小牵引值）	输入最小牵引值 (.MINTIE)，它等于从牵引值的模拟量输入信道接收的最小不成比例值。
PID Initialized （PID 已初始化）	如果您在运行模式期间更改比例常数，则关闭此字段以重新初始化内部缩小比例的值 (.INI)。

使用 PID 指令

PID 闭环控制使过程量保持在所需设置点。下图显示了一个流速 / 液位示例：



在上述示例中，将槽中的液位与设置点相比较。如果液位高于设置点， PID 等式将增加控制量并使槽的出口阀打开；从而降低槽内液位。

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，比例增益被影响所有三项的控制器增益代替。您可以使用任何一种形式的等式来执行同类型的控制。所提供的两个等式类型仅用于让您使用最熟悉的等式类型。

增益选项：	微分：	等式：
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 （预调时间）之间转换，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	预调时间 （分）
SP	设置点
PV	过程量
E	误差 [(SP-PV) 或 (PV-SP)]
BIAS	前馈或偏置
CV	控制量
dt	循环更新时间

如果不想使用 PID 等式的特定项，仅需将其增益设置为零。例如，如果不想使用微分操作，则将 K_d 或 T_d 设置为等于零。

防止积分饱和及由手动向自动的无冲击转换

PID 指令通过防止在 CV 输出达到由 .MAXO 和 .MINO 设置的其最大值或最小值时积分项进行累加，来自动避免积分饱和。累加的积分项保持不变，直到 CV 输出降至其最大上限之下或升至其最小下限之上。然后正常的积分累加会自动重新开始。

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) 值被用作循环的输出。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)	将牵引值作为输入，并调整其内部变量，以便在输出时生成相同的值 根据 .MINTIE 和 .MAXTIE 的值，将 PID 指令的牵引输入的比例定为 0-100%，并将其作为循环的输出。牵引输入通常来自硬件手动 / 自动站点的输出，该硬件手动 / 自动站点绕过控制器的输出。 **说明：**如果两种模式位都被设置为打开，手动模式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冲击转换。 PID 指令反向计算积分累加项的值，该值是使 CV 输出跟踪软件手动模式中的设置输出 (.SO) 值和手动模式中的牵引输入所必需的。在这种情况下，当循环切换到自动模式时， CV 输出从设置输出或牵引值开始，因此在输出值中不会出现 “冲击”。

即使没有使用积分控制 （即， $K_i = 0$）， PID 指令也可以自动提供从手动模式到自动模式的无冲击转换。在这种情况下，该指令修改 .BIAS 项以使 CV 输出跟踪设置输出或牵引值。恢复自动控制后， .BIAS 项将保持其最后的值。您可以通过设置 PID 数据结构中的 .NOBC 位来禁用 .BIAS 项的反向计算。请注意，如果将 .NOBC 设置为 true，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冲击转换。

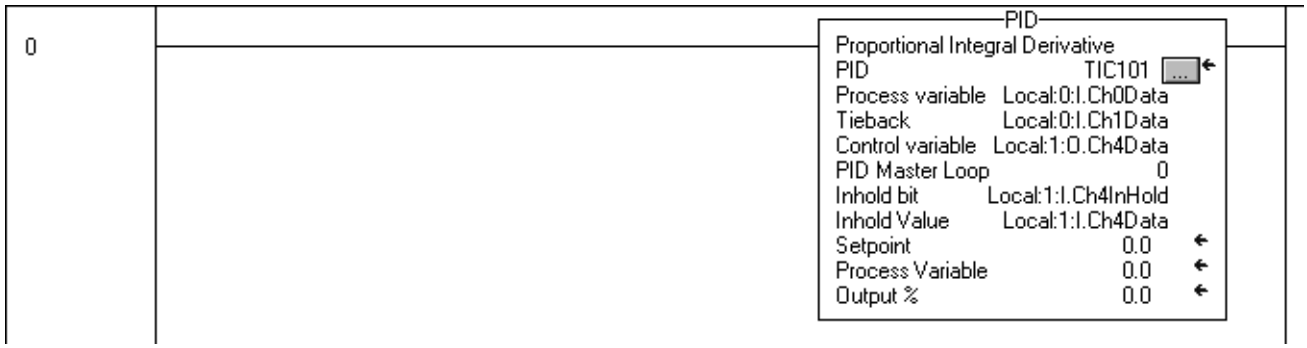
PID 指令计时

需要对 PID 指令和过程量的采样定期进行更新。更新时间与您所控制的物理过程有关。对于非常缓慢的循环 （如温度循环），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。某些稍快的循环 （如压力或流循环）则可能需要像 250 毫秒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 10 毫秒一次或更快的循环更新时间。

由于 PID 指令在其计算中使用时间基，因此您需要将此指令的执行与过程量 (PV) 采样同步。

执行 PID 指令最简单的方式是将 PID 指令用在周期任务中。将循环更新时间 (.UPD) 设置为等于周期任务速率，并确保每次扫描时周期任务都执行 PID 指令

梯形图



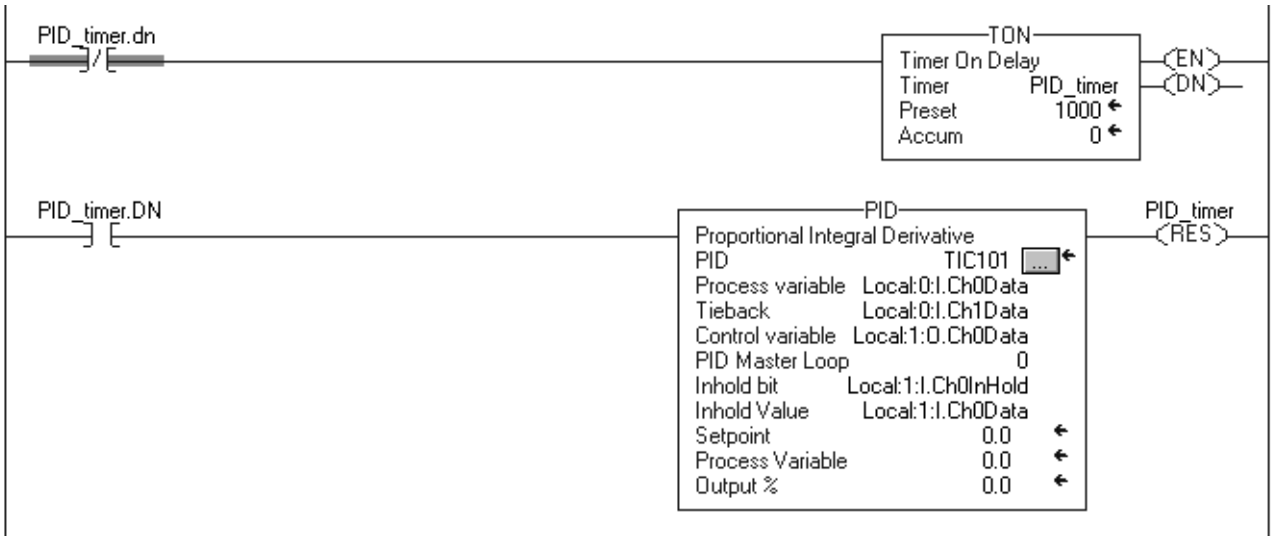
结构化文本

```
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，应确保以比周期任务速率快得多的速率将用于过程量的模拟量输入更新到处理器。理想情况下，至少应以比周期任务数率快 5 - 10 倍的速率将过程量发送到处理器。这可使过程量的实际采样和 PID 循环的执行的时间差最小。例如，如果 PID 循环位于一个 250 毫秒的周期任务中，使用 250 毫秒的循环更新时间 (.UPD = .25)，并将模拟量输入模块配置为大约至少每 25 - 50 毫秒产生一次数据。

执行 PID 指令的另一种方法是将指令放在连续任务中，并使用计时器完成位来触发 PID 指令的执行，但精度稍差。

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 指令的循环更新时间设置位等于计时器预设值。这与使用周期任务的情况相同，您应将模拟量输入模块产生过程量的速率设置为比循环更新时间快得多。只有循环的循环更新时间至少比连续任务的最差执行时间长若干倍时，才能使用 PID 执行的计时器方法。

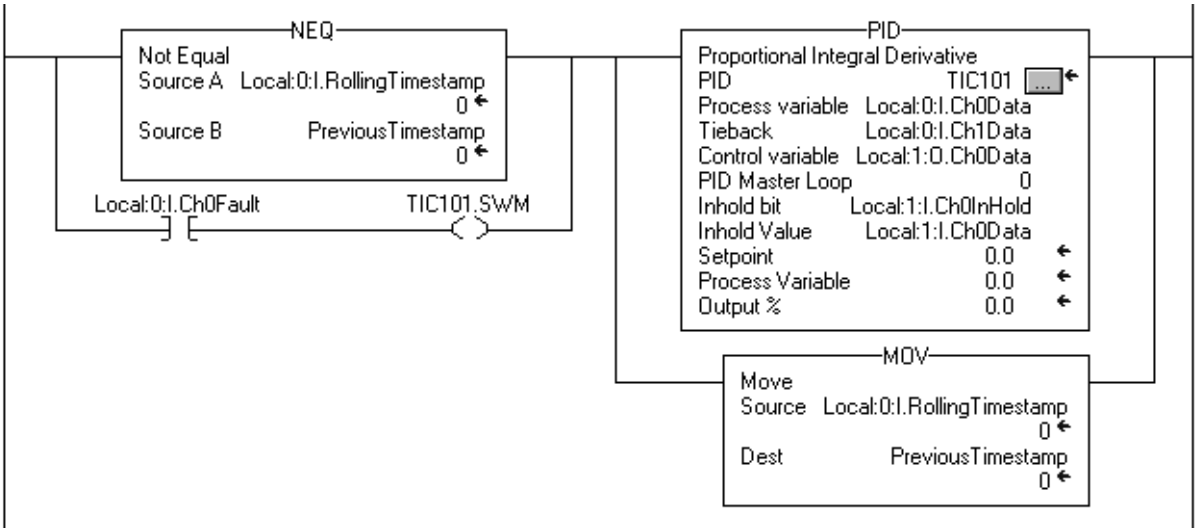
执行 PID 指令最准确的方法是使用 1756 模拟量输入模块的实时采样 (RTS) 功能。模拟量输入模块以您在设置模块时配置的实时采样速率对其输入进行采样。当模块的实时采样周期结束时，模块会更新其输入并更新由模块产生的滚动时间戳 （由模拟量输入数据结构的 .RollingTimestamp 成员表示）。

时间戳范围为 0-32767 毫秒。监控时间戳。当时间戳发生变化时，已收到新的过程量采样。每次时间戳变化时，都会执行一次 PID 指令。由于过程量采样是由模拟量输入模块驱动的，因此输入采样时间非常准确，应将 PID 指令使用的循环更新时间设置为等于模拟量输入模块的 RTS 时间。

为了确保不错过过程量的采样，应以比 RTS 时间更快的速率执行逻辑。例如，如果 RTS 时间为 250 毫秒，则您可以将 PID 逻辑放在每 100 毫秒运行一次的周期任务中，以确保不错过任何采样。只要能够确保以比每 250 毫秒一次更快的频率更新逻辑，您甚至可以将 PID 逻辑放在连续任务中。

下面显示了一个 RTS 执行方法的示例。 PID 指令的执行取决于新模拟量输入数据的接收。如果模拟量输入模块失败或被删除，控制器将停止接收滚动时间戳， PID 循环也会停止执行。应监控 PV 模拟量输入的状态位，如果它显示错误状态，则强制循环进入软件手动模式，并在每次扫描时都执行该循环。这样，操作员仍然可以手动更改 PID 循环的输出。

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;
```

无冲击重新启动

PID 指令可以与 1756 模拟量输出模块进行交互，以便当控制器从编程模式更改为运行模式或控制器加电时，支持无冲击重新启动。

当 1756 模拟量输出模块与控制器失去通讯或检测到控制器处于编程模式时，模拟量输出模块都会将其输出设置为您配置模块时指定的错误条件值。当控制器随后返回到运行模式或与模拟量输出模块重新建立通讯时，您可以通过使用 PID 指令上的保持位和保持值参数，使 PID 指令将其控制量输出自动复位为等于模拟量输出。

设置无冲击重新启动：

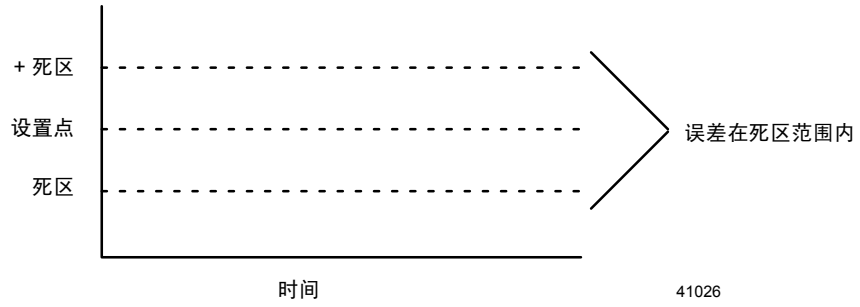
执行如下操作：	详细信息：
配置 1756 模拟量输出模块的信道，它从 PID 指令接收控制量	在模块的特定信道的属性页上，选中 “hold for initialization （保持以进行初始化）”复选框。 该选项通知模拟量输出模块：当控制器返回到运行模式或与模块重新建立通讯时，模块应将模拟量输出保持为其当前值，直到控制器发送的值与输出信道使用的当前值相匹配 （在 0.1% 范围内）。控制器的输出通过使用 .BIAS 项，以一定的斜率过渡到当前保持的输出值。这种过渡方式类似于自动无冲击转换。
在 PID 指令中输入 “保持位”标记和 “保持值”标记	1756 模拟量输出模块为其输入数据结构中的每个信道返回两个值。当保持状态位 （例如 .Ch2InHold）为 true 时，表示模拟量输出信道正在保持其值。数据读回值 （例如 .Ch2Data）以执行单位显示当前输出值。 将保持状态位的标记作为 PID 指令的保持位参数输入。将数据读回值的标记作为保持值参数输入。 当保持位变为 true 时， PID 指令会将保持值移动到控制量输出，并进行重新初始化以支持从该值进行无冲击重新启动。当模拟量输出模块接收到从控制器返回的该值时，会关闭保持状态位，从而允许 PID 指令正常启动控制。

微分平滑

微分平滑滤波器增强了微分计算。这种一阶低通数字滤波器有助于最大限度减少由 PV 中噪音引起的较大微分项尖峰。这种平滑作用对于较大的微分增益值的影响更加明显。如果您的过程需要非常大的微分增益值 （例如 $K_d > 10$），可以禁用微分平滑。要禁用微分平滑，请选择 Configuration （配置）选项卡上的 “No derivative smoothing （禁用微分平滑）”选项，或设置 PID 结构中的 .NDF 位。

设置死区

使用可调整死区，您可以在设置点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，您可以控制过程量与设置点匹配的接近程度，而不改变输出。死区还有助于最大程度地减少最终控制设备的磨损。



过零是一种死区控制，当过程量进入死区时，过零允许指令将误差用于计算，直到过程量经过设置点。一旦过程量经过设置点 （误差过零且改变符号），只要过程量保持在死区中，输出就不会改变。

死区按您指定的值从设置点向上和向下扩展。输入零可禁止死区。死区与设置点具有相同的比例单位。您可以通过选择 Configuration （配置）选项卡上的 “no zero crossing for deadband （禁用过零死区）”选项，或在 PID 结构中设置 .NOZC 位，来使用不具有过零功能的死区。

如果要使用死区，控制量必须为 REAL，否则，当误差位于死区内时，它将被强制为 0。

使用输出限制

您可以为控制输出设置输出界限 （输出百分比）。当指令检测到输出已达到某个界限时，会设置警报位，并阻止输出超出更低界限或更高界限。

前馈或输出偏置

您可以通过将 .BIAS 值馈入 PID 指令的前馈 / 偏置值中，以前馈来自系统的干扰。

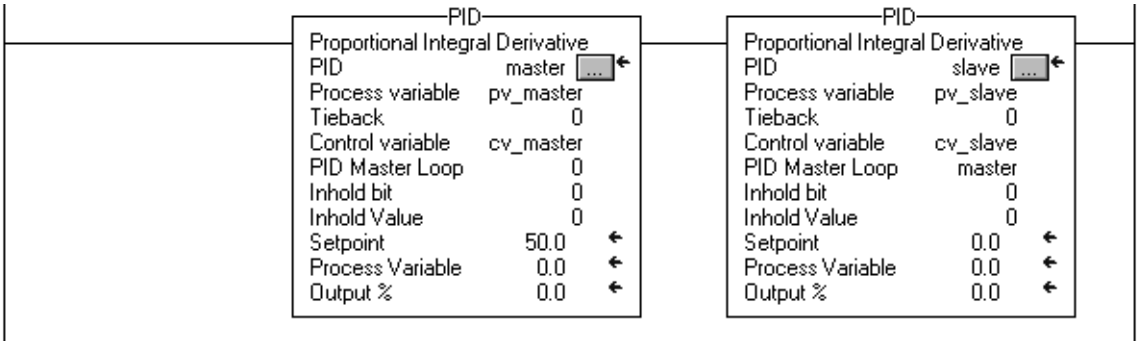
前馈值代表在干扰有机会更改过程量之前馈入 PID 指令的干扰。前馈通常用于控制具有运输滞后的过程。例如，代表 “注入温水混合物的冷水”的前馈值可能会比等待由混合结果引起的过程量变化更快地启动输出值。

不使用积分控制时，通常使用偏置值。在这种情况下，可通过调整偏置值来将输出保持在所需范围内，从而保持 PV 接近设置点。

级联循环

PID 通过将主循环的百分比输出赋值给从属循环的设置点，来级联两个循环。根据从属循环 .MAXS 和 .MINS 的值，从属循环自动将主循环的输出转换为从属循环设置点的正确的执行单位。

梯形图



结构化文本

```
PID(master,pv_master,0,cv_master,0,0,0);

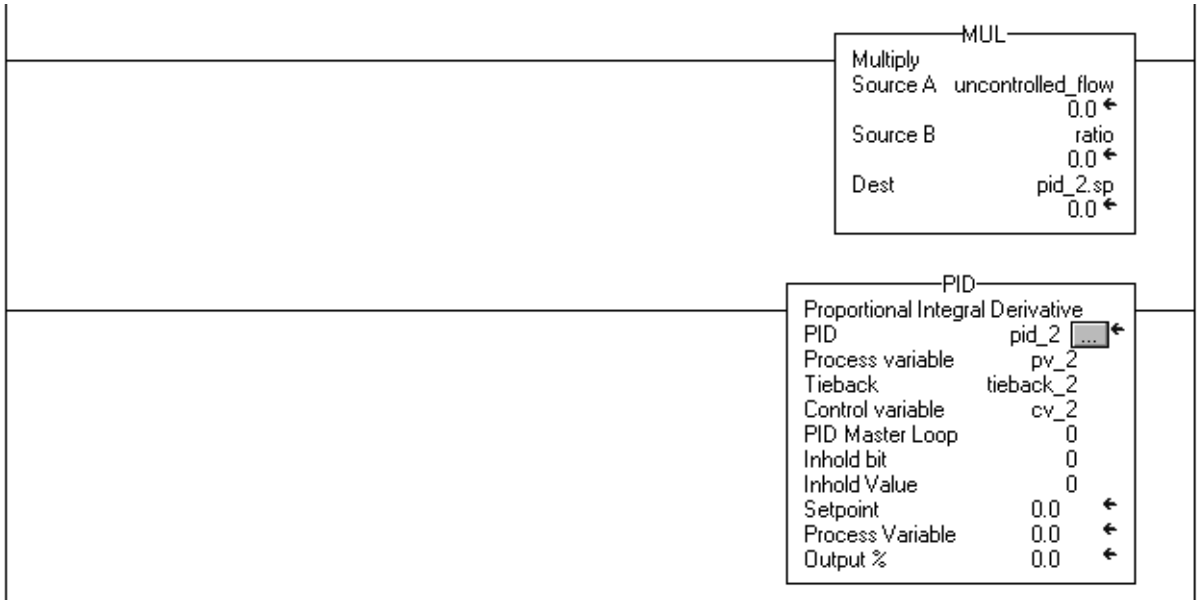
PID (slave,pv_slave,0,cv_slave,master,0,0);
```

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 指令要使用的结果设置点）
- 两个值之间的比率

梯形图



结构化文本

```
pid_2.sp := uncontrolled_flow * ratio

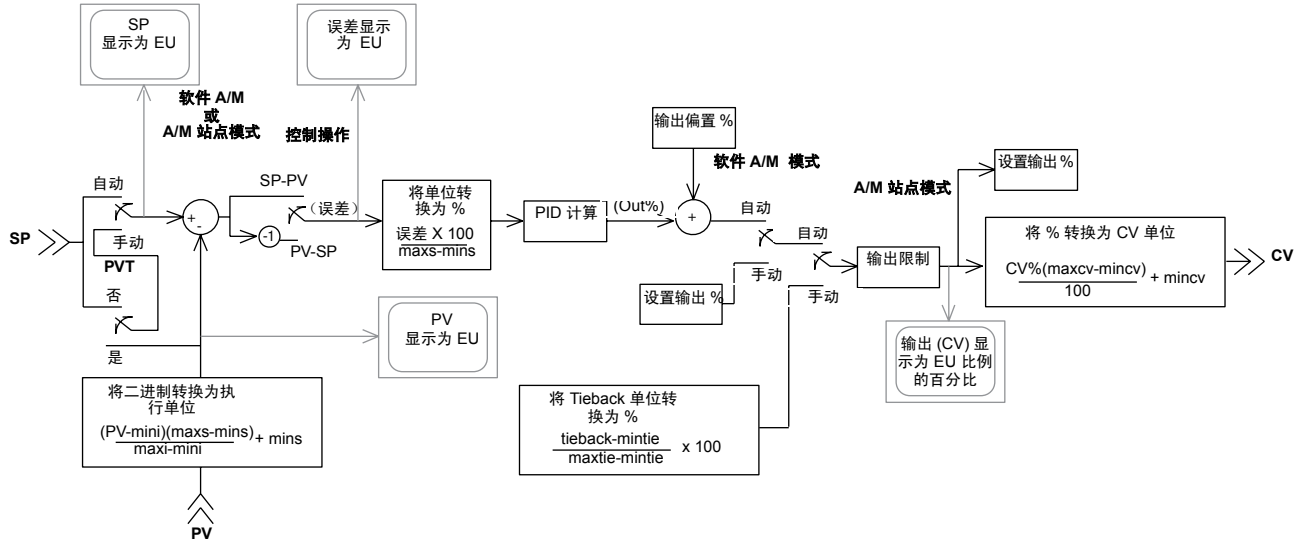
PID(pid_2,pv_2,tieback_2,cv_2,0,0,0);
```

对于如下乘法参数：	输入如下值：
Destination	控制值
Source A	未控制值
Source B	比率

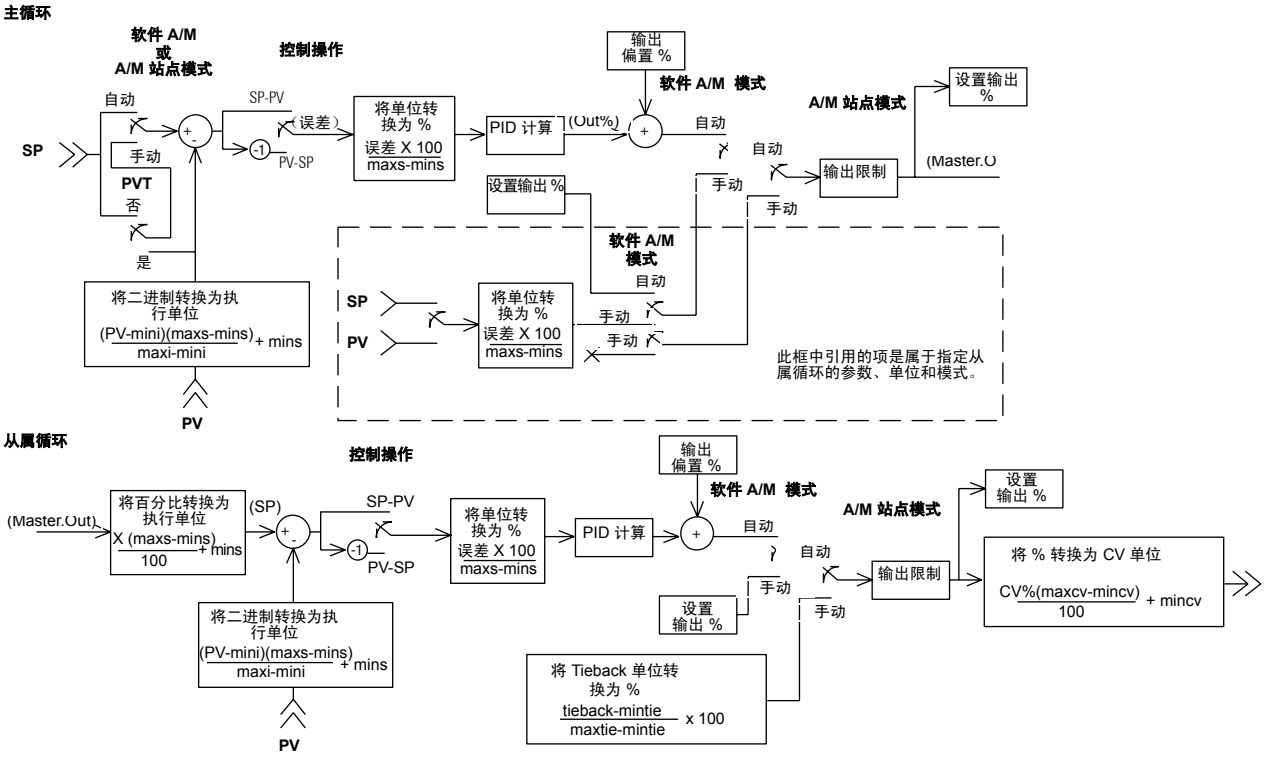
PID 原理

下图显示 PID 指令的过程流。

PID 过程



具有主 / 从属循环的 PID 过程



说明：

第 13 章

三角指令
（SIN、 COS、 TAN、 ASN、 ASIN、 ACS、 ACOS、 ATN、 ATAN）

简介

三角函数指令使用三角运算来计算算术运算。

操作目的：	操作指令：	适用语言：	参考页面：
计算值的正弦值。	SIN	梯形图 结构化文本 功能块	第 13 章 -2
计算值的余弦值。	COS	梯形图 结构化文本 功能块	第 13 章 -5
计算值的正切值。	TAN	梯形图 结构化文本 功能块	第 13 章 -8
计算值的反正弦值。	ASN ASIN(1)	梯形图 结构化文本 功能块	第 13 章 -11
计算值的反余弦值。	ACS ACOS(1)	梯形图 结构化文本 功能块	第 13 章 -14
计算值的反正切值。	ATN ATAN(1)	梯形图 结构化文本 功能块	第 13 章 -17

(1) 仅结构化文本。

您可以混用数据类型，但可能发生精度损失和舍入误差，并且指令需要更长的执行时间。检查溢出状态位 (S:V) 来查看结果是否被截断。

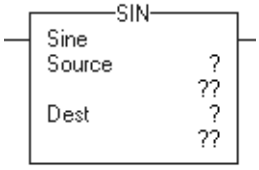
对于梯形图指令，**粗体**数据类型表示最佳数据类型。如果指令的所有操作数都使用同样的最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

正弦 (Sine, SIN)

SIN 指令计算 Source 值 （以弧度为单位）的正弦值并将结果存储在 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	查找该值的正弦值
Destination	SINT INT DINT **REAL**	标记	存储结果的标记

结构化文本

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

将 SIN 用作函数。该函数计算 *source* 的正弦值并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
SIN 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为设置状态。
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： Source 必须大于或等于 -205887.4 ($-2\pi x2^{15}$) 并且小于或等于 205887.4 ($2\pi x2^{15}$)。Destination 中的结果值总是大于或等于 -1 并且小于或等于 1。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source 的正弦值并将结果放入 Destination 中。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

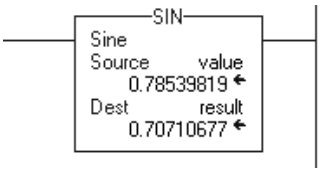


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于设置状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 计算 *value* 的正弦值并将结果放入 *result* 中。

梯形图



结构化文本

```
result := SIN(value);
```

功能块

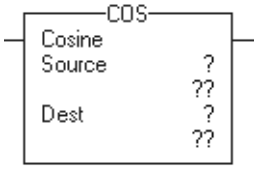


余弦 (Cosine, COS)

COS 指令计算 Source 值 （以弧度为单位）的余弦值并将结果存储在 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	查找该值的余弦值
Destination	SINT INT DINT **REAL**	标记	存储结果的标记

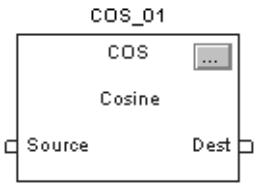
结构化文本

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

将 COS 用作函数。该函数计算 *source* 的余弦值并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
COS 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为设置状态。
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： Source 必须大于或等于 -205887.4 ($-2\pi x2^{15}$) 并且小于或等于 205887.4 ($2\pi x2^{15}$)。Destination 中的结果值总是大于或等于 -1 并且小于或等于 1。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source 的余弦值并将结果放入 Destination 中。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

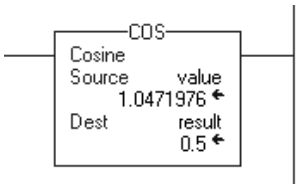


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于设置状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 计算 *value* 的余弦值并将结果放入 *result* 中。

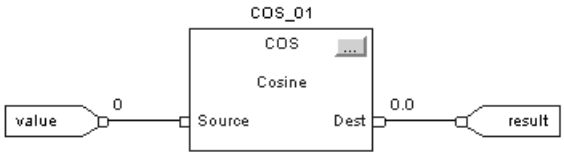
梯形图



结构化文本

```
result := COS(value);
```

功能块

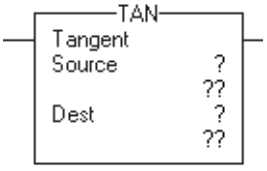


正切 (Tangent, TAN)

TAN 指令计算 Source 值 （以弧度为单位）的正切值并将结果存储在 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	查找该值的正切值
Destination	SINT INT DINT **REAL**	标记	存储结果的标记

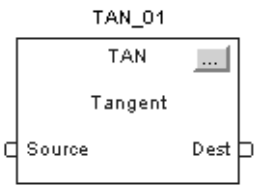
结构化文本

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

将 TAN 用作函数。该函数计算 *source* 的正切值并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
TAN 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为设置状态。
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： Source 必须大于或等于 -102943.7($-2\pi x2^{14}$)，小于或等于 102943.7 ($2\pi x2^{14}$)。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source 的正切值并将结果放入 Destination 中。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

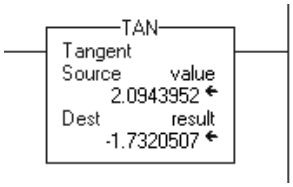


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于设置状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 计算 *value* 的正切值并将结果放入 *result* 中。

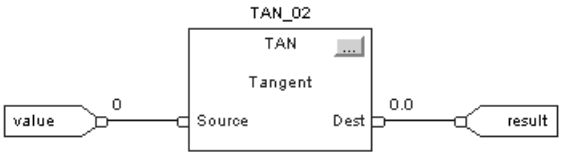
梯形图



结构化文本

```
result := TAN(value);
```

功能块

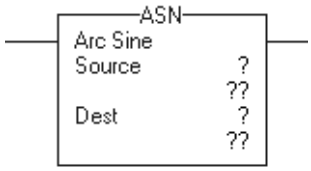


反正弦 (Arc Sine，ASN)

ASN 指令计算 Source 值的反正弦值并将结果存储在 Destination 中（以弧度为单位）。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	查找该值的反正弦值
Destination	SINT INT DINT **REAL**	标记	存储结果的标记

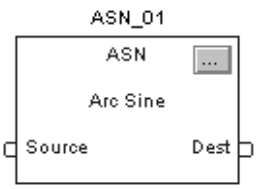
结构化文本

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

将 ASIN 用作函数。该函数计算 *source* 的反正弦值并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
ASN 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为设置状态。
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： Source 必须大于或等于 -1，小于或等于 1。 Destination 中的结果值始终大于或等于 $-\pi/2$，小于或等于 $\pi/2$ （其中 $\pi = 3.141593$）。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source 的反正弦值并将结果放入 Destination 中。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

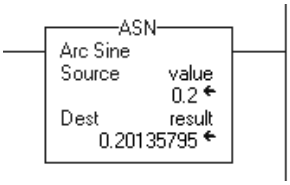


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于设置状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 计算 *value* 的反正弦值并将结果放入 *result* 中。

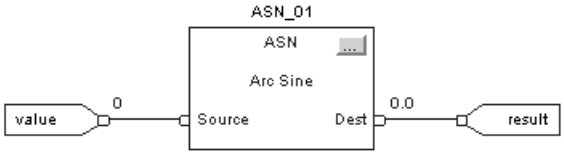
梯形图



结构化文本

```
result := ASIN(value);
```

功能块

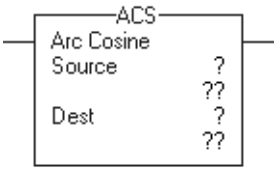


反余弦 (Arc Cosine, ACS)

ACS 指令计算 Source 值的反余弦值并将结果存储在 Destination 中（以弧度为单位）。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	查找该值的反余弦值
Destination	SINT INT DINT **REAL**	标记	存储结果的标记

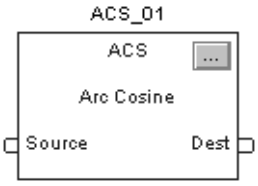
结构化文本

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

将 ACOS 用作函数。该函数计算 *source* 的反余弦值并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
ACS 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为设置状态。
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： Source 必须大于或等于 -1，小于或等于 1。 Destination 中的结果值始终大于或等于 0，小于或等于 π（其中 π = 3.141593）。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source 的反余弦值并将结果放入 Destination 中。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

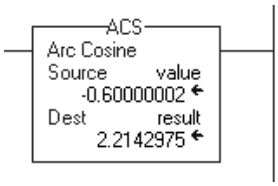


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于设置状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 计算 *value* 的反余弦值并将结果存储在 *result* 中。

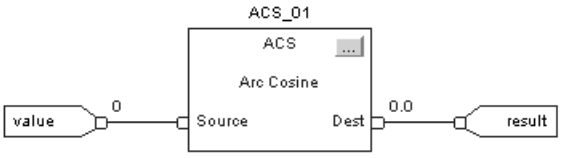
梯形图



结构化文本

```
result := ACOS(value);
```

功能块

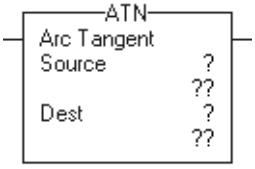


反正切 (Arc Tangent, ATN)

ATN 指令计算 Source 值的反正切值并将结果存储在 Destination 中（以弧度为单位）。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	求该值的反正切值
Destination	SINT INT DINT **REAL**	标记	存储结果的标记

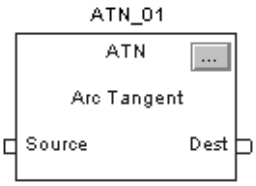
结构化文本

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

将 ATAN 用作函数。该函数计算 *source* 的反正切值并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
ATN 标记	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	启用输入。如果为清零状态，则不执行指令，也不更新输出。 默认为设置状态。
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： Destination 中的结果值始终大于或等于 $-\pi/2$，小于或等于 $\pi/2$（其中 $\pi = 3.141593$）。

算术状态标志： 算术状态标志受影响。

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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source 的反正切值并将结果放入 Destination 中。 梯级输出条件设置为 true。
后期扫描	梯级输出条件设置为 false。

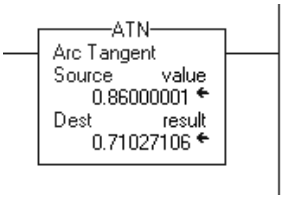


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于设置状态	指令执行。 设置 EnableOut。
后期扫描	不采取任何操作。

示例： 计算 *value* 的反正切值并将结果放入 *result* 中。

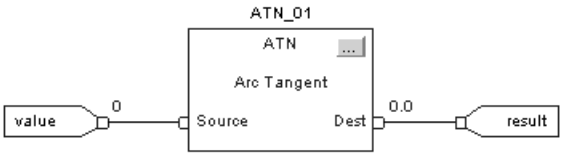
梯形图



结构化文本

```
result := ATAN(value);
```

功能块



说明：

第 14 章

高级数学指令
（LN、LOG、XPY）

简介

高级数学指令包括下列指令：

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
计算值的自然对数。	LN	梯形图 结构化文本 功能块	第 14 章 -2
计算值的以 10 为底的对数。	LOG	梯形图 结构化文本 功能块	第 14 章 -4
计算一个值的另一个值次幂，即 X 的 Y 次幂。	XPY	梯形图 结构化文本(1) 功能块	第 14 章 -6

(1) 没有等效的结构化文本指令。在表达式中使用运算符。

您可以混用数据类型，但可能发生精度损失和舍入误差，并且指令需要更长的执行时间。检查 S:V 位，以确定结果是否被截断。

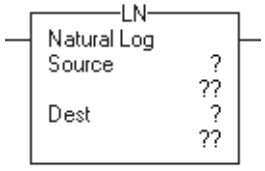
对于梯形图指令，**粗体**数据类型表示最佳数据类型。如果指令的所有操作数都使用同样的最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

自然对数 (Natural Log, LN)

LN 指令计算 Source （源）的自然对数并将结果存储在 Destination （目标）中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source （源）	SINT INT DINT **REAL**	立即数 标记	查找该值的自然对数
Destination （目标）	SINT INT DINT **REAL**	标记	存储结果的标记

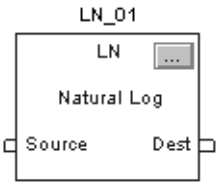
结构化文本

```
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。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
LN 标记	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	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
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	指令产生有效结果。

说明： Source （源）必须大于 0，否则会设置溢出状态位 (S:V)。所得的 Destination （目标）大于或等于 -87.33655，小于或等于 88.72284。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

梯形图

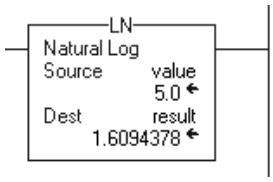
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source （源）的自然对数并将结果放入 Destination （目标）中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 计算 *value* 的自然对数并将结果放入 *result* 中。

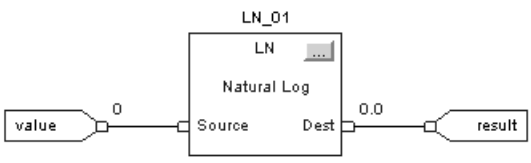
梯形图示例



结构化文本

```
result := LN(value);
```

功能块

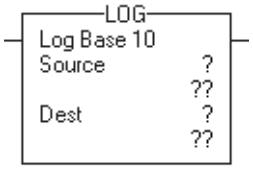


以 10 为底的对数 (Log Base 10, LOG)

LOG 指令计算 Source （源）的以 10 为底的对数并将结果存储在 Destination （目标）中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source （源）	SINT INT DINT **REAL**	立即数 标记	查找该值的对数
Destination （目标）	SINT INT DINT **REAL**	标记	存储结果的标记

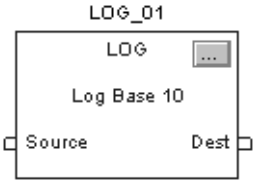
结构化文本

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

使用 LOG 作为函数。该函数计算 *source* 的对数并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
LOG 标记	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	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
Source （源）	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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： Source （源）必须大于 0，否则会设置溢出状态位 (S:V)。所得的 Destination （目标）大于或等于 -37.92978，小于或等于 38.53184。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

梯形图

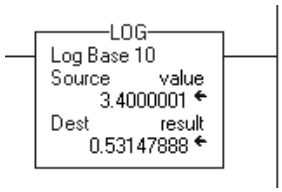
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source （源）的对数并将结果放入 Destination （目标）中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 计算 *value* 的对数并将结果放入 *result* 中。

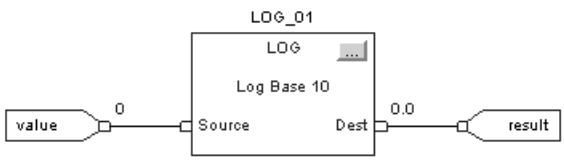
梯形图



结构化文本

```
result := LOG(value);
```

功能块

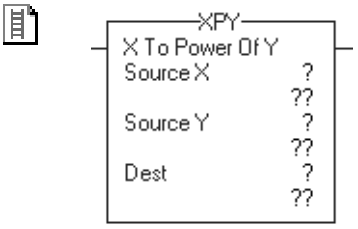


X 的 Y 次幂 (X to the Power of Y, XPY)

XPY 指令计算 Source A (X) 的 Source B (Y) 次幂并将结果存储在 Destination （目标）中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source X	SINT INT DINT **REAL**	立即数 标记	底
Source Y	SINT INT DINT **REAL**	立即数 标记	指数
Destination （目标）	SINT INT DINT **REAL**	标记	存储结果的标记

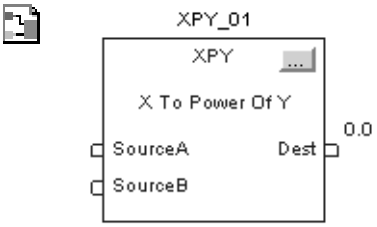
结构化文本

```
dest := sourceX ** sourceY;
```

在表达式中使用两个相邻的乘号 “**”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。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
XPY 标记	FBD_MATH	结构	XPY 结构

FBD_MATH 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
Source X	REAL	底。 任何浮点数都有效
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	指令产生有效结果。
Dest	REAL	数学指令的结果。此输出设置了算术状态标志。

说明： 如果 Source X 为负数，则 Source Y 必须是整数值，否则将产生轻微错误。

XPY 指令使用以下算法：*Destination* = X**Y

控制器计算 $x^0=1$ 和 $0^x=0$。

算术状态标志： 算术状态标志受影响。

错误条件：

出现轻微错误的条件：	错误类型：	错误代码：
Source X 是负数，Source Y 不是整数值	4	4

执行：



梯形图

条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器计算 Source X 的 Source Y 次幂并将结果放入 Destination（目标）中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

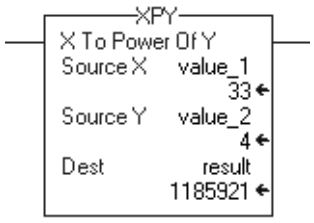


功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： XPY 指令计算 *value_1* 的 *value_2* 次幂并将结果放入 *result* 中。

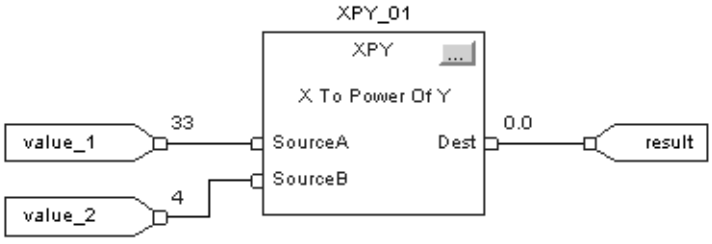
梯形图



结构化文本

```
result := (value_1 ** value_2);
```

功能块



第 15 章

数学转换指令
（DEG、 RAD、 TOD、 FRD、 TRN 和 TRUNC）

简介

数学转换指令用于转换值。

如果需要：	请使用指令：	在以下语言中可用：	请参见页：
将弧度转换为角度。	DEG	梯形图 结构化文本 功能块	第 15 章 -2
将角度转换为弧度。	RAD	梯形图 结构化文本 功能块	第 15 章 -4
将整数值转换为 BCD 值。	TOD	梯形图 功能块	第 15 章 -6
将 BCD 值转换为整数值。	FRD	梯形图 功能块	第 15 章 -9
删除值的小数部分	TRN TRUNC(1)	梯形图 结构化文本 功能块	第 15 章 -11

(1) 仅结构化文本。

您可以混用数据类型，但可能发生精度损失和舍入误差，并且指令需要更长的执行时间。检查 S:V 位，以确定结果是否被截断。

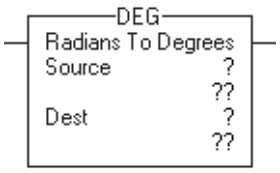
对于梯形图指令，**粗体**数据类型表示最佳数据类型。如果指令的所有操作数都使用同样的最佳数据类型 （通常为 DINT 或 REAL），指令的执行速度会更快，所需内存也更少。

角度 (Degrees, DEG)

DEG 指令将 Source （以弧度为单位）转换为角度并将结果存储在 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	要转换为角度的值
Destination	SINT INT DINT REAL	标记	存储结果的标记

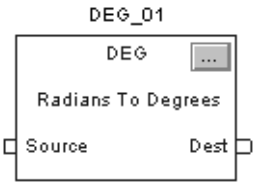
结构化文本

```
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。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
DEG 标记	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	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
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	指令产生有效结果。
Dest	REAL	转换指令的结果。此输出设置了算术状态标志。

说明： DEG 指令使用以下算法：
Source*180/π （其中 π = 3.141593）

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

梯形图

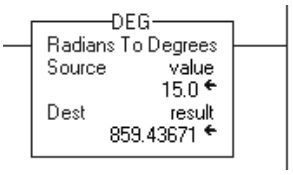
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器将 Source 转换为角度并将结果放入 Destination 中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	对 EnableOut 清零。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 将 *value* 转换为角度并将结果放入 *result* 中。

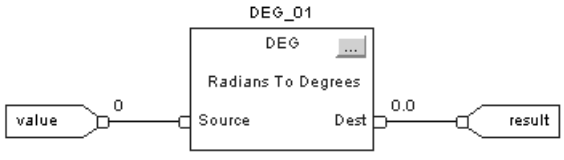
梯形图



结构化文本

```
result := DEG(value);
```

功能块

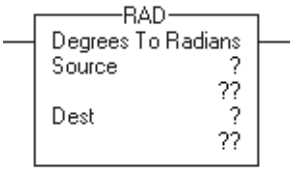


弧度 (Radian, RAD)

RAD 指令将 Source （以角度为单位）转换为弧度并将结果存储在 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT **REAL**	立即数 标记	要转换为弧度的值
Destination	SINT INT DINT REAL	标记	存储结果的标记

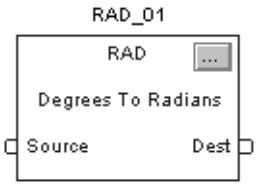
结构化文本

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

使用 RAD 作为函数。此函数将 *source* 转换为弧度并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
RAD 标记	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	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
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	指令产生有效结果。
Dest	REAL	转换指令的结果。此输出设置了算术状态标志。

说明： RAD 指令使用以下算法：
Source*π/180 （其中 π = 3.141593）

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

梯形图

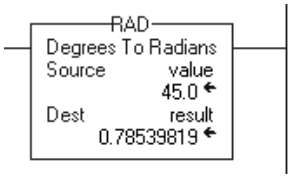
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器将 Source 转换为弧度并将结果放入 Destination 中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 将 *value* 转换为弧度并将结果放入 *result* 中。

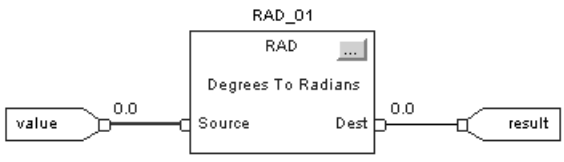
梯形图



结构化文本

```
result := RAD(value);
```

功能块

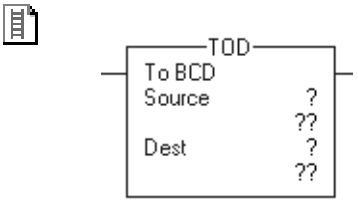


转换为 BCD (Convert to BCD, TOD)

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。

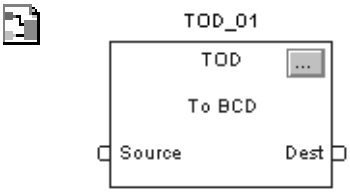
操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT	立即数 标记	要转换为十进制的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Destination	SINT INT DINT	标记	存储结果

功能块



操作数：	类型：	格式：	说明：
TOD 标记	FBD _CONVERT	结构	TOD 结构

FBD_CONVERT 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
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	指令产生有效结果。
Dest	DINT	转换指令的结果。此输出设置了算术状态标志。

说明： BCD 是二进制编码十进制 (Binary Coded Decimal) 数字系统，它用 4 位二进制表示法表示每个十进制数 (0-9)。

如果输入一个负的 Source，该指令将生成一个轻微错误并清除 Destination。

算术状态标志： 算术状态标志受影响。

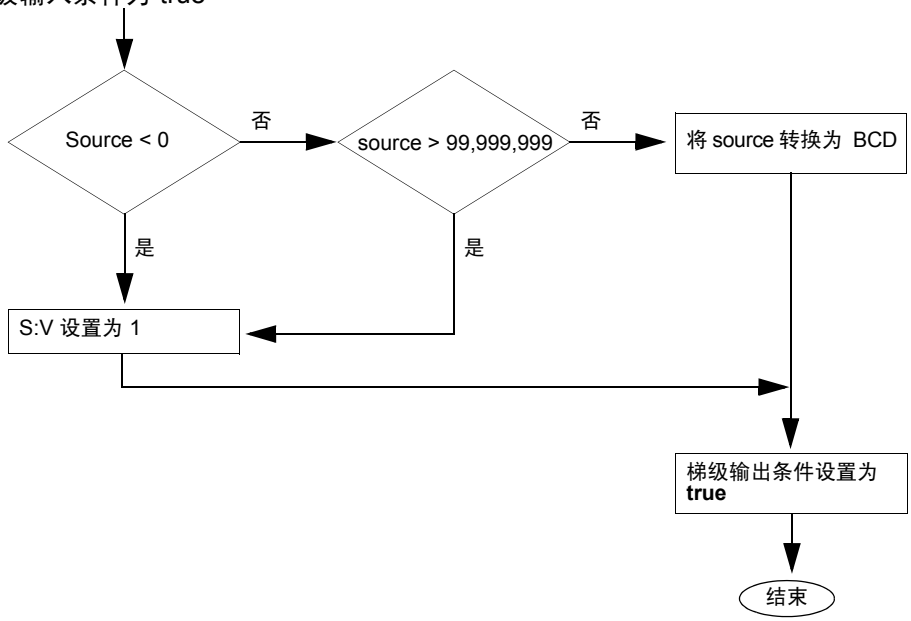
错误条件：

出现轻微错误的条件：	错误类型：	错误代码：
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 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	
梯级输入条件为 true	控制器将 Source 转换为 BCD 并将结果放入 Destination。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。



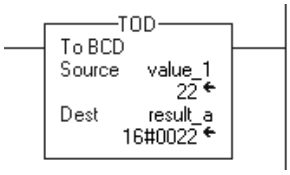
功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。

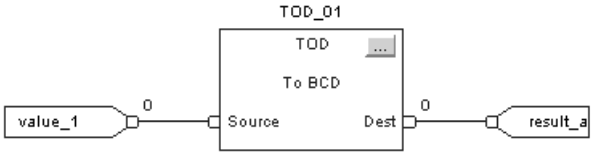
条件：	操作：
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： TOD 指令将 *value_1* 转换为 BCD 值并将结果放入 *result_a*。

梯形图



功能块

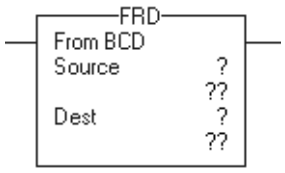


转换为整数 (Convert to Integer, FRD)

FRD 指令将 BCD 值 (Source) 转换为十进制值并将结果存储在 Destination 中。

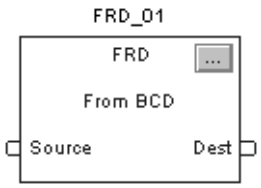
操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	SINT INT DINT	立即数 标记	要转换为十进制的值
	SINT 或 INT 标记通过填充 0 转换为 DINT 值。		
Destination	SINT INT DINT	标记	存储结果

功能块



操作数：	类型：	格式：	说明：
FRD 标记	FBD_CONVERT	结构	FRD 结构

FBD_CONVERT 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
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	指令产生有效结果。
Dest	DINT	转换指令的结果。此输出设置了算术状态标志。

说明： FRD 指令将 BCD 值 (Source) 转换为十进制值并将结果存储在 Destination 中。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

梯形图

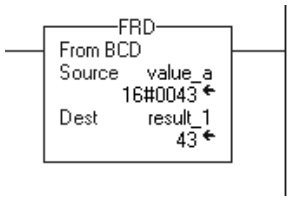
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器将 Source 转换为十进制值并将结果放入 Destination 中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

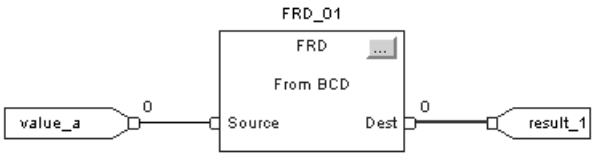
条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： FRD 指令将 *value_a* 转换为十进制值并将结果放入 *result_1* 中。

梯形图



功能块

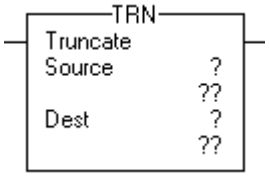


截断 (Truncate, TRN)

TRN 指令删除 （截断） Source 的小数部分并将结果存储在 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	说明：
Source	**REAL**	立即数 标记	要截断的值
Destination	SINT INT **DINT** **REAL**	标记	存储结果的标记

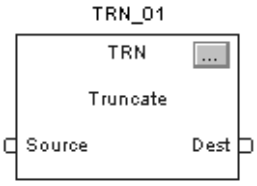
结构化文本

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

使用 TRUNC 作为函数。该函数截断 *source* 并将结果存储在 *dest* 中。

有关结构化文本中的表达式语法的信息，请参见 C。

功能块



操作数：	类型：	格式：	说明：
TRN 标记	FBD_TRUNCATE	结构	TRN 结构

FBD_TRUNCATE 结构

输入参数：	数据类型：	说明：
EnableIn	BOOL	输入使能。如果为清零状态，则不执行指令，也不更新输出。 默认为置位状态。
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	指令产生有效结果。
Dest	DINT	转换指令的结果。此输出设置了算术状态标志。

说明： 截断过程中不对值取整；无论小数部分的值是什么，非小数部分都保持不变。

算术状态标志： 算术状态标志受影响。

错误条件： 无

执行：

梯形图

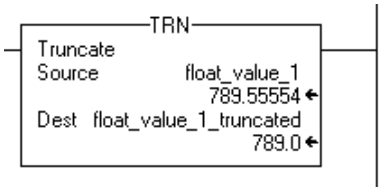
条件：	操作：
预扫描	梯级输出条件设置为 false。
梯级输入条件为 false	梯级输出条件设置为 false。
梯级输入条件为 true	控制器删除 Source 的小数部分并将结果放入 Destination 中。 梯级输出条件设置为 true。
后扫描	梯级输出条件设置为 false。

功能块

条件：	操作：
预扫描	不采取任何操作。
指令第一次扫描	不采取任何操作。
指令第一次运行	不采取任何操作。
EnableIn 处于清零状态	清除 EnableOut。
EnableIn 处于置位状态	指令执行。 设置 EnableOut。
后扫描	不采取任何操作。

示例： 删除 *float_value_1* 的小数部分，保持非小数部分不变，并将结果放入 *float_value_1_truncated* 中。

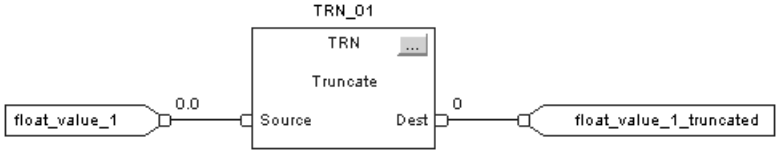
梯形图



结构化文本

```
float_value_1_truncated := TRUNC(float_value_1);
```

功能块



第 16 章

ASCII 串行端口指令

（ABL、ACB、ACL、AHL、ARD、ARL、AWA、AWT）

简介

使用 ASCII 串行端口指令可以读写 ASCII 字符。

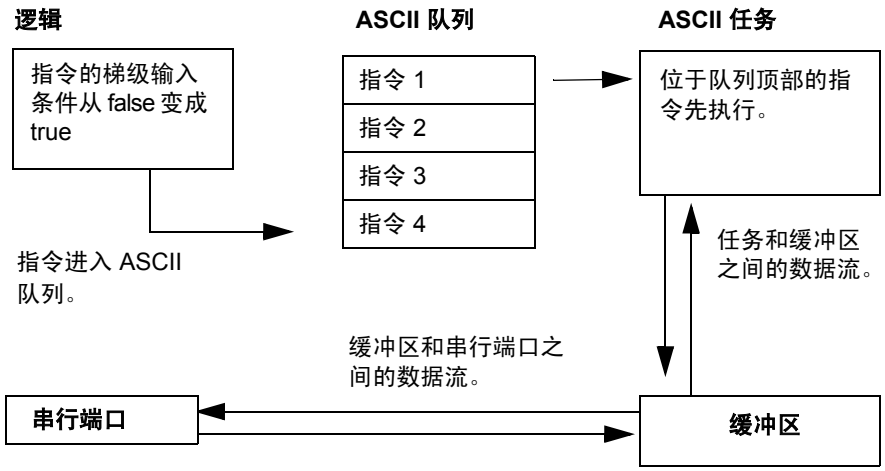
重要事项

为了使用 ASCII 串行端口指令，您必须配置控制器的串行端口。请参见 *Logix5000 Controllers Common Procedures*（Logix5000 控制器通用程序），出版号 1756-PM001。

如果需要：	例如：	请使用指令：	在以下语言中可用：	请参见页：
确定缓冲区何时包含结束符	检查包含结束符的数据	ABL	梯形图 结构化文本	第 16 章 -5
对缓冲区中的字符进行计数	读缓冲区之前检查所需的字符数	ACB	梯形图 结构化文本	第 16 章 -8
清除缓冲区 清除当前正在执行的或位于队列中的 ASCII 串行端口指令	• 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	第 16 章 -10
获取串行端口控制行的状态 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	第 16 章 -12
读取固定数目的字符	从每次传输相同数目的字符的设备读取数据	ARD	梯形图 结构化文本	第 16 章 -16
读取不定数目的字符，直到读到第一组结束符为止	从每次传输不定数目的字符的设备读取数据	ARL	梯形图 结构化文本	第 16 章 -19
发送字符并自动附加一个或两个字符以标记数据的结束	发送始终使用相同结束符的消息	AWA	梯形图 结构化文本	第 16 章 -23
发送字符	发送使用各种结束符的消息	AWT	梯形图 结构化文本	第 16 章 -28

指令执行

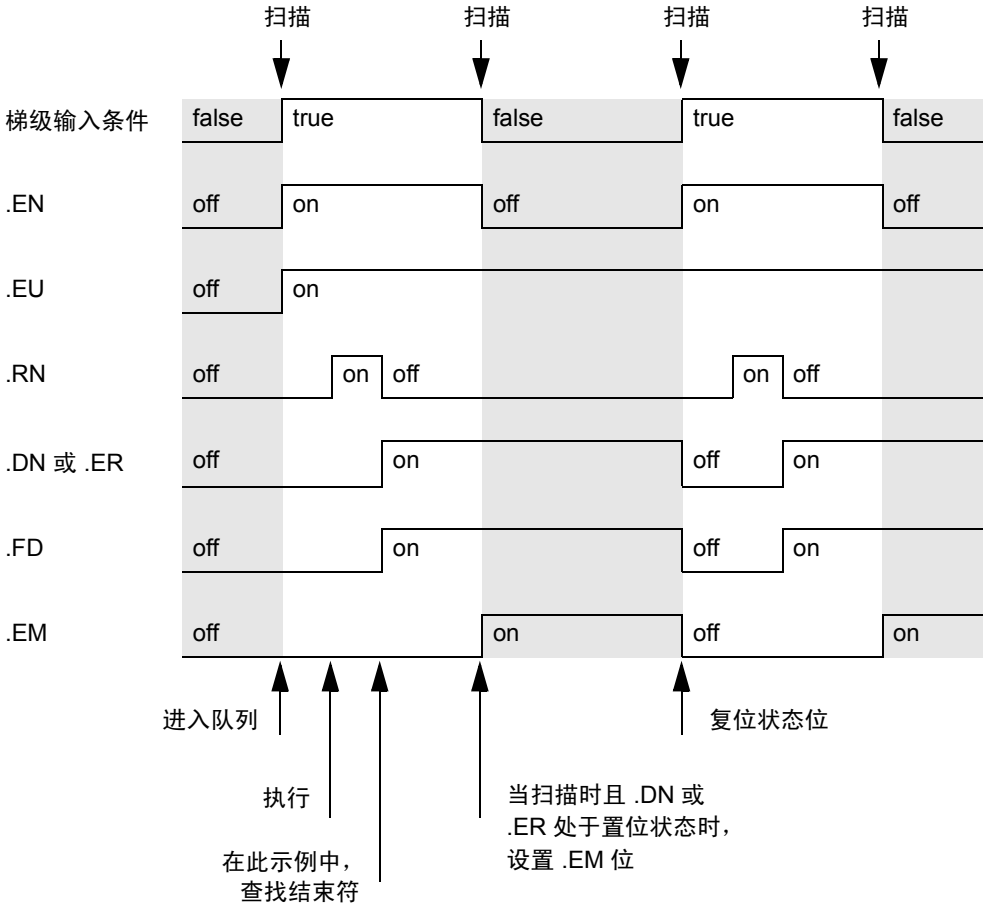
ASCII 串行端口指令的执行与逻辑扫描异步：



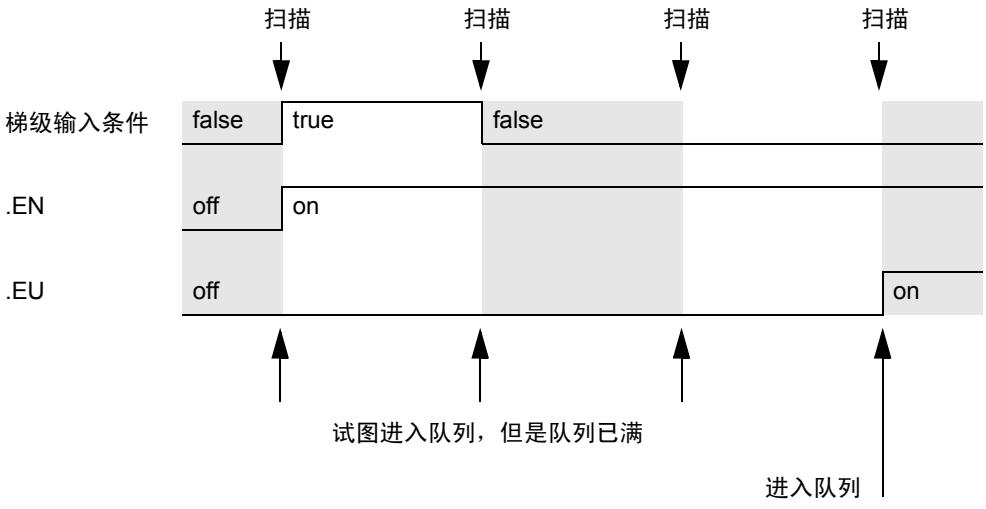
每个 ASCII 串行端口指令 （ACL 除外）都使用 SERIAL_PORT_CONTROL 结构实现以下功能：

- 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 指令检测缓冲区中的结束符时状态位的变化。



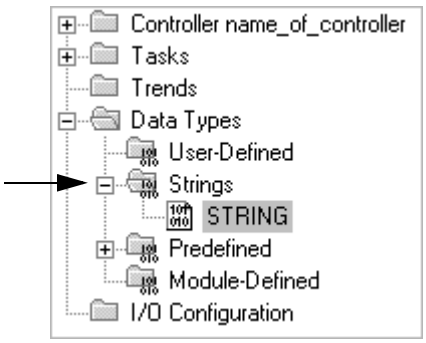
ASCII 队列最多容纳 16 条指令。当队列满时，某指令试图在每次后续扫描指令时进入队列，如下所示：



ASCII 错误代码

如果一个 ASCII 串行端口指令执行失败，其 SERIAL_PORT_CONTROL 结构的 ERROR 成员将包含下列十六进制错误代码之一：

十六进制代码：	表示：
16#2	调制解调器脱机了。
16#3	通讯过程中 CTS 信号丢失。
16#4	串行端口处于系统模式。
16#A	指令执行前， .UL 位处于置位状态。这将阻止指令的执行。
16#C	控制器从运行模式变为程序模式。这将停止 ASCII 串行端口指令的执行并清除队列。
16#D	在 Controller Properties （控制器属性）对话框的 User Protocol （用户协议）选项卡中，更改并应用了缓冲区大小或回声模式参数。这将停止 ASCII 串行端口指令的执行并清除队列。
16#E	执行了 ACL 指令。
16#F	串行端口配置从用户模式变为系统模式。这将停止 ASCII 串行端口指令的执行并清除 ASCII 串行端口指令队列。
16#51	字符串标记的 LEN 值为负或大于字符串标记的 DATA 的大小。
16#54	Serial Port Control Length 大于缓冲区的大小。
16#55	Serial Port Control Length 为负或大于 Source 或 Destination 的大小。



字符串数据类型

您将 ASCII 字符存储在使用字符串数据类型的标记中。

- 您可以使用默认的 STRING 数据类型。它最多存储 82 个字符。
- 您可以创建新的存储更少或更多字符的字符串数据类型。

要创建新的字符串数据类型，请参见 *Logix5000 Controllers Common Procedures* （Logix5000 控制器通用程序），出版号 1756-PM001。

每个字符串数据类型包含下列成员：

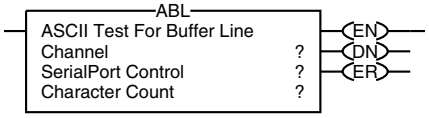
名称：	数据类型：	说明：	说明：
LEN	DINT	字符串中的字符数	当您执行以下操作时， LEN 将自动更新为字符的最新计数： • 使用 String Browser （字符串浏览器）对话框输入字符 • 使用读取、转换或操作字符串的指令 LEN 显示当前字符串的长度。 DATA 成员可能包含其他旧字符，这些字符不包括在 LEN 计数中。
DATA	SINT 数组	ASCII 字符组成的字符串	• 要访问字符串的字符，请使用标记的名称。例如，要访问 *string_1* 标记的字符，请输入 *string_1*。 • DATA 数组的每个元素包含一个字符。 • 您可以创建新的存储更少或更多字符的字符串数据类型。

缓冲区行的 ASCII 测试 (ASCII Test For Buffer Line, ABL)

ABL 指令对缓冲区中第一个结束符以前的字符 （包括第一个结束符）计数。

操作数：

梯形图



操作数：	类型：	格式：	说明：
信道	DINT	立即数 标记	0
串行端口 CONTROL	SERIAL_PORT_ CONTROL	标记	控制操作的标记
Character Count	DINT	立即数	0 在执行过程中，显示缓冲区中的字符数，包括第一组结束符。

结构化文本

```
ABL(Channel
    SerialPortControl);
```

操作数与梯形图 ABL 指令的操作数相同。您通过 SERIAL_PORT_CONTROL 结构的 .POS 成员访问 Character Count 值。

SERIAL_PORT_CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	该启用位指示指令已启用。
.EU	BOOL	该队列位指示指令已进入 ASCII 队列。
.DN	BOOL	该完成位在指令完成时进行指示，但它与逻辑扫描不同步。
.RN	BOOL	该运行位指示指令正在执行。
.EM	BOOL	该空位指示指令已完成，但它与逻辑扫描同步。
.ER	BOOL	该错误位在指令失败 （出现错误）时进行指示。
.FD	BOOL	发现位指示指令发现了结束符。
.POS	DINT	该位置确定缓冲区中的字符数，包括第一组结束符。只有当指令发现一个或多个结束符以后，它才返回此数目。
.ERROR	DINT	错误包含一个十六进制值，该值标识错误的原因。

说明： ABL 指令在缓冲区中搜索第一组结束符。如果指令发现了结束符，它将执行以下操作：

- 设置 .FD 位
- 对缓冲区中第一组结束符以前的字符 （包括第一组结束符）计数

Controller Properties （控制器属性）对话框的 User Protocol （用户协议）选项卡定义了指令认为是结束符的 ASCII 字符。

要对 ABL 指令编程，请遵循以下准则：

1. 配置用于用户模式的控制器的串行端口，并定义作为结束符的字符。

2. 这是一个可转换指令：
   - 在梯形图中，每次应执行指令时都将梯级输入条件由清零状态切换为置位状态。
   - 在结构化文本中，限制指令只对转换执行。请参见 C。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	该指令在梯级输入条件由清零状态转换为置位状态时执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令对缓冲区中的字符进行计数。 设置 .EN 位。 清除除 .UL 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 false。	不采取任何操作。

示例： 连续测试缓冲区，以查找结束符。

梯形图



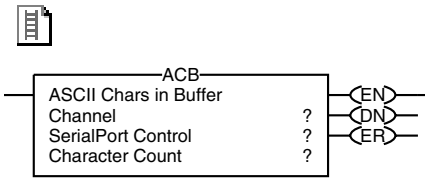
结构化文本

```
ABL(0,MV_line);
```

缓冲区中的 ASCII 字符 (ASCII Chars in Buffer, 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：	输入：
信道	DINT	立即数 标记	0
串行端口 CONTROL	SERIAL_PORT_ CONTROL	标记	控制操作的标记
Character Count	DINT	立即数	0 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。不过，可以通过访问 SERIAL_PORT_CONTROL 结构的 .POS 成员指定 Character Count 的值，而不必在操作数列表中提供该值。

SERIAL_PORT_CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	该启用位指示指令已启用。
.EU	BOOL	该队列位指示指令已进入 ASCII 队列。
.DN	BOOL	该完成位在指令完成时进行指示，但它与逻辑扫描不同步。
.RN	BOOL	该运行位指示指令正在执行。
.EM	BOOL	该空位指示指令已完成，但它与逻辑扫描同步。
.ER	BOOL	该错误位在指令失败 （出现错误）时进行指示。
.FD	BOOL	发现位指示指令找到一个字符。
.POS	DINT	该位置确定缓冲区中的字符数，包括第一组结束符。
.ERROR	DINT	错误包含一个十六进制值，该值标识错误的原因。

说明： ACB 指令对缓冲区中的字符进行计数。

要对 ACB 指令编程，请遵循以下准则：

1. 配置控制器的串行端口用于用户模式。
2. 这是一个可转换指令：
   - 在梯形图中，每次应执行指令时都将梯级输入条件由清零状态切换为置位状态。
   - 在结构化文本中，限制指令只对转换执行。请参见 C。

算术状态标志： 不受影响

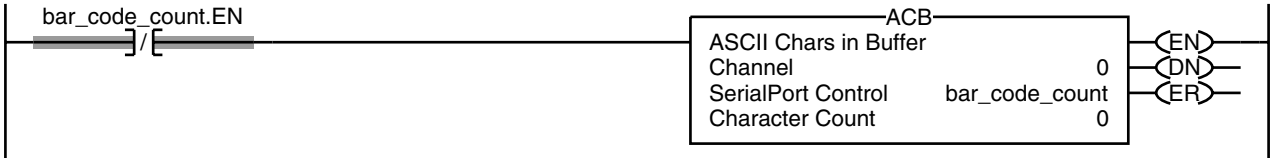
错误条件： 无

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	该指令在梯级输入条件由清零状态转换为置位状态时执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令对缓冲区中的字符进行计数。 设置 .EN 位。 清除除 .UL 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 false。	不采取任何操作。

示例： 连续对缓冲区中的字符计数。

梯形图



结构化文本

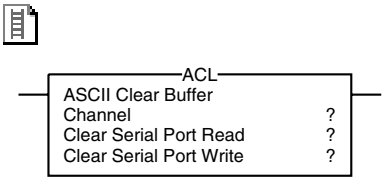
```
ACB(0,bar_code_count);
```

ASCII 清除缓冲区(ASCII Clear Buffer, ACL)

ACL 指令立即清除缓冲区和 ASCII 队列。

操作数：

梯形图



操作数：	类型：	格式：	输入：
信道	DINT	立即数 标记	0
Clear Serial Port Read	BOOL	立即数 标记	要清空缓冲区并从队列中删除 ARD 和 ARL 指令，请输入 Yes （是）。
Clear Serial Port Write	BOOL	立即数 标记	要从队列中删除 AWA 和 AWT 指令，请输入 Yes （是）。

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的一个或两个：

- 清除字符缓冲区并清除读取指令的 ASCII 队列
- 清除写入指令的 ASCII 队列

要对 ACL 指令编程，请遵循以下准则：

1. 配置控制器的串行端口：

如果您的应用程序：	则：
使用 ARD 或 ARL 指令	选择用户模式
《不》使用 ARD 或 ARL 指令	选择系统或用户模式

2. 要确定指令是否已从队列删除或放弃，请检查以下适当指令：
   - 设置 .ER 位
   - .ERROR 成员为 16#E

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	指令清除指定的指令和缓冲区。	
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 当控制器进入运行模式时，清除缓冲区和 ASCII 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);

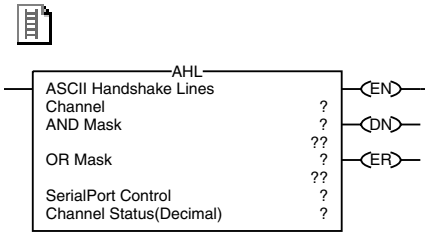
END_IF;
```

ASCII 握手行 (ASCII Handshake Lines, AHL)

AHL 指令获取控制行的状态，打开或关闭 DTR 和 RTS 信号。

操作数：

梯形图



操作数：	类型：	格式：	输入：
信道	DINT	立即数 标记	0
ANDMask	DINT	立即数 标记	请参考说明。
ORMask	DINT	立即数 标记	
Serial Port Control	SERIAL_PORT_CONTROL	标记	控制操作的标记
Channel Status（十进制）	DINT	立即数	0 执行过程中，显示控制行的状态。

对于以下控制行的状态：	检查位：
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。不过，可以通过访问 SERIAL_PORT_CONTROL 结构的 .POS 成员指定 Channel Status 的值，而不必在操作数列表中提供该值。

SERIAL_PORT_CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	启用位指示指令已启用。
.EU	BOOL	该队列位指示指令已进入 ASCII 队列。
.DN	BOOL	该完成位指示指令何时完成，但它与逻辑扫描不同步。
.RN	BOOL	该运行位指示指令正在执行。
.EM	BOOL	该空位指示指令已完成，但它与逻辑扫描同步。
.ER	BOOL	该错误位指示指令何时失败 （出现错误）。
.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	错误包含一个十六进制值，该值标识错误的原因。

说明： AHL 指令可以实现以下功能：

- 获取串行端口的控制行的状态
- 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

要对 AHL 指令编程，请遵循以下准则：

1. 配置控制器的串行端口：

如果您的应用程序：	则：
使用 ARD 或 ARL 指令	选择用户模式
《不》使用 ARD 或 ARL 指令	选择系统或用户模式

2. 使用下表为 ANDMask 和 ORMask 操作数选择正确的值：

要将 DTR 变为：	并将 RTS 变为：	请输入此 ANDMask 值：	并输入此 ORMask 值：
off	off	3	0
	on	1	2
	不变	1	0
on	off	2	1
	on	0	3
	不变	0	1
不变	off	2	0
	on	0	2
	不变	0	0

3. 这是一个可转换指令：
   - 在梯形图中，每次应执行指令时都将梯级输入条件由清零状态切换为置位状态。
   - 在结构化文本中，限制指令只对转换执行。请参见 C。

算术状态标志： 不受影响

错误条件：

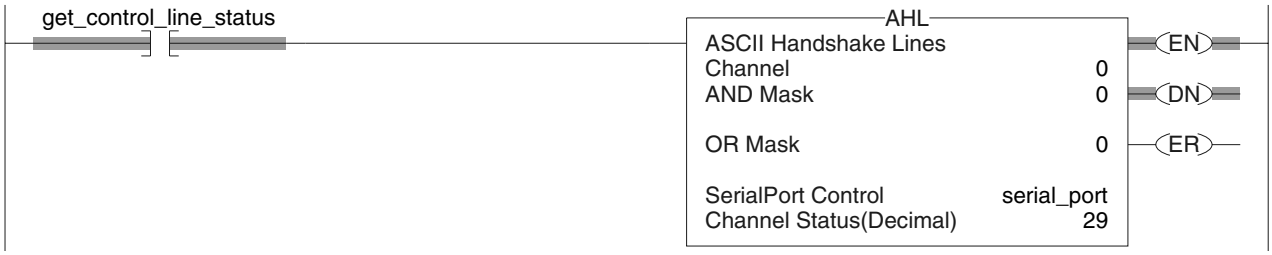
类型：	代码：	原因：	恢复方法：
4	57	AHL 指令执行失败，因为串行端口被设置为没有握手。	二选一： • 更改串行端口的控制行设置。 • 删除 AHL 指令。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	该指令在梯级输入条件由清零状态转换为置位状态时执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	指令获取控制行状态，打开或关闭 DTR 和 RTS 信号。 设置 .EN 位。 清除除 .UL 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 false。	不采取任何操作。

示例： 当 *get_control_line_status* 变成置位状态时，获取串行端口的控制行的状态，并将该状态存储到 Channel Status 操作数中。要查看特定控制行的状态，请监视 SerialPortControl 标记并扩展 POS 成员。

梯形图



结构化文本

```
osri_1.InputBit := get_control_line_status;
OSRI(osri_1);

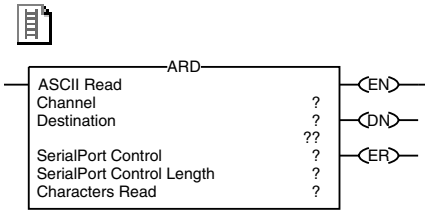
IF (osri_1.OutputBit) THEN
     AHL(0,0,0,serial_port);
END_IF;
```

ASCII 读取(ASCII Read, ARD)

ARD 指令删除缓冲区中的字符，并将它们存储到 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
信道	DINT	立即数 标记	0	
Destination	**string** SINT INT DINT	标记	要将字符移入 （读入） 其中的标记： • 对于字符串数据类型，请输入标记的名称。 • 对于 SINT、INT 或 DINT 数组，请输入数组的第一个元素。	• 要比较、转换或操作字符，请使用字符串数据类型。 • 字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
串行端口 CONTROL	SERIAL_PORT_ CONTROL	标记	控制操作的标记	
串行端口 Control Length	DINT	立即数	要移动到 destination (read) 中的字符的数目	• Serial Port Control Length 必须小于或等于 Destination 的大小。 • 要将 Serial Port Control Length 设置为与 Destination 的大小相等，请输入 0。
Characters Read	DINT	立即数	0	在执行过程中，显示已读取的字符的数目。

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。不过，可以通过访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 成员指定 Serial Port Control Length 和 Characters Read 的值，而不必在操作数列表中提供这些值。

SERIAL_PORT_CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	该启用位指示指令已启用。
.EU	BOOL	该队列位指示指令已进入 ASCII 队列。
.DN	BOOL	该完成位在指令完成时进行指示，但它与逻辑扫描不同步。
.RN	BOOL	该运行位指示指令正在执行。
.EM	BOOL	该空位指示指令已完成，但它与逻辑扫描同步。
.ER	BOOL	该错误位在指令失败 （出现错误）时进行指示。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示要移动到 destination (read) 的字符的数目。
.POS	DINT	位置显示已读取的字符的数目。
.ERROR	DINT	错误包含一个十六进制值，该值标识错误的原因。

说明： ARD 指令从缓冲区中删除指定数目的字符，并将它们存储到 Destination 中。

- ARD 指令继续执行，直到它删除了指定数目的字符 (Serial Port Control Length)。
- 在 ARD 指令执行期间，不执行其他任何 ASCII 串行端口指令。

要对 ARD 指令编程，请遵循以下准则：

1. 配置控制器的串行端口用于用户模式。
2. 使用 ACB 指令的结果触发 ARD 指令。这将防止 ARD 指令在等待所需数目的字符时挂起 ASCII 队列。
3. 这是一个可转换指令：
   - 在梯形图中，每次应执行指令时都将梯级输入条件由清零状态切换为置位状态。
   - 在结构化文本中，限制指令只对转换执行。请参见 C。
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

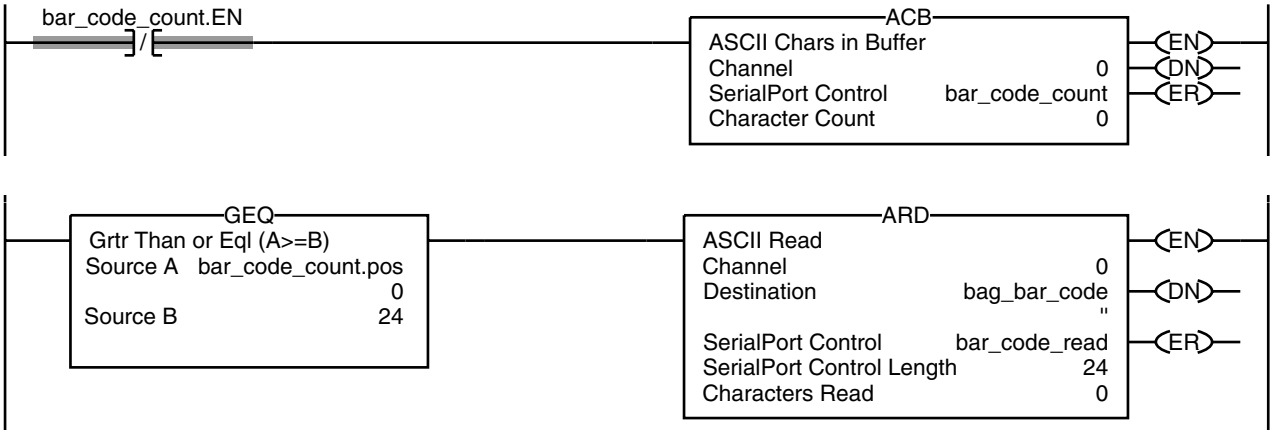
错误条件： 无

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	该指令在梯级输入条件由清零状态转换为置位状态时执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	指令从缓冲区中删除字符，并将它们存储到 destination 中。 设置 .EN 位。 清除除 .UL 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 false。	不采取任何操作。

示例： 条形码读取器将条形码发送到控制器的串行端口 （信道 0）。每个条形码包含 24 个字符。为了确定控制器何时接收条形码， ACB 指令连续对缓冲区中的字符计数。当缓冲区至少包含 24 个字符时，控制器就接收了一个条形码。 ARD 指令将条形码移动到 *bag_bar_code* 标记的 DATA 成员 （它是一个字符串）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);

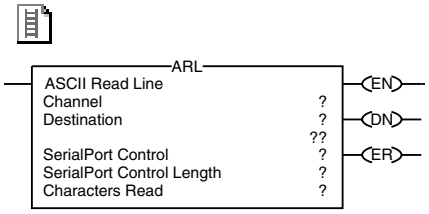
END_IF;
```

ASCII 读取行 (ASCII Read Line, ARL)

ARL 指令从缓冲区中删除指定的字符，并将它们存储到 Destination 中。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
信道	DINT	立即数 标记	0	
Destination	**string** SINT INT DINT	标记	要将字符移入 （读入） 其中的标记： • 对于字符串数据类型，请输入标记的名称。 • 对于 SINT、 INT 或 DINT 数组，请输入数组的第一个元素。	• 要比较、转换或操作字符，请使用字符串数据类型。 • 字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
串行端口 CONTROL	SERIAL_PORT_ CONTROL	标记	控制操作的标记	
Serial Port Control Length	DINT	立即数	在未找到结束符的情况下要读取的最大字符数	• 输入消息将包含的最大字符数 （即如果未找到结束符何时停止读取）。 例如，如果消息的长度范围为 3 到 6 个字符，请输入 6。 • Serial Port Control Length 必须小于或等于 Destination 的大小。 • 要将 Serial Port Control Length 设置为与 Destination 的大小相等，请输入 0。
Characters Read	DINT	立即数	0	在执行过程中，显示已读取的字符的数目。

结构化文本

```
ARL(Channel,Destination,
    SerialPortControl);
```

操作数与梯形图 ARL 指令的操作数相同。不过，可以通过访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 成员指定 Serial Port Control Length 和 Characters Read 的值，而不必在操作数列表中提供这些值。

SERIAL_PORT_CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	该启用位指示指令已启用。
.EU	BOOL	该队列位指示指令已进入 ASCII 队列。
.DN	BOOL	该完成位在指令完成时进行指示，但它与逻辑扫描不同步。
.RN	BOOL	该运行位指示指令正在执行。
.EM	BOOL	该空位指示指令已完成，但它与逻辑扫描同步。
.ER	BOOL	该错误位在指令失败 （出现错误）时进行指示。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示要移动到 destination 的最大字符数 （即如果未找到结束符何时停止读取）。
.POS	DINT	位置显示已读取的字符的数目。
.ERROR	DINT	错误包含一个十六进制值，该值标识错误的原因。

说明： ARL 指令从缓冲区中删除字符，并将它们存储到 Destination 中，如下所示：

- ARL 指令继续执行，直到它删除下列之一：
  - 第一组结束符
  - 指定的字符数 (Serial Port Control Length)
- 在 ARL 指令执行期间，其他任何 ASCII 串行端口指令都不执行。

要对 ARL 指令编程，请遵循以下准则：

1. 配置控制器的串行端口：
   a. 选择用户模式。
   b. 定义作为结束符的字符。
2. 用 ABL 指令的结果触发 ARL 指令。这将防止 ARL 指令在等待结束符期间挂起 ASCII 队列。
3. 这是一个可转换指令：
   - 在梯形图中，每次应执行指令时都将梯级输入条件由清零状态切换为置位状态。
   - 在结构化文本中，限制指令只对转换执行。请参见 C。
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

错误条件： 无

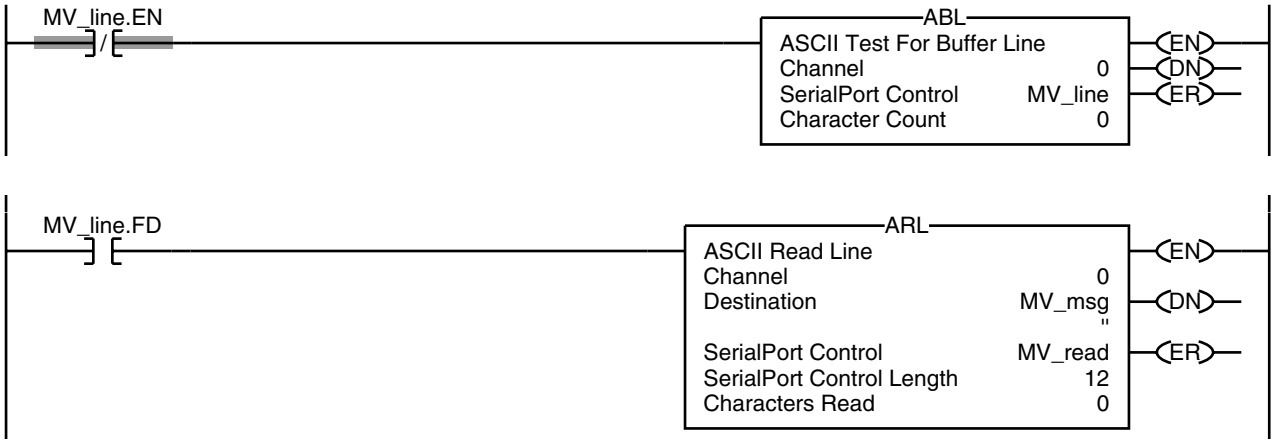
执行：

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	该指令在梯级输入条件由清零状态转换为置位状态时执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	指令从缓冲区中删除指定的字符，并将它们存储到 destination 中。 设置 .EN 位。 清除除 .UL 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 false。	不采取任何操作。

示例： 连续测试缓冲区，以查找来自 MessageView 终端的消息。由于每条消息都以回车符 ($r) 结束，在 Controller Properties （控制器属性）对话框的 User Protocol （用户协议）选项卡中回车符被配置为结束符。当 ABL 发现回车符时，它将设置 FD 位。

当 ABL 指令发现回车符时 （设置 *MV_line.FD*），控制器就接收了一条完整的消息。 ARL 指令从缓冲区中删除字符，一直删到回车符（包括删除回车符），并将它们放入 *MV_msg* 标记的 DATA 成员（它是一个字符串）中。

梯形图



结构化文本

```
ABL(0,MV_line);


osri_1.InputBit := MVLine.FD;
OSRI(osri_1);

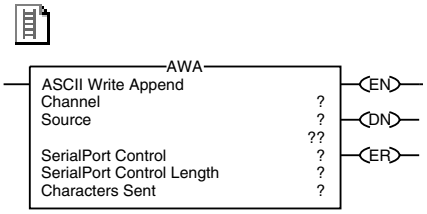

IF (osri_1.OutputBit) THEN
     mv_read.LEN := 12;
     ARL(0,MV_msg,MV_read);
END_IF;
```

ASCII 写入附加 (ASCII Write Append, AWA)

AWA 指令将 Source 标记中的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的字符。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
信道	DINT	立即数 标记	0	
Source	**string** SINT INT DINT	标记	包含要发送的字符的标记： • 对于字符串数据类型，请输入标记的名称。 • 对于 SINT、 INT 或 DINT 数组，请输入数组的第一个元素。	• 要比较、转换或操作字符，请使用字符串数据类型。 • 字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
串行端口 CONTROL	SERIAL_PORT_ CONTROL	标记	控制操作的标记	
Serial Port Control Length	DINT	立即数	要发送的字符的数目	• Serial Port Control Length 必须小于或等于 Source 的大小。 • 要将 Serial Port Control Length 设置为与 Source 中的字符数相等，请输入 0。
Characters Sent	DINT	立即数	0	在执行过程中，显示已发送的字符的数目。

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。不过，可以通过访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 成员指定 Serial Port Control Length 和 Characters Sent 的值，而不必在操作数列表中提供这些值。

SERIAL_PORT_CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	该启用位指示指令已启用。
.EU	BOOL	该队列位指示指令已进入 ASCII 队列。
.DN	BOOL	该完成位在指令完成时进行指示，但它与逻辑扫描不同步。
.RN	BOOL	该运行位指示指令正在执行。
.EM	BOOL	该空位指示指令已完成，但它与逻辑扫描同步。
.ER	BOOL	该错误位在指令失败 （出现错误）时进行指示。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示要发送的字符的数目。
.POS	DINT	位置显示已发送的字符的数目。
.ERROR	DINT	错误包含一个十六进制值，该值标识错误的原因。

说明： AWA 指令：

- 将 Source 标记中指定数目的字符 (Serial Port Control Length) 发送到与控制器的串行端口相连的设备
- 向字符末尾添加 （附加）一个或两个在 Controller Properties （控制器属性）对话框的 User Protocol （用户协议）选项卡中定义的字符

要对 AWA 指令编程，请遵循以下准则：

1. 配置控制器的串行端口：

   a. 您的应用程序是否还包括 ARD 或 ARL 指令？

如果：	则：
是	选择用户模式
否	选择系统或用户模式

   b. 定义要附加到数据的字符。

2. 这是一个可转换指令：
   - 在梯形图中，每次应执行指令时都将梯级输入条件由清零状态切换为置位状态。
   - 在结构化文本中，限制指令只对转换执行。请参见 C。

3. 每次执行指令时，您是否始终发送相同数目的字符？

如果：	则：
是	在 Serial Port Control Length 中，输入要发送的字符数。
否	在指令执行前，将 Source 标记的 LEN 成员设置为 Serial Port Control 标记的 LEN 成员。

算术状态标志： 不受影响

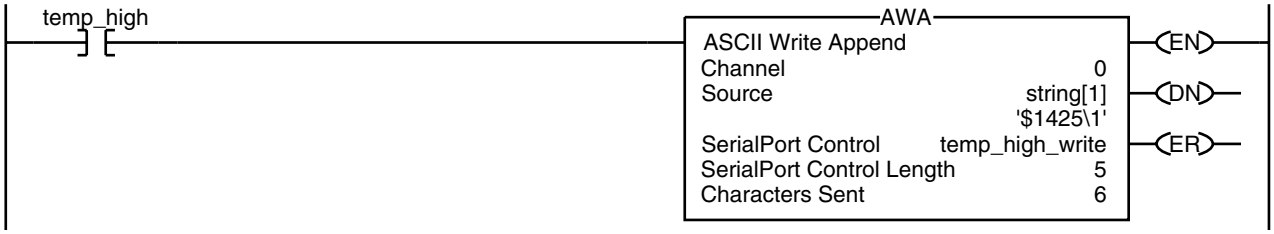
错误条件： 无

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	该指令在梯级输入条件由清零状态转换为置位状态时执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	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。 设置 .EN 位。 清除除 .UL 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 false。	不采取任何操作。

示例 1： 当温度超过最高限度 （*temp_high* 处于置位状态）， AWA 指令向与控制器的串行端口相连的 MessageView 终端发送一条消息。该消息包含 *string[1]* 标记的 DATA 成员 （它是一个字符串）的五个字符。（*$14* 被视为一个字符。它是 Ctrl-T 字符对应的十六进制代码。）该指令还发送 （附加）在控制器属性中定义的字符。在此示例中， AWA 指令发送一个回车符 ($0D)，它标志消息的结束。

梯形图



结构化文本

```
IF temp_high THEN

     temp_high_write.LEN := 5;

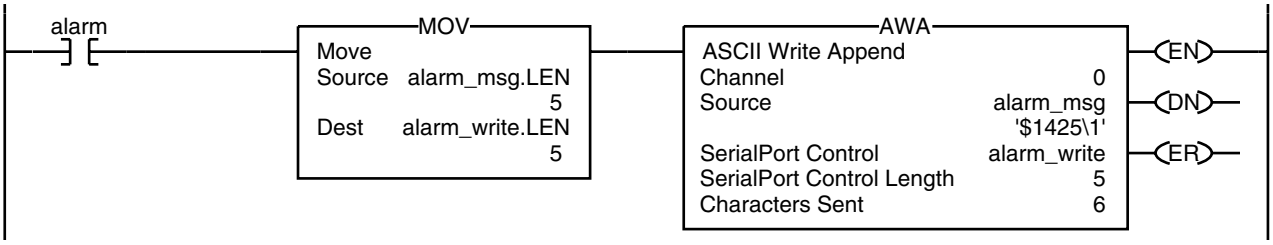
     AWA(0,string[1],temp_high_write);

     temp_high := 0;

END_IF;
```

示例 2： 当 *alarm* 处于置位状态时， AWA 指令发送 *alarm_msg* 中指定数目的字符，并附加一个结束符。因为 *alarm_msg* 中的字符数是变化的，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* 被视为一个字符。它是 Ctrl-T 字符对应的十六进制代码。

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);

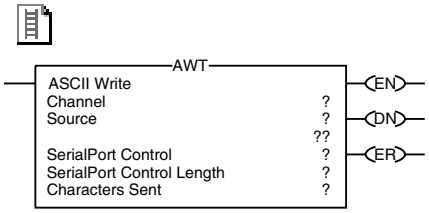
END_IF;
```

ASCII 写入 (ASCII Write, AWT)

AWT 指令将 Source 标记中的指定数目的字符发送到串行设备中。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
信道	DINT	立即数 标记	0	
Source	**string** SINT INT DINT	标记	包含要发送的字符的标记： • 对于字符串数据类型，请输入标记的名称。 • 对于 SINT、 INT 或 DINT 数组，请输入数组的第一个元素。	• 要比较、转换或操作字符，请使用字符串数据类型。 • 字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
串行端口 CONTROL	SERIAL_PORT_ CONTROL	标记	控制操作的标记	
Serial Port Control Length	DINT	立即数	要发送的字符的数目	• Serial Port Control Length 必须小于或等于 Source 的大小。 • 要将 Serial Port Control Length 设置为与 Source 中的字符数相等，请输入 0。
Characters Sent	DINT	立即数	0	在执行过程中，显示已发送的字符的数目。

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。不过，可以通过访问 SERIAL_PORT_CONTROL 结构的 .LEN 和 .POS 成员指定 Serial Port Control Length 和 Characters Sent 的值，而不必在操作数列表中提供这些值

SERIAL_PORT_CONTROL 结构

助记符：	数据类型：	说明：
.EN	BOOL	该启用位指示指令已启用。
.EU	BOOL	该队列位指示指令已进入 ASCII 队列。
.DN	BOOL	该完成位在指令完成时进行指示，但它与逻辑扫描不同步。
.RN	BOOL	该运行位指示指令正在执行。
.EM	BOOL	该空位指示指令已完成，但它与逻辑扫描同步。
.ER	BOOL	该错误位在指令失败 （出现错误）时进行指示。
.FD	BOOL	发现位不适用于此指令。
.LEN	DINT	长度指示要发送的字符的数目。
.POS	DINT	位置显示已发送的字符的数目。
.ERROR	DINT	错误包含一个十六进制值，该值标识错误的原因。

说明： AWT 指令将 Source 标记中指定数目的字符 (Serial Port Control Length) 发送到与控制器的串行端口相连的设备。

要对 AWT 指令编程，请遵循以下准则：

1. 配置控制器的串行端口：

如果您的应用程序：	则：
使用 ARD 或 ARL 指令	选择用户模式
《不》使用 ARD 或 ARL 指令	选择系统或用户模式

2. 这是一个可转换指令：
   - 在梯形图中，每次应执行指令时都将梯级输入条件由清除状态切换为置位状态。
   - 在结构化文本中，限制指令只对转换执行。请参见 C。

3. 每次执行指令时，您是否始终发送相同数目的字符？

如果：	则：
是	在 Serial Port Control Length 中，输入要发送的字符数。
否	在执行指令前，将 Source 标记的 LEN 成员移动到 Serial Port Control 标记的 LEN 成员。

算术状态标志： 不受影响

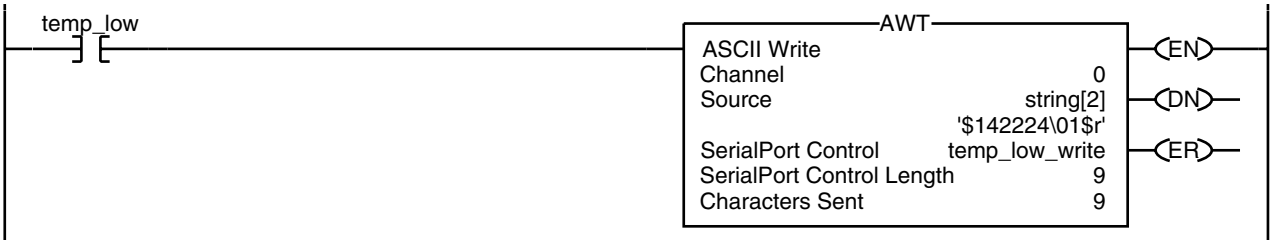
错误条件： 无

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	该指令在梯级输入条件由清除状态转换为置位状态时执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	指令发送指定数目的字符。 设置 .EN 位。 清除除 .UL 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 false。	不采取任何操作。

示例 1： 当温度达到最低限度 （*temp_low* 处于置位状态）， AWT 指令将向连接到控制器的串行端口的 MessageView 终端发送消息。该消息包含来自 *string[2]* 标记的 DATA 成员 （它是一个字符串）的九个字符。（*$14* 被视为一个字符。它是 Ctrl-T 字符对应的十六进制代码。）最后一个字符是回车符 ($r)，它标记消息的结束。

梯形图



结构化文本

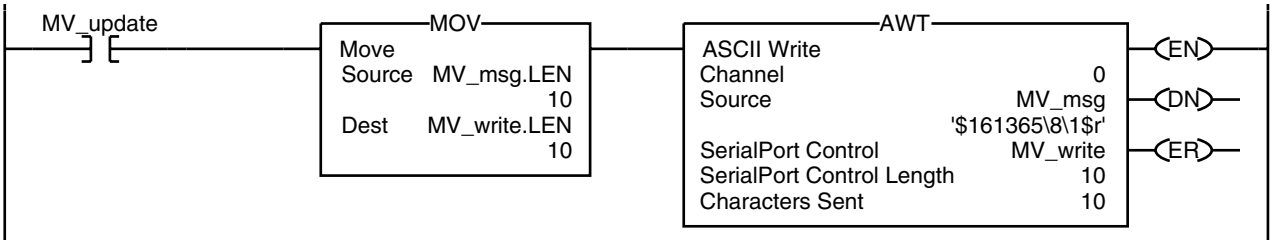
```
osri_1.InputBit := temp_low;

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* 处于置位状态时， AWT 指令发送 *MV_msg* 中的字符。因为 *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) 中。在 *MV_msg* 中， *$16* 被视为一个字符。它是 Ctrl-V 字符对应的十六进制代码。

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;
```

说明：

第 17 章

ASCII 字符串指令
（CONCAT、 DELETE、 FIND、 INSERT、 MID）

简介

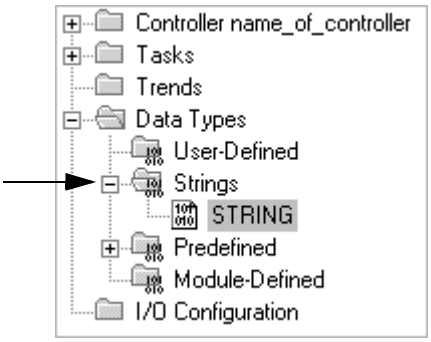
ASCII 字符串指令用于修改和创建 ASCII 字符串。

如果需要：	例如：	请使用指令：	在以下语言中可用：	请参见页：
将字符添加到字符串末尾	将结束字符或分隔符添加到字符串	CONCAT	梯形图 结构化文本	第 17 章 -2
从字符串中删除字符	从字符串删除头字符或控制字符	DELETE	梯形图 结构化文本	第 17 章 -5
确定子字符串的起始字符	在字符串中定位一组字符	FIND	梯形图 结构化文本	第 17 章 -7
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	第 17 章 -9
从字符串中提取字符	从条形码提取信息	MID	梯形图 结构化文本	第 17 章 -11

此外，还可以使用下列指令比较或转换 ASCII 字符：

如果需要：	请使用指令：	请参见页：
比较两个字符串	CMP	第 4 章 -2
查看字符是否等于特定字符	EQU	第 4 章 -7
查看字符是否不等于特定字符	NEQ	第 4 章 -38
查看字符是否等于或大于特定字符	GEQ	第 4 章 -11
查看字符是否大于特定字符	GRT	第 4 章 -15
查看字符是否等于或小于特定字符	LEQ	第 4 章 -19
查看字符是否小于特定字符	LES	第 4 章 -23
重新排列 INT、 DINT 或 REAL 标记的字节	SWPB	第 6 章 -19
在一组字符串中查找某个字符串	FSC	第 7 章 -19
将字符转换为 SINT、 INT、 DINT 或 REAL 值	STOD	第 18 章 -3
将字符转换为 REAL 值	STOR	第 18 章 -5
将 SINT、 INT、 DINT 或 REAL 值转换为 ASCII 字符串	DTOS	第 18 章 -7
将 REAL 值转换为 ASCII 字符串	RTOS	第 18 章 -9

字符串数据类型



ASCII 字符存储在使用字符串数据类型的标记中。

- 可以使用默认的 STRING 数据类型。该数据类型最多存储 82 个字符。
- 您可以创建新的字符串数据类型，以存储更少或更多的字符。

若要创建新的字符串数据类型，请参见 *Logix5000* 控制器通用编程步骤，出版号 1756-PM001。

每种字符串数据类型都包含以下成员：

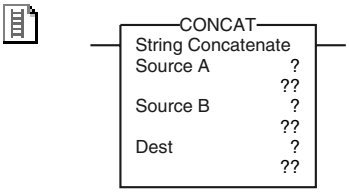
名称：	数据类型：	说明：	注意：
LEN	DINT	字符串中的字符数	每当执行以下操作时， LEN 都会自动更新为新的字符数： • 使用 String Browser （字符串浏览器）对话框输入字符 • 使用字符串读取、转换或操作指令 LEN 显示当前字符串的长度。 DATA 成员可能包含附加的或旧的字符，这些字符不包含在 LEN 计算中。
DATA	SINT 数组	字符串中的 ASCII 字符	• 要访问字符串中的字符，请引用标记的名称。例如，若要访问 *string_1* 标记中的字符，请输入 *string_1*。 • DATA 数组的每个元素包含一个字符。 • 您可以创建新的字符串数据类型，以存储更少或更多的字符。

字符串串联 (String Concatenate, CONCAT)

CONCAT 指令将 ASCII 字符添加到字符串的末尾。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
Source A	string	标记	包含起始字符的标记	字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
Source B	string	标记	包含结束字符的标记	
Destination	string	标记	存储结果的标记	

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 指令合并了 Source A 和 Source B 中的字符，并将结果放在 Destination 中。

- 首先放置 Source A 中的字符，然后是 Source B 中的字符。
- 除非 Source A 和 Destination 是同一个标记，否则 Source A 保持不变。

算术状态标志： 不受影响

错误条件：

类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令将字符串串联在一起。	
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 要触发 MessageView 终端中的消息，控制器必须发送包含消息编号和节点编号的 ASCII 字符串。*String_1* 包含消息编号。当 *add_node* 为置位状态时， CONCAT 指令将 *node_num_ascii*（节点编号）中的字符添加到 *string_1* 中的字符的末尾，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;

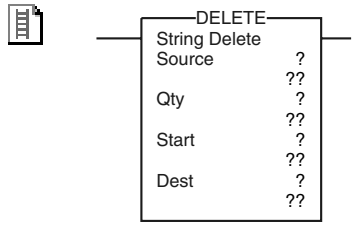
END_IF;
```

删除字符串 (String Delete, DELETE)

DELETE 指令从字符串中删除 ASCII 字符。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
Source	string	标记	包含要从中删除字符的字符串的标记	字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
Quantity	SINT INT **DINT**	立即数 标记	要删除的字符数	Start 与 Quantity 之和必须小于或等于 Source 的 DATA 大小。
Start	SINT INT **DINT**	立即数 标记	要删除的第一个字符的位置	输入一个介于 1 与 Source 的 DATA 大小之间的数值。
Destination	string	标记	存储结果的标记	

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。

说明： DELETE 指令从 Source 中删除 （移除）一组字符，并将其余的字符放在 Destination 中。

- Start 位置和 Quantity 定义要删除的字符。
- 除非 Source 与 Destination 是同一个标记，否则 Source 保持不变。

算术状态标志： 不受影响

错误条件：

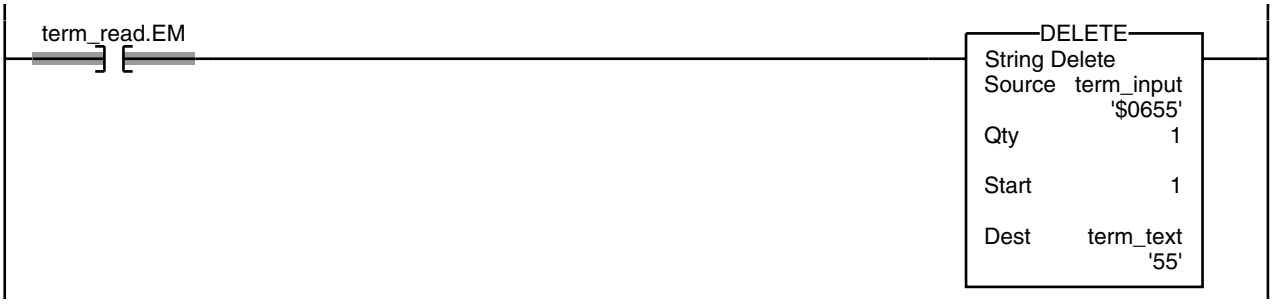
类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	56	Start 或 Quantity 值无效。	1. 确保 Start 值介于 1 与 Source 的 DATA 大小之间。 2. 确保 Start 值与 Quantity 值之和小于或等于 Source 的 DATA 大小。

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 false。	无操作
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令删除指定的字符。	
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 终端中的 ASCII 信息包含头字符。控制器读取数据 （设置 *term_read.EM*）之后， DELETE 指令将删除头字符。

梯形图



结构化文本

```
IF term_read.EM THEN

     DELETE(term_input,1,1,term_text);

     term_read.EM := 0;

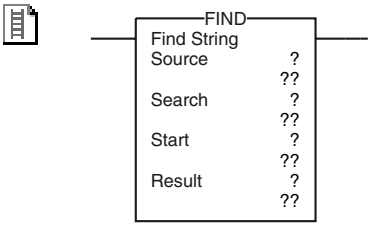
END_IF;
```

查找字符串 (Find String, FIND)

FIND 指令在另一个字符串中定位指定字符串的起始位置

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
Source	string	标记	要在其中进行搜索的字符串	字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
Search	string	标记	要查找的字符串	
Start	SINT INT **DINT**	立即数 标记	Source 中要开始进行搜索的位置	输入一个介于 1 与 Source 的 DATA 大小之间的数值。
Result	SINT INT **DINT**	标记	存储要查找的字符串的起始位置的标记	

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 Source 字符串中搜索 Search 字符串。如果该指令找到 Search 字符串，Result 将显示 Search 字符串在 Source 字符串中的起始位置。

算术状态标志： 不受影响

错误条件：

类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	56	Start 值无效。	确保 Start 值介于 1 与 Source 的 DATA 大小之间。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令搜索指定的字符。	
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： MessageView 终端中的消息包含几段信息。反斜杠字符 [\ ] 用于分隔每段信息。要定位某一段信息， FIND 指令将搜索反斜杠字符，然后将其位置记录在 *find_pos* 中。

梯形图



结构化文本

```
IF MV_read.EM THEN

     FIND(MV_msg,find,1,find_pos);

     MV_read.EM := 0;

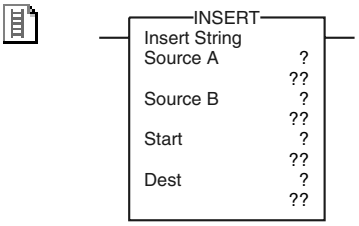
END_IF;
```

插入字符串 (Insert String, INSERT)

INSERT 指令将 ASCII 字符添加到字符串中的指定位置。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
Source A	string	标记	要向其中添加字符的字符串	字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
Source B	string	标记	包含要添加的字符的字符串	
Start	SINT INT **DINT**	立即数 标记	要在 Source A 中添加字符的位置	输入一个介于 1 与 Source 的 DATA 大小之间的数值。
Result	string	标记	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 Source B 中的字符添加到 Source A 中的指定位置，并将结果放在 Destination 中：

- Start 定义在 Source A 中添加 Source B 的位置。
- 除非 Source A 和 Destination 是同一个标记，否则 Source A 保持不变。

算术状态标志： 不受影响

错误条件：

类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	56	Start 值无效。	确保 Start 值介于 1 与 Source 的 DATA 大小之间。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令插入指定的字符。	
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 当 *temp_high* 为置位状态时，INSERT 指令将 *string_2* 中的字符添加到 *string_1* 中的第 2 个位置，并将结果放在 *string_3* 中：

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;
```

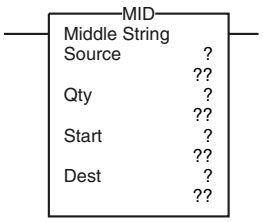
提取字符串 (Middle String, MID)

MID 指令复制字符串中指定数目的 ASCII 字符，并将这些字符存储在另一个字符串中。

操作数：

梯形图





操作数：	类型：	格式：	输入：	说明：
Source	string	标记	要从中复制字符的字符串	字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
Quantity	SINT INT **DINT**	立即数 标记	要复制的字符数	Start 与 Quantity 之和必须小于或等于 Source 的 DATA 大小。
Start	SINT INT **DINT**	立即数 标记	要复制的第一个字符的位置	输入一个介于 1 与 Source 的 DATA 大小之间的数值。
Destination	string	标记	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 指令复制 Source 中的一组字符，并将结果放在 Destination 中。

- Start 位置和 Quantity 定义要复制的字符。
- 除非 Source 与 Destination 是同一个标记，否则 Source 保持不变。

算术状态标志： 不受影响

错误条件：

类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	56	Start 或 Quantity 值无效。	1. 确保 Start 值介于 1 与 Source 的 DATA 大小之间。 2. 确保 Start 值与 Quantity 值之和小于或等于 Source 的 DATA 大小。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于置位状态	不可用	EnableIn 始终为置位状态。 指令执行。
指令执行	该指令复制字符串中的指定字符，并将这些字符存储在另一个字符串中。	
后扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 在机场的行李处理传送带上，每个行李都有一个条形码。条形码中的字符 9 - 17 是航班号和行李的到港机场。读取条形码 （*bag_read.EM* 为置位状态）之后， MID 指令会将航班号和到港机场字符复制到 *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;
```

第 18 章

ASCII 转换指令

（STOD、 STOR、 DTOS、 RTOS、 UPPER、 LOWER）

简介

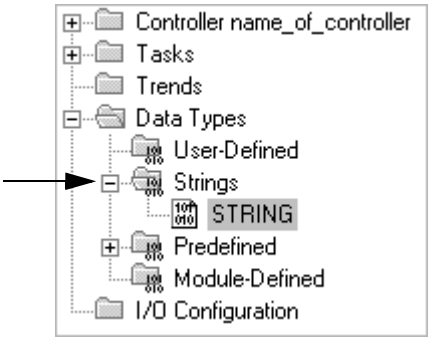
ASCII 转换指令用于改变数据的格式。

操作目的：	示例：	操作指令：	适用语言：	参考页面：
将整数值的 ASCII 表示转换为 SINT、INT、DINT 或 REAL 值	将值从台秤或其他 ASCII 设备转换为整数值，以便您可以在自己的逻辑中使用它	STOD	梯形图 结构化文本	第 18 章 -4
将浮点值的 ASCII 表示转换为 REAL 值	将值从台秤或其他 ASCII 设备转换为 REAL 值，以便您可以在自己的逻辑中使用它	STOR	梯形图 结构化文本	第 18 章 -6
将 SINT、INT、DINT 或 REAL 值转换为 ASCII 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	第 18 章 -8
将 REAL 值转换为 ASCII 字符组成的字符串	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	第 18 章 -10
将由 ASCII 字符组成的字符串中的字母转换为大写	将由操作员输入的内容全部转换为大写，以便您可以在数组中搜索它	UPPER	梯形图 结构化文本	第 18 章 -12
将由 ASCII 字符组成的字符串中的字母转换为小写	将由操作员输入的内容全部转换为小写，以便您可以在数组中搜索它	LOWER	梯形图 结构化文本	第 18 章 -14

您还可以使用下列指令来比较或操作 ASCII 字符：

操作目的：	**操作指令：**	**参考页面：**
将字符添加到字符串末尾	CONCAT	第 17 章 -2
从字符串中删除字符	DELETE	第 17 章 -5
确定子字符串的起始字符	FIND	第 17 章 -7
将字符插入字符串	INSERT	第 17 章 -9
从字符串中提取字符	MID	第 17 章 -11
重新排列 INT、 DINT 或 REAL 标记的字节	SWPB	第 6 章 -19
比较两个字符串	CMP	第 4 章 -2
查看字符是否等于特定字符	EQU	第 4 章 -7
查看字符是否不等于特定字符	NEQ	第 4 章 -38
查看字符是否等于或大于特定字符	GEQ	第 4 章 -11
查看字符是否大于特定字符	GRT	第 4 章 -15
查看字符是否等于或小于特定字符	LEQ	第 4 章 -19
查看字符是否小于特定字符	LES	第 4 章 -23
在一组字符串中查找某个字符串	FSC	第 7 章 -19

字符串数据类型



ASCII 字符存储在使用字符串数据类型的标记中。

- 可以使用默认的 STRING 数据类型。该数据类型最多存储 82 个字符。
- 您可以创建新的字符串数据类型，以存储更少或更多的字符。

若要创建新的字符串数据类型，请参见 *Logix5000* 控制器通用编程步骤，出版号 1756-PM001。

每种字符串数据类型都包含以下成员：

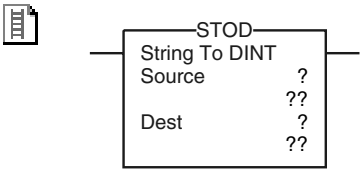
名称：	数据类型：	说明：	注意：
LEN	DINT	字符串中的字符数	每当执行以下操作时， LEN 都会自动更新为新的字符数： • 使用 String Browser （字符串浏览器）对话框输入字符 • 使用字符串读取、转换或操作指令 LEN 显示当前字符串的长度。 DATA 成员可能包含附加的或旧的字符，这些字符不包含在 LEN 计算中。
DATA	SINT 数组	字符串中的 ASCII 字符	• 要访问字符串中的字符，请引用标记的名称。例如，若要访问 *string_1* 标记中的字符，请输入 *string_1*。 • DATA 数组的每个元素包含一个字符。 • 您可以创建新的字符串数据类型，以存储更少或更多的字符。

字符串到 DINT (String To DINT, STOD)

STOD 指令将整数的 ASCII 表示转换为整数或 REAL 值。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
Source	string	标记	包含 ASCII 形式的值的标记	字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
Destination	SINT INT **DINT** REAL	标记	存储整数值的标记	如果 Source 值是浮点数，则该指令将仅仅转换数的非小数部分 （不考虑目标数据类型）。

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 Source 转换为整数并将结果放入 Destination 中。

- 该指令转换正数和负数。
- 如果 Source 字符串包含非数字字符， STOD 将转换第一组连续的数字：

  聚 t 该指令跳过所有值首控制字符和非数字字符 （数值前面的负号除外）。

  – 如果字符串包含多组以分隔符 （例如 /）分隔的数值，则该指令只转换第一组数值。

算术状态标志： 算术状态标志受影响。

错误条件：

类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	53	输出数值超出目标数据类型的限制。	二选一： • 减小 ASCII 值的大小。 • 为目标选用更大的数据类型。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	设置 S:C。 Destination 清零。 该指令转换 Source。 如果结果为零，则设置 S:Z。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 当设置了 *MV_read.EM* 时， STOD 指令将 *MV_msg* 中的第一组数字字符转换为整数值。该指令跳过开头控制字符 ($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;

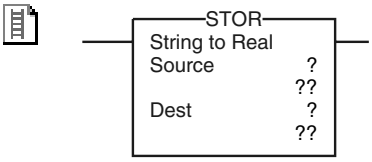
END_IF;
```

字符串到 REAL (String To REAL, STOR)

STOR 指令将浮点值的 ASCII 表示转换为 REAL 值。

操作数：

梯形图操作数



操作数：	类型：	格式：	输入：	说明：
Source	string	标记	包含 ASCII 形式的值的标记	字符串数据类型为： • 默认 STRING 数据类型 • 您创建的任何新的字符串数据类型
Destination	REAL	标记	存储 REAL 值的标记	

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 Source 转换为 REAL 值并将结果放入 Destination 中。

- 该指令转换正数和负数。
- 如果 Source 字符串包含非数字字符， STOR 将转换第一组连续数字，包括小数点 [.]：

  彙 t 该指令跳过所有值首控制字符和非数字字符 （数值前面的负号除外）。

  彙 t 如果字符串包含多组以分隔符 （例如 /）分隔的数字，则该指令只转换第一组数字。

算术状态标志： 算术状态标志受影响。

错误条件：

类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	53	输出数值超出目标数据类型的限制。	二选一： • 减小 ASCII 值的大小。 • 为目标选用更大的数据类型。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 为设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	设置 S:C。 Destination 清零。 该指令转换 Source。 如果结果为零，则设置 S:Z	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 从台秤读取了重量值后 （设置了 *weight_read.EM*）， STOR 指令将 *weight_ascii* 中的数字字符转换为 REAL 值。

您 《也许》会发现 Source 与 Destination 的小数部分有细微的差异。

梯形图



结构化文本

```
IF weight_read.EM THEN

     STOR(weight_ascii,weight);

     weight_read.EM := 0;

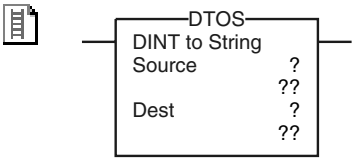
END_IF;
```

DINT 到字符串 (DINT to String, DTOS)

DTOS 指令生成值的 ASCII 表示。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
Source	SINT INT **DINT** REAL	标记	包含值的标记	如果 Source 是 REAL，该指令会将它转换为 DINT 值。请参见第 A-6 页的 “ REAL 转换为整数” 。
Destination	string	标记	用于存储 ASCII 值的标记	字符串数据类型为： • 默认 STRING 数据类型 • 您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 Source 转换为由 ASCII 字符组成的字符串并将结果放入 Destination 中。

算术状态标志： 不受影响

错误条件：

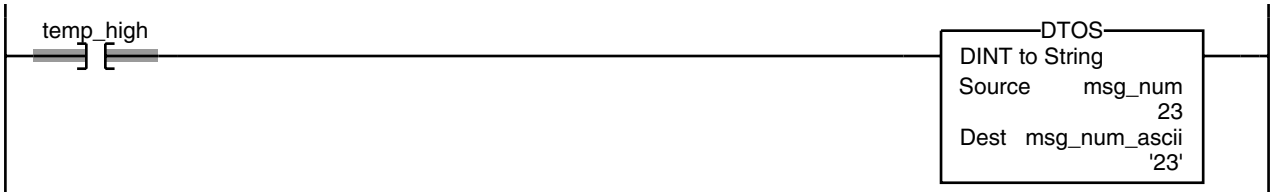
类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	52	输出字符串大于目标。	创建一个对输出字符串而言足够大的新的字符串数据类型。将新的字符串数据类型用作目标的数据类型。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	该指令转换 Source。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 当设置了 *temp_high* 时， DTOS 指令将 *msg_num* 中的值转换为由 ASCII 字符组成的字符串，并将结果放入 *msg_num_ascii* 中。后续梯级插入 *msg_num_ascii* 或将它与其他字符串连接以生成用于显示在终端上的完整消息。

梯形图



结构化文本

```
IF temp_high THEN

     DTOS(msg_num,msg_num_ascii);

     temp_high := 0;

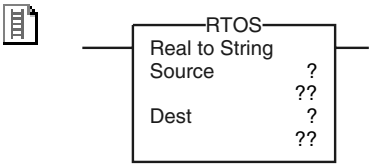
END_IF;
```

REAL 到字符串 (REAL to String, RTOS)

RTOS 指令生成 REAL 值的 ASCII 表示。

操作数：

梯形图



操作数：	类型：	格式：	输入：	说明：
Source	REAL	标记	包含 REAL 值的标记	
Destination	string	标记	用于存储 ASCII 值的标记	字符串数据类型为： • 默认 STRING 数据类型 • 您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 Source 转换为由 ASCII 字符组成的字符串并将结果放入 Destination 中。

算术状态标志： 不受影响

错误条件：

类型：	代码：	原因：	恢复方法：
4	51	字符串标记的 LEN 值大于字符串标记的 DATA 大小。	1. 检查以确保没有任何指令正在向字符串标记的 LEN 成员写入。 2. 在 LEN 值中，输入字符串所包含的字符数。
4	52	输出字符串大于目标。	创建一个对输出字符串而言足够大的新的字符串数据类型。将新的字符串数据类型用作目标的数据类型。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	该指令转换 Source。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 当设置了 *send_data* 时， RTOS 指令将 *data_1* 中的值转换为由 ASCII 字符组成的字符串，并将结果放入 *data_1_ascii* 中。后续梯级插入 *data_1_ascii* 或将它与其他字符串连接以生成用于显示在终端上的完整消息。

您 《也许》会发现 Source 与 Destination 的分数部分有细微的差异。

梯形图



结构化文本

```
IF send_data THEN

     RTOS(data_1,data_1_ascii);

     send_data := 0;

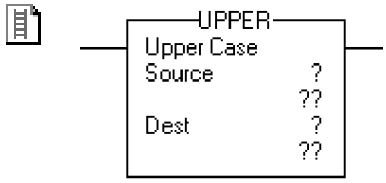
END_IF;
```

大写 (Upper Case, UPPER)

UPPER 指令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：
Source	string	标记	包含要转换为大写的字符的标记
Destination	string	标记	存储大写字符的标记

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 Source 中的所有字母都转换为大写并将结果放入 Destination 中。

- ASCII 字符串区分大小写。大写 “A” ($41) *不*等于小写 “a” ($61)。
- 如果操作员直接输入 ASCII 字符，请在比较之前将字符转换为全大写或全小写。

Source 字符串中的所有非字母字符都保持不变。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	该指令将 Source 转换为大写。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 为了查找有关特定项的信息，操作员向 ASCII 终端输入该项的目录编号。在控制器从终端 （设置了 *terminal_read.EM*）读取输入时，UPPER 指令将 *catalog_number* 中的字符全部转换为大写字符，并将结果存储在 *catalog_number_upper_case* 中。然后，后续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);

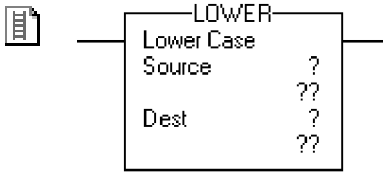
     terminal_read.EM := 0;

END_IF;
```

小写 (Lower Case, LOWER)

LOWER 指令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：
Source	string	标记	包含要转换为小写的字符的标记
Destination	string	标记	存储小写字符的标记

```
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 Source 中的所有字母转换为小写并将结果放入 Destination 中。

- ASCII 字符串区分大小写。大写 “A” ($41) *不*等于小写 “a” ($61)。
- 如果操作员直接输入 ASCII 字符，请在比较之前将字符转换为全大写或全小写。

Source 字符串中的所有非字母字符都保持不变。

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 false。	不采取任何操作。
梯级输入条件为 false	梯级输出条件设置为 false。	不可用
梯级输入条件为 true	指令执行。 梯级输出条件设置为 true。	不可用
EnableIn 处于设置状态	不可用	EnableIn 始终为设置状态。 指令执行。
指令执行	该指令将 Source 转换为小写。	
后期扫描	梯级输出条件设置为 false。	不采取任何操作。

示例： 为了查找有关特定项的信息，操作员向 ASCII 终端输入该项的编号。在控制器从终端 （设置了 *terminal_read.EM*）读取输入时， LOWER 指令将 *item_number* 中的字符全部转换为小写字符，并将结果存储 *item_number_lower_case* 中。然后，后续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;
```

说明：

常用属性

简介

本附录描述 Logix 指令的常用属性。

有关信息：	请参见页：
立即数	A-1
数据转换	A-1

立即数

当您以十进制格式 （例如 -2、 3）输入立即数 （常量）时，控制器使用 32 位来存储该值。如果您输入值时使用的基数不是十进制格式 （例如是二进制或十六进制），且未指定所有 32 位，则控制器在您未指定的位填入零 （填零）。

示例 立即数的填零

如果您输入：	则控制器存储：
-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

如果您在编程中混用数据类型，则会发生数据转换：

当使用以下对象编程时：	会在以下情况下发生转换：
梯形图逻辑	在一个指令中混用参数数据类型
功能块	连接两个具有不同数据类型的参数

如果指令的所有操作数使用以下类型，则指令执行速度更快且所需内存更少：

- 相同数据类型
- 最佳数据类型：
  - 在本手册的每个指令的 “操作数”部分中，粗体数据类型表示了最佳数据类型。
  - DINT 和 REAL 数据类型通常是最佳数据类型。
  - 大多数功能块指令只为其操作数支持一个数据类型 （最佳数据类型）。

如果您混用数据类型并使用非最佳数据类型的标记，控制器将根据以下规则转换数据

- 有操作数是 REAL 值吗？

如果：	则输入操作数 （例如 Source （源）、表达式中的标记、限制）转换为：
是	REAL
否	DINT

- 指令执行后，如果需要，结果 （DINT 或 REAL 值）将转换为目标数据类型。

不能在对整数或 REAL 数据类型执行操作的指令中指定 BOOL 标记。

由于数据转换需要花费额外的时间和内存，您可以通过以下方法提高程序效率：

- 在整个指令中使用同一数据类型
- 最大程度地减少 SINT 或 INT 数据类型的使用

换句话说，在指令中全部使用 DINT 标记或全部使用 REAL 标记加上立即数。

下面各部分解释在您使用 SINT 或 INT 标记或在混用数据类型时数据如何进行转换。

SINT 或 INT 转换为 DINT

对于将 SINT 或 INT 值转换为 DINT 值的指令，本手册的 “操作数” 部分标识了转换方法。

转换方法：	通过以下操作转换数据：
符号扩展	将最左位的值 （值的符号）填入现有位左侧每一位的位置，直到填满 32 位。
填零	将零填在现有位的左侧，直到填满 32 位

下面的示例显示使用符号扩展和填零来转换值的结果。

此值	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)

由于立即数总是填零，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 （源 A， INT）通过符号扩展转换，而 Source B （源 B，立即数）填零，因此比较结果为 false。



如果在通过符号扩展转换数据的指令中使用 SINT 或 INT 标记和立即数，请使用下列方法之一处理立即数：

- 以十进制基数指定任何立即数
- 如果输入值时使用的基数不是十进制数，请指定立即数的所有 32 位。为此，请将最左位的值输入其左侧每一位的位置，直到填满 32 位。
- 为每个操作数创建一个标记，并在整个指令中使用同一数据类型。要分配常量值，请执行下列任一操作：
  - 将其输入其中一个标记
  - 添加 MOV 指令，该指令将值移入其中一个标记。
- 使用 MEQ 指令只检查所需的位

下面的示例显示两种混用立即数和 INT 标记的方法。这两个示例都检查 1771 I/O 模块的位，以确定所有位是否启用。由于 1771 I/O 模块的输入数据字为 INT 标记，因此使用 16 位常量值最方便。

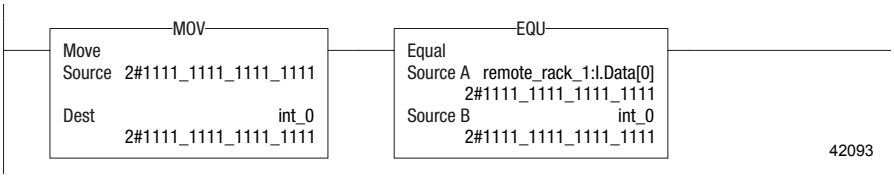
示例 混用 INT 标记和立即数

由于 remote_rack_1:I.Data[0] 是 INT 标记，因此检查它时所依据的值也作为 INT 标记输入。



示例 混用 INT 标记和立即数

由于 remote_rack_1:I.Data[0] 是 INT 标记，因此检查它时所依据的值首先移入 int_0（int_0 也是一个 INT 标记）。然后 EQU 指令再对这两个标记进行比较。



整数转换为 REAL

控制器以 IEEE 单精度浮点数格式存储 REAL 值。它使用一位表示值的符号， 23 位表示基本值，八位表示指数 （总共 32 位）。如果在同一指令的输入中混用整数标记 （SINT、 INT 或 DINT）和 REAL 标记，控制器会在指令执行之前将整数值转换为 REAL 值。

- SINT 或 INT 值始终转换为同一 REAL 值。
- DINT 值不能转换为同一 REAL 值：
  - REAL 值最多使用 24 位来表示基本值 （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 个以上有效位，则可能不转换为同一 REAL 值。如果它不需要 24 个以上有效位，则控制器四舍五入到使用 24 个有效位的最近的 REAL 值。

DINT 转换为 SINT 或 INT

为了将 DINT 值转换为 SINT 或 INT 值，控制器将 DINT 的高位部分截断，并设置溢出状态位 （如果需要的话）。下面的示例显示 DINT 转换为 SINT 或 INT 的结果。

示例 DINT 转换为 INT 和 DINT 转换为 SINT

此 DINT 值：	转换为此较小的值：	
16#0001_0081 (65,665)	INT：	16#0081 (129)
	SINT：	16#81 (-127)

REAL 转换为整数

为了将 REAL 值转换为整数值，控制器对小数部分进行四舍五入，并截断非小数部分的高位部分。如果数据丢失，则控制器设置溢出状态标志。数字按如下方式四舍五入：

- 非 x.5 的数四舍五入到最近的整数。
- X.5 四舍五入到最近的偶数。

下面的示例显示将 REAL 值转换为 DINT 值的结果。

示例 REAL 值转换为 DINT 值

此 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 算数状态标志是根据存储的值设置的。如果由于混用指令参数的数据类型而导致类型转换，则通常不影响算数状态关键字的指令可能也会产生影响。类型转换过程设置算数状态关键字。

功能块属性

简介

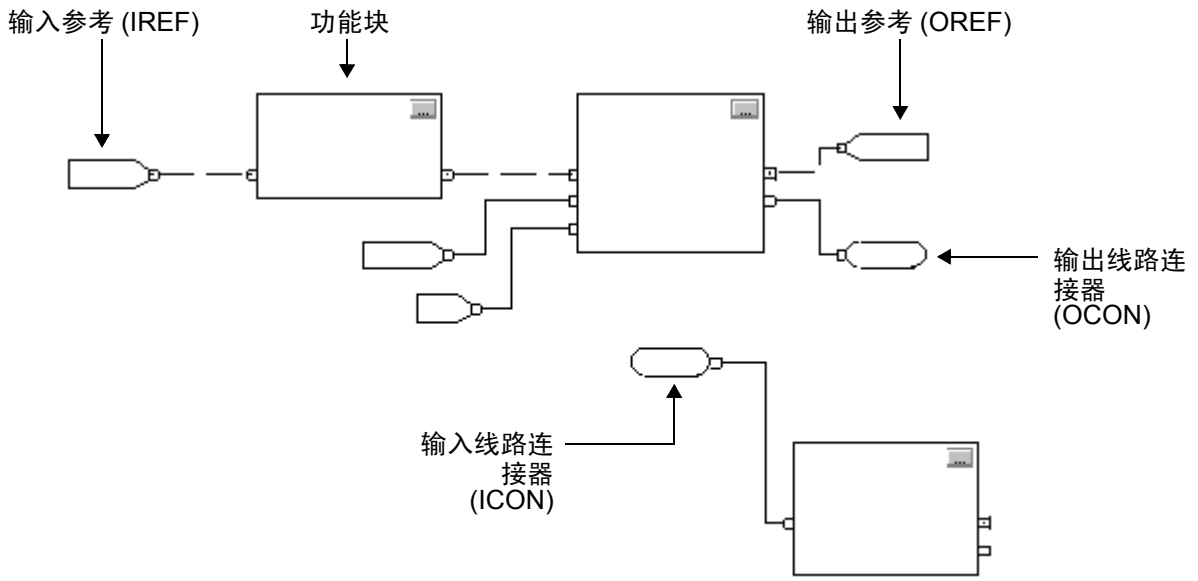
本附录描述功能块指令的特定问题。参阅本附录中的信息可帮助您了解功能块例程的运行方式。

重要事项

在功能块中编程时，请将工程单位的范围限制到 $+/-10^{+/-15}$，原因是内部浮点计算是使用单精度浮点数进行的。如果结果超出单精度浮点数的限制 ($+/-10^{+/-38}$)，此范围外的工程单位可能会有精度损失。

选择功能块元素

要控制设备，请使用下列元素：

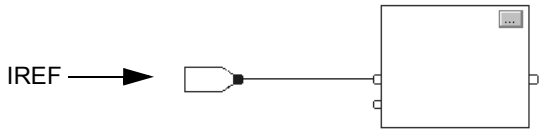


使用下表可选择功能块元素：

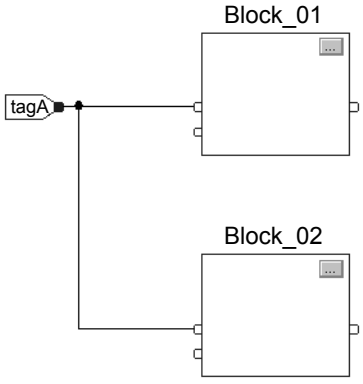
如果需要：	则使用：
从输入设备或标记提供值	输入参考 (IREF)
将值发送到输出设备或标记	输出参考 (OREF)
对一个或多个输入值执行操作，并生成一个或多个输出值	功能块
当数据为以下情况时在功能块之间传输数据： • 在同一表单上相距很远 • 在同一例程的不同表单上	输出线路连接器 (OCON) 和输入线路连接器 (ICON)
将数据分散到例程中的若干点	一个输出线路连接器 (OCON) 和多个输入线路连接器 (ICON)

锁存数据

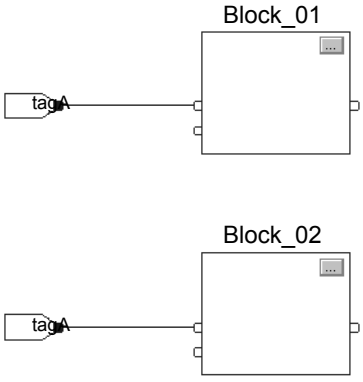
如果您使用 IREF 为功能块指令指定输入数据，则会为功能块例程的扫描锁存该 IREF 中的数据。 IREF 锁存来自程序范围标记和控制器范围标记的数据。控制器在每次扫描开始时更新所有 IREF 数据。



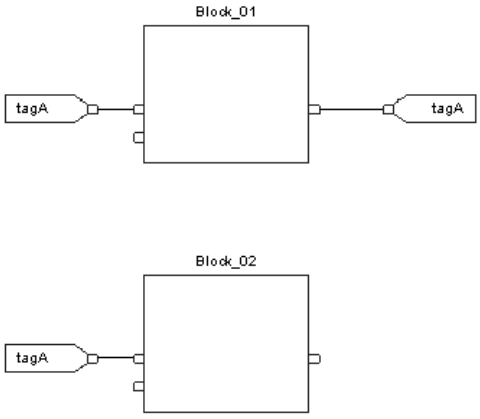
在下面的示例中，在例程开始执行时存储 tagA 的值。当 Block_01 执行时使用该存储值。当 Block_02 执行时也使用同一存储值。如果 tagA 的值在例程执行过程中更改， IREF 中 tagA 的存储值不会更改，直到例程下次执行为止。



下面的示例与上面的示例相同。只在例程开始执行时存储一次 tagA 的值。例程在整个例程中使用此存储值。



从 RSLogix 5000 软件版本 11 开始，您可以在同一例程中的多个 IREF 和一个 OREF 中使用同一标记。由于每次通过例程进行扫描时都锁存 IREF 中的标记的值，因此，即使在例程执行过程中 OREF 获得另一个标记值，所有 IREF 也会使用同一值。在下面的示例中，如果在例程开始执行此扫描时 tagA 的值为 25.4，而 Block_01 将 tagA 的值更改为 50.9，则在 Block_02 执行此扫描时，连接到 Block_02 的第二个 IREF 仍将使用值 25.4。在此例程中，任何 IREF 都不使用新 tagA 值 50.9，直到下次扫描开始为止。



执行顺序

当您执行下列操作时， RSLogix 5000 编程软件自动确定功能块在例程中的执行顺序：

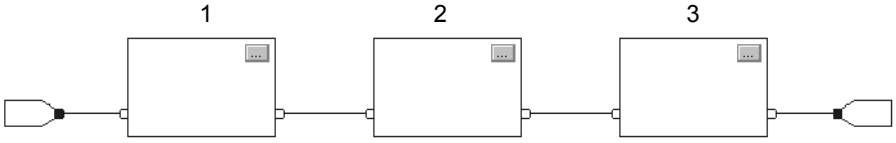
- 验证功能块例程
- 验证包含功能块例程的项目
- 下载包含功能块例程的项目

如有必要，可以通过将功能块连接在一起并指明所有反馈线路的数据流来定义执行顺序。

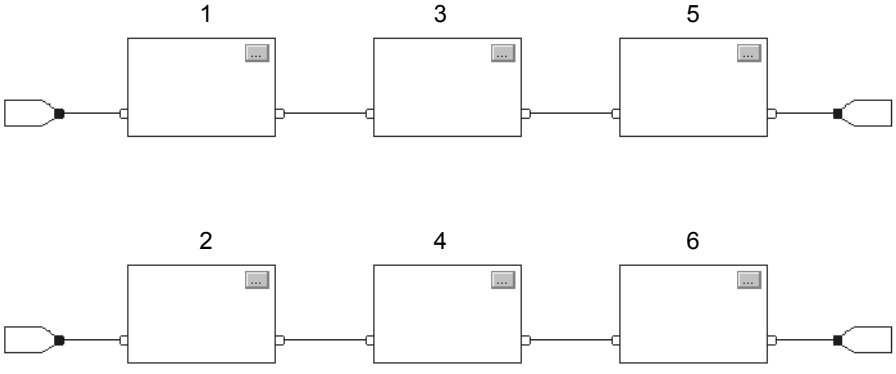
如果功能块不连接在一起，则哪一个块先执行并不重要。块之间没有数据流。



如果按顺序连接这些块，则执行顺序是从输入到输出。在控制器可以执行某块之前，该块的输入需要数据处于可用状态。例如，由于块 2 的输出作为块 3 的输入，因此块 2 必须在块 3 之前执行。

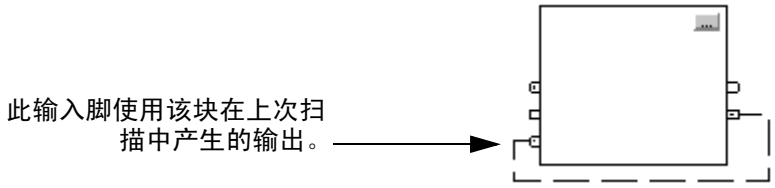


执行顺序只与连接在一起的块相关。下面的示例是正确的，因为两组块未连接在一起。特定组中的块按与该组中的块相关的相应顺序执行。

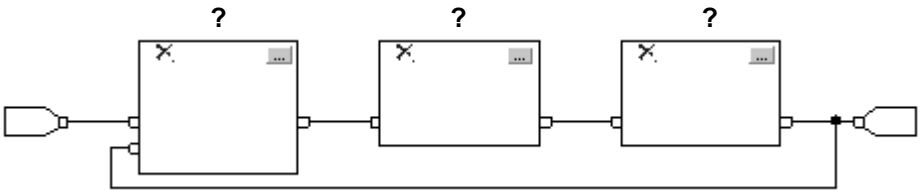


解析循环

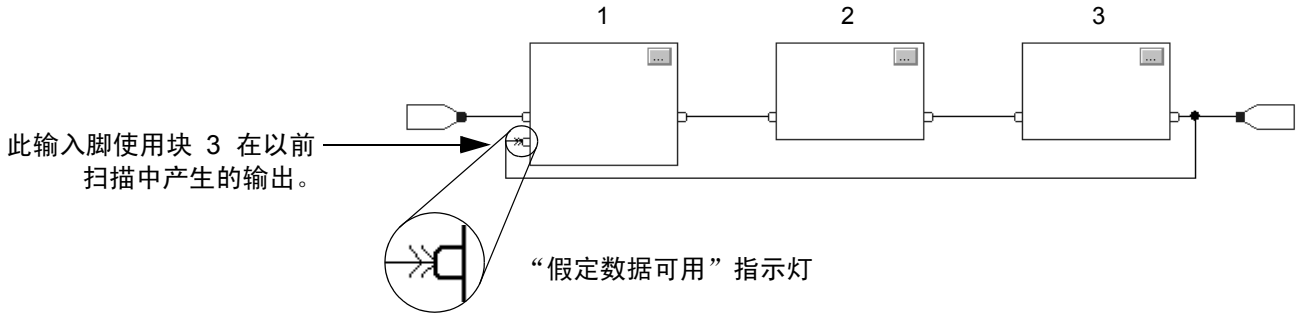
要围绕一个块创建反馈循环，请将该块的输出脚连接到同一块的输入脚。下面的示例是正确的。该循环只包含一个块，因此执行顺序并不重要。



如果一个循环中有一组块，则控制器无法确定哪一个块先执行。也就是说，它无法解析该循环。

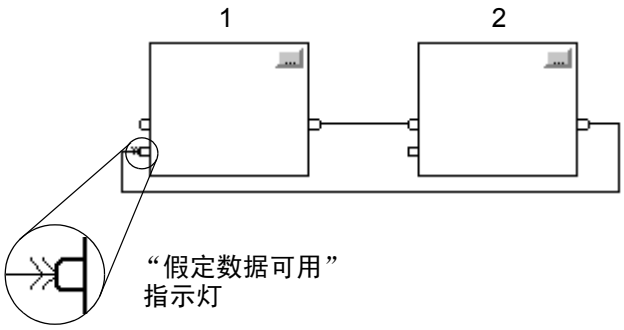
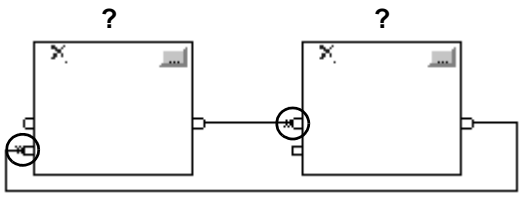


要标识哪一个块先执行，请使用 《假定数据可用》指示灯来标记创建该循环的输入线 （反馈线）。在下面的示例中，块 1 使用块 3 在以前例程执行中产生的输出。



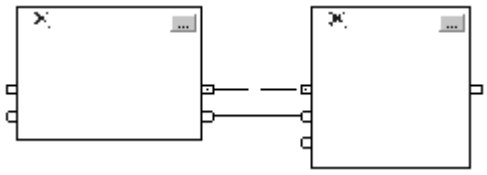
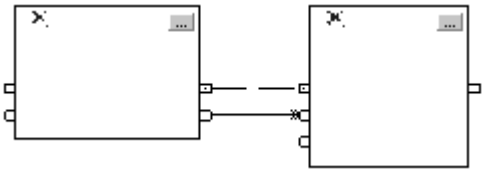
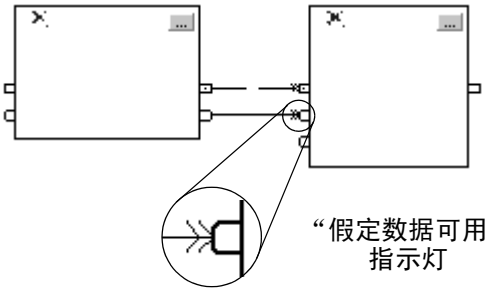
《假定数据可用》指示灯定义循环中的数据流。箭头表示该数据作为循环中第一个块的输入。

《不要》使用 《假定数据可用》指示灯来标记循环中的所有线路。

正常运行	不能正常运行
	
《假定数据可用》指示灯定义循环中的数据流。	控制器无法解析循环，因为所有线路都使用 《假定数据可用》指示灯。

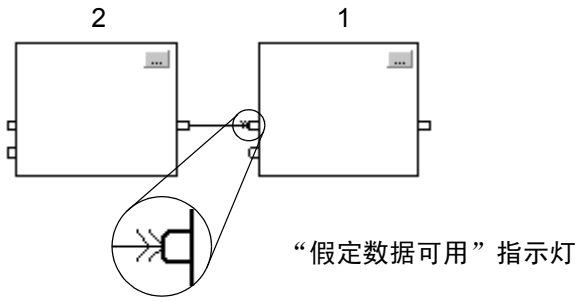
解析两个块之间的数据流

如果使用两条或更多条线来连接两个块，请对两个块之间的所有线路使用相同的数据流指示灯。

正常运行	不能正常运行
	
任何一条线都不使用 《假定数据可用》指示灯。	一条线使用 《假定数据可用》指示灯，而另一条线不使用。
	
两条线都使用 《假定数据可用》指示灯。	

创建一次扫描延迟

要在块之间生成一次扫描延迟，请使用 《假定数据可用》指示灯。在下面的示例中，块 1 先执行。它使用块 2 在上次例程扫描中产生的输出。



总结

概括而言，功能块例程按以下顺序执行：

1. 控制器获取 IREF 中的所有数据值。
2. 控制器按其他功能块连接方式所确定的顺序执行其他功能块。
3. 控制器将输出写入 OREF。

功能块对溢出情况的响应

通常而言，在发生溢出时，维护历史记录的功能块指令不使用 ± NAN 或 ± INF 值更新历史记录。每个指令对溢出情况做出下列响应之一：

响应 1：		响应 2：		响应 3：	
块执行其自己的算法，并检查结果以查找 ±NAN 或 ±INF。如果找到 ±NAN 或 ±INF，则块输出 ±NAN 或 ±INF。		带有输出限制的块执行其自己的算法，并检查结果以查找 ±NAN 或 ±INF。输出限制由 HighLimit 和 LowLimit 输入参数定义。如果找到 ±INF，则块输出受限的结果。如果找到 ±NAN，则不使用输出限制，块输出 ±NAN。		溢出情况不适用。这些指令通常具有一个布尔输出。	
ALM	NTCH	HLL		BAND	OSRI
DEDT	PMUL	INTG		BNOT	RESD
DERV	POSP	PI		BOR	RTOR
ESEL	RLIM	PIDE		BXOR	SETD
FGEN	RMPS	SCL		CUTD	TOFR
HPF	SCRV	SOC		D2SD	TONR
LDL2	SEL			D3SD	
LDLG	SNEG			DFF	
LPF	SRTP			JKFF	
MAVE	SSUM			OSFI	
MAXC	TOT				
MINC	UPDN				
MSTD					
MUX					

计时模式

下列过程控制和驱动器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	

有下列三种不同的计时模式：

计时模式：	**说明：**	
定期	定期模式是默认模式，适合于大多数控制应用。建议您将使用此模式的指令放入在定期任务中执行的例程中。指令的增量时间 (DeltaT) 按如下方式确定：	
	如果指令在以下任务中执行：	**则 DeltaT 等于：**
	定期任务	任务的周期
	事件任务或连续任务	自从上次执行以来经过的时间 控制器将经过的时间截断为整数毫秒 (ms)。例如，如果经过的时间 = 10.5 ms，则控制器设置 DeltaT = 10 ms。
	过程输入的更新需要与任务的执行同步，或者采样速度比任务执行的速度快 5-10 倍，以最大程度地减少输入和指令之间的采样错误。	
过采样	在过采样模式下，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，请转而使用实时采样模式。 将逻辑添加到程序中以控制指令何时执行。例如，通过使用指令的 EnableIn 输入，可以使用设置为 OversampleDeltaT 值的计时器控制执行。 过程输入的采样速度需要比指令的执行速度快 5-10 倍，以最大程度地减少输入与指令之间的采样错误。	
实时采样	在实时采样模式下，指令使用的增量时间 (DeltaT) 是与过程输入的更新对应的两个时间戳值之间的差。当过程输入具有与其更新关联的时间戳，并且您需要精确调整时，请使用此模式。 时间戳值是从为指令的 RTSTimeStamp 参数输入的标记名称读取的。通常，此标记名称是与过程输入关联的输入模块中的一个参数。 指令将经过配置的 RTSTime 值 （期望更新周期）与经过计算的 DeltaT 进行比较，以确定过程输入的每一更新是否正被指令读取。如果 DeltaT 不在配置时间的 1 毫秒内，则指令设置 RTSMissed 状态位，以指出读取模块输入的更新时出现问题。	

基于时间的指令需要一个常量值作为 DeltaT，以便控制算法正确计算过程输出。如果 DeltaT 变化，则过程输出中会出现不连续。不连续的严重性取决于指令和 DeltaT 变化的范围。如果存在以下情况，则会发生不连续：

- 在扫描过程中未执行指令。
- 在任务过程中多次执行了指令。
- 任务正在运行，而任务扫描率或过程输入的采样时间发生了更改。
- 在任务运行期间，用户更改了时间基本模式。
- 在任务运行期间，筛选器块中更改了 Order 参数。更改 Order 参数将在指令中选择另一种控制算法。

计时模式的通用指令参数

支持时间基本模式的指令具有下列输入和输出参数：

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 定期模式 1 过采样模式 2 实时采样模式 有效值 = 0 到 2 默认值 = 0 当 TimingMode = 0 且任务为定期任务时，启用定期计时，且 DeltaT 设置为任务扫描率。当 TimingMode = 0 且任务为事件或连续任务时，启用定期计时，且 DeltaT 设置为等于自从上次执行指令以来经过的时间跨度。 当 TimingMode = 1 时，启用过采样计时，且 DeltaT 设置为 OversampleDT 参数的值。 当 TimingMode = 2 时，启用实时采样计时，且 DeltaT 是从与输入关联的模块读取的当前和上次时间戳值之间的差。 如果 TimingMode 无效，则指令在 Status 中设置相应的位。
OversampleDT	REAL	过采样计时的执行时间。用于 DeltaT 的值以秒为单位。如果 TimingMode = 1，则 OversampleDT = 0.0 禁用控制算法的执行。如果无效，则指令设置 DeltaT = 0.0 并在 Status 中设置相应的位。 有效值 = 0 到 4194.303 秒 默认值 = 0.0

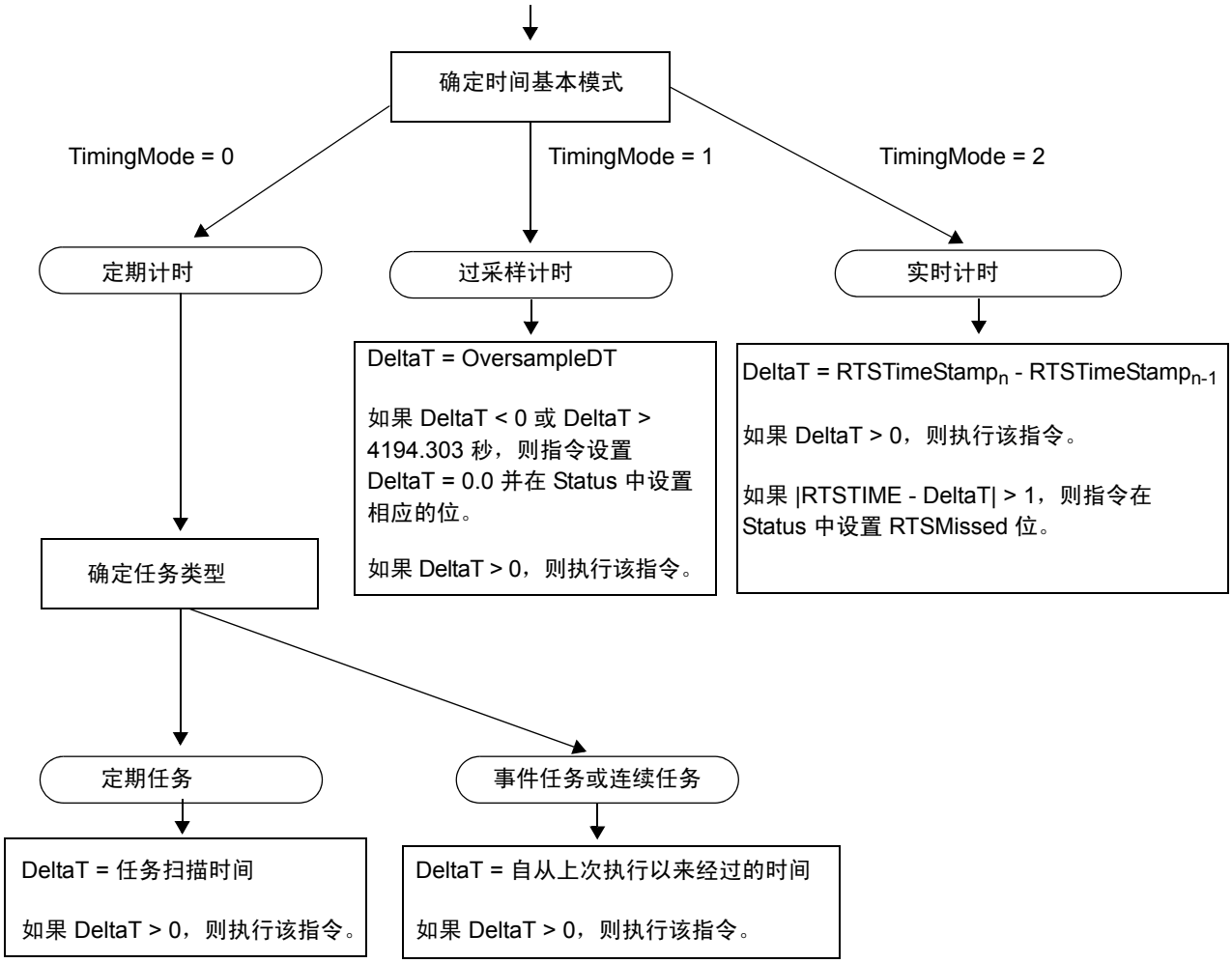
输入参数：	数据类型：	说明：
RTSTime	DINT	实时采样计时的模块更新周期。期望的 DeltaT 更新周期以毫秒为单位。更新周期通常是用于配置模块的更新时间的值。如果无效，则指令在 Status 中设置相应的位，并禁用 RTSMissed 检查。 有效值 = 1 到 32,767ms 默认值 = 1
RTSTimeStamp	DINT	实时采样计时的模块时间戳值。对应于输入信号的上次更新的时间戳值。此值用于计算 DeltaT。如果无效，则指令在 Status 中设置相应的位，禁用控制算法的执行，并禁用 RTSMissed 检查。 有效值 = 1 到 32,767ms （反之从 32767 到 0） 1 次计数 = 1 毫秒 默认值 = 0

输出参数

输出参数：	数据类型：	说明：
DeltaT	REAL	更新之间经过的时间。这是控制算法用来计算过程输出的经过时间 （以秒为单位）。 定期：如果任务为 “定期”任务，则 DeltaT = 任务扫描率；如果任务为 “事件”或 “连续”任务，则 DeltaT = 自从上次执行指令以来经过的时间 过采样：DeltaT = OversampleDT 实时采样：DeltaT = ($RTSTimeStamp_n$ - $RTSTimeStamp_{n-1}$)
状态	DINT	功能块状态。
TimingModeInv (Status.27)	BOOL	无效 TimingMode 值。
RTSMissed (Status.28)	BOOL	只在实时采样模式下使用。当 ABS | DeltaT - RTSTime | > 1 （.001 秒）时设置。
RTSTimeInv (Status.29)	BOOL	无效 RTSTime 值。
RTSTimeStampInv (Status.30)	BOOL	无效 RTSTimeStamp 值。
DeltaTInv (Status.31)	BOOL	无效 DeltaT 值。

计时模式概述

下图显示了指令如何确定相应的计时模式。



程序 / 操作员控制

某些指令支持 “程序 / 操作员”控制的概念。这些指令包括：

- 增强的选择 (Enhanced Select, ESEL)
- 累加器 (Totalizer, TOT)
- 增强的 PID (Enhanced PID, PIDE)
- 上升 / 保持 (Ramp/Soak, RMPS)
- 离散 2 状态设备 (Discrete 2-State Device, D2SD)
- 离散 3 状态设备 (Discrete 3-State Device, D3SD)

通过 “程序 / 操作员”控制，可以同时从用户程序和操作员接口设备控制这些指令。在 “程序”控制下，指令受指令的 “程序”输入控制；在 “操作员”控制下，指令受指令的 “操作员”输入控制。

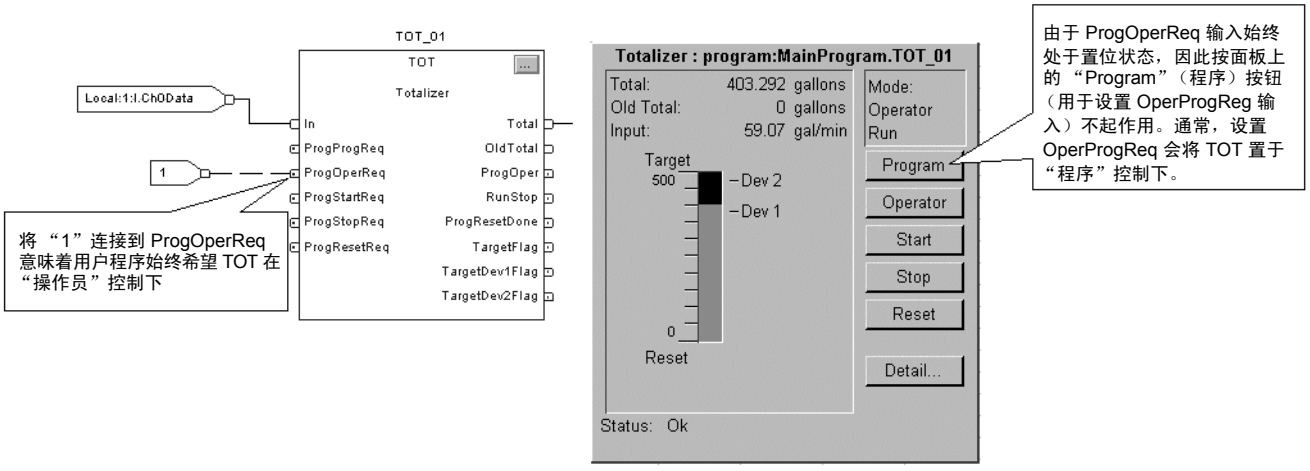
使用下列输入确定 “程序”或 “操作员”控制：

输入：	说明：
.ProgProgReq	要求进入 “程序”控制的程序请求。
.ProgOperReq	要求进入 “操作员”控制的程序请求。
.OperProgReq	要求进入 “程序”控制的操作员请求。
.OperOperReq	要求进入 “操作员”控制的操作员请求。

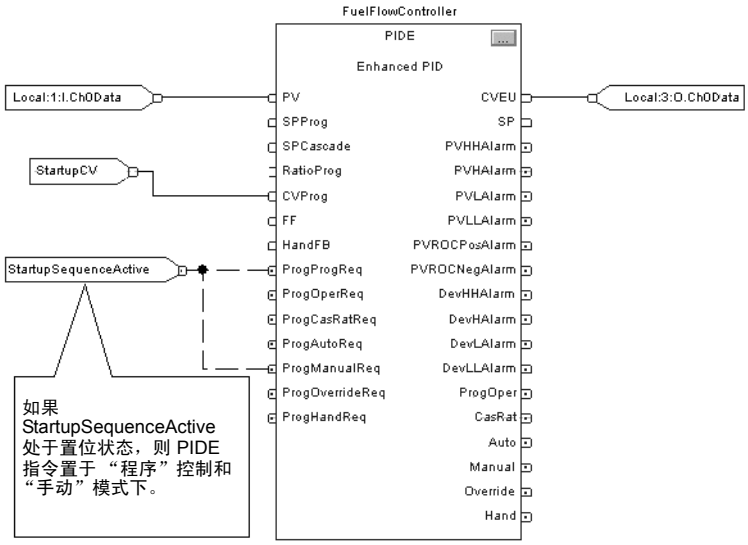
要确定指令是在 “程序”还是在 “操作员”控制下，请检查 ProgOper 输出。如果 ProgOper 处于置位状态，则指令在 “程序”控制下；如果 ProgOper 处于清零状态，则指令在 “操作员”控制下。

如果同时设置了两个输入请求位，则 “操作员”控制优先于 “程序”控制。例如，如果同时设置了 ProgProgReq 和 ProgOperReq，则指令进入 “操作员”控制。

“程序”请求输入优先于 “操作员”请求输入。这提供了使用 ProgProgReq 和 ProgOperReq 输入将指令 “锁定”在所需控制下的功能。例如，假定累加器 (Totalizer) 指令总是在 “操作员”控制中使用，且您的用户程序将不再控制累加器 (Totalizer) 的运行或停止。在这种情况下，您可以将文本值 1 连接到 ProgOperReq。这可以避免操作员通过从操作员接口设备设置 OperProgReq 而将累加器 (Totalizer) 置于 “程序”控制下。



同样，固定设置 ProgProgReq 会将指令 “锁定”在 “程序”控制下。在您希望程序控制指令的操作而不必担心操作员在控制指令中出现疏忽时，这对于自动启动序列尤其有用。在此示例中，您让程序在启动过程中设置 ProgProgReq 输入，一旦启动完成即对 ProgProgReq 输入清零。一旦对 ProgProgReq 输入清零，则指令一直保持在 “程序”控制下，直到收到更改请求。例如，操作员可以设置来自面板的 OperOperReq 输入，以接管对该指令的控制。下面的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。这使得只需设置所需模式请求位，操作员接口即可与这些指令一起工作。您不必对操作员接口进行编程来复位请求位。例如，如果操作员接口将 OperAutoReq 输入设置为 PIDE 指令，则当 PIDE 指令执行时，它确定相应的响应应该是什么，然后对 OperAutoReq 清零。

程序请求输入通常不被指令清零，因为它们通常作为输入连接到指令。如果指令对这些输入清零，则输入只需通过连接的输入即可再次获得设置。可能有这样的情况：您希望使用其他逻辑将 “程序”请求设置成由指令来对 “程序”请求清零。在此情况下，可以设置 ProgValueReset 输入，指令将总是在执行时对 “程序”模式请求清零。

在下面的示例中，另一个例程中的梯形梯级逻辑用于在按钮被按下时将 ProgAutoReq 单触发锁存到 PIDE 指令。由于 PIDE 指令自动对 “程序”模式请求清零，您不必写入任何梯形逻辑来在例程执行后对 ProgAutoReq 清零，在每次按钮被按下时， PIDE 指令将只收到一个进入 “自动”的请求。

当 TIC101AutoReq 按钮被按下时，将为 PIDE 指令 TIC101 单触发锁存 ProgAutoReq。TIC101 已配置成设置了 ProgValueReset 输入，所以当 PIDE 指令执行时，它自动对 ProgAutoReq 清零。



结构化文本编程

简介

本附录描述结构化文本编程的特定问题。参阅本附录中的信息可以帮助您了解结构化文本编程的执行方式。

有关信息：	请参见页：
结构化文本语法	C-1
赋值	C-2
表达式	C-4
指令	C-11
结构	C-12
注释	C-28

结构化文本语法

结构化文本是一种使用语句定义执行内容的文本编程语言。结构化文本不区分大小写。结构化文本可以包含以下组成部分：

术语：	定义：		示例：
赋值（请参见第 C-2 页）	使用赋值语句为标记赋值。 := 运算符是赋值运算符。 使用分号 “;” 结束赋值。		*tag := expression;*
表达式（请参见第 C-4 页）	表达式是完整赋值或结构语句的一部分。表达式的计算结果为一个数字 （数值表达式）或 true 或 false 状态（BOOL 表达式）。		
	表达式包含：		
	标记	用于存储数据的内存的命名区域 （BOOL、SINT、 INT、 DINT、 REAL、字符串）。	*value1*
	立即数	常量值。	*4*
	运算符	用于指定表达式中的运算的符号或助记符。	*tag1 + tag2* *tag1 >= value1*
	功能	当执行时，函数生成一个值。使用括号包含函数的操作数。 尽管语法类似，函数与指令仍有区别，函数只能在表达式中使用。指令不能在表达式中使用。	*function(tag1)*

术语：	定义：	示例：
指令 （请参见第 C-11 页）	指令是独立的语句。 指令使用括号包含其操作数。 取决于指令，可以有零个、一个或多个操作数。 当执行时，指令生成一个或多个作为数据结构一部分的值。 使用分号 “;” 结束指令。 尽管语法类似，指令与函数仍有区别，指令不能在表达式中使用。函数只能在表达式中使用。	*instruction();* *instruction(operand);* *instruction(operand1, operand2,operand3);*
结构 （请参见第 C-12 页）	用于触发结构化文本代码 （即其他语句）的条件语句。 使用分号 “;” 结束结构。	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
注释 （请参见第 C-28 页）	用于解释或阐明某一部分结构化文本的操作的文本。 • 使用注释可以使结构化文本的解释更加轻松。 • 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中的任何位置。	// 注释 (* 注释开始 . . . 注释结束 *) /* 注释开始 . . . 注释结束 */

赋值

使用赋值可以更改标记中存储的值。赋值的语法如下：

tag := *expression*;

其中：

项：	说明：	
tag	表示正在获取新值的标记 标记必须是 BOOL、 SINT、 INT、 DINT 或 REAL	
:=	是赋值符号	
expression	表示要赋给标记的新值	
	如果 *tag* 的数据类型为：	**则使用的表达式类型为：**
	BOOL	BOOL 表达式
	SINT INT DINT REAL	数值表达式
;	赋值结束	

tag 保持所赋的值，直到另一个赋值更改了该值。

表达式可以简单，例如可以是一个立即数或另一个标记名称；表达式也可以复杂，包括若干运算符和 / 或函数。有关详细信息，请参见下面的 “表达式”一节 （位于第 C-4 页上）。

指定非保持赋值

非保持赋值与上述常规赋值不同，非保持赋值中的标记在控制器每次处于以下状态时复位为零：

- 进入 RUN 模式
- 如果您配置 SFC 进行 《自动复位》，则离开 SFC 步骤；这仅在您将赋值操作嵌入步骤操作或在使用步骤操作通过 JSR 指令调用结构化文本例程的情况下才适用。

非保持赋值的语法如下：

tag [:=] *expression*;

其中：

项：	说明：	
tag	表示正在获取新值的标记 标记必须是 BOOL、 SINT、 INT、 DINT 或 REAL	
[:=]	为非保持赋值符号	
expression	表示要赋给标记的新值	
	如果 *tag* 的数据类型为：	**则使用的表达式类型为：**
	BOOL	BOOL 表达式
	SINT INT DINT REAL	数值表达式
;	赋值结束	

将 ASCII 字符赋值给字符串

使用赋值运算符可以将 ASCII 字符赋值给字符串标记的 DATA 成员的元素。要用字符赋值，请指定字符的值或指定标记名称、 DATA 成员和字符元素。例如：

以下赋值是正确的：	以下赋值*不*正确：
`string1.DATA[0]:= 65;`	`string1.DATA[0]:= A;`
`string1.DATA[0]:= string2.DATA[0];`	`string1 := string2;`

要将字符串添加或插入到字符串标记中，请使用下列 ASCII 字符串指令之一：

如果需要：	请使用指令：
将字符添加到字符串末尾	CONCAT
将字符插入字符串	INSERT

表达式

表达式是标记名称、等式或比较。要编写表达式，请使用下列任意内容：

- 存储值的标记名称 （变量）
- 直接输入表达式的数字 （立即数）
- 函数，例如：ABS、 TRUNC
- 运算符，例如：+、 -、 <、 >、 And、 Or

当编写表达式时，请遵循下列一般规则：

- 使用大小写字母的任意组合。例如，可以接受 “AND”的以下三种大小写变化：AND、 And、 and。
- 对于更复杂的要求，请使用括号将表达式组合在表达式中。这可以使整个表达式更易于阅读，并确保表达式按所需的顺序执行。请参见 “确定执行顺序”（位于第 C-10 页 上）。

在结构化文本中，使用两种表达式：

BOOL 表达式：生成 BOOL 值 1 (true) 或 0 (false) 的表达式。

- 布尔表达式使用布尔标记、关系运算符和逻辑运算符来比较值或检查条件为 true 还是为 false。例如， `tag1>65`。
- 简单的布尔表达式可以是一个 BOOL 标记。
- 通常，使用布尔表达式决定其他逻辑的执行。

数值表达式：计算整数或浮点值的表达式。

- 数值表达式使用算术运算符、算数函数和位运算符。例如，`tag1+5`。
- 通常，将数值表达式嵌套在布尔表达式中。例如，`(tag1+5)>65`。

使用下表为表达式选择运算符：

如果需要：	则：
计算算术值	“使用算术运算符和算术函数”（位于第 C-6 页上）。
比较两个值或字符串	“使用关系运算符”（位于第 C-7 页上）。
检查条件为 true 还是为 false	“使用逻辑运算符”（位于第 C-9 页上）。
比较值中的位	“使用位运算符”（位于第 C-10 页上）。

使用算术运算符和算术函数

可以在算术表达式中联合使用多个运算符和函数。

算术运算符对新值进行计算。

如果需要：	使用运算符：	最佳数据类型：
相加	+	DINT、 REAL
相减 / 求反	-	DINT、 REAL
相乘	*	DINT、 REAL
指数 （x 的 y 次方）	**	DINT、 REAL
相除	/	DINT、 REAL
取余	MOD	DINT、 REAL

算术函数执行数学运算。请为函数指定常量、非布尔标记或表达式。

对于：	使用函数：	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
从弧度求角度	DEG (*numeric_expression*)	DINT、 REAL
自然对数	LN (*numeric_expression*)	REAL
底为 10 的对数	LOG (*numeric_expression*)	REAL
从角度求弧度	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

例如：

使用格式：	示例：	
	对于以下情况：	编写：
value1 operator value2	如果 *gain_4* 和 *gain_4_adj* 是 DINT 标记，并且指定：“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* 是 DINT 标记，并且指定：“对 *high_alarm* 求反，并将结果存储在 *alarm* 中。”	`alarm:= -high_alarm;`
function(numeric_expression)	如果 *overtravel* 和 *overtravel_POS* 是 DINT 标记，并且指定：“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* 是 DINT 标记，*sensor1* 和 *sensor2* 是 REAL 标记，并且指定：“求出 *sensor1* 和 *sensor2* 的平均值的绝对值，与 *adjustment* 相加，并将结果存储在 *position* 中。”	`position := adjustment + ABS((sensor1 + sensor2)/2);`

使用关系运算符

关系运算符比较两个值或字符串，以提供 true 或 false 结果。关系运算的结果为 BOOL 值：

如果比较为：	结果为：
true	1
false	0

使用下列关系运算符：

对于以下比较：	使用运算符：	最佳数据类型：
等于	=	DINT、REAL、string
小于	<	DINT、REAL、string
小于或等于	<=	DINT、REAL、string
大于	>	DINT、REAL、string
大于或等于	>=	DINT、REAL、string
不等于	<>	DINT、REAL、string

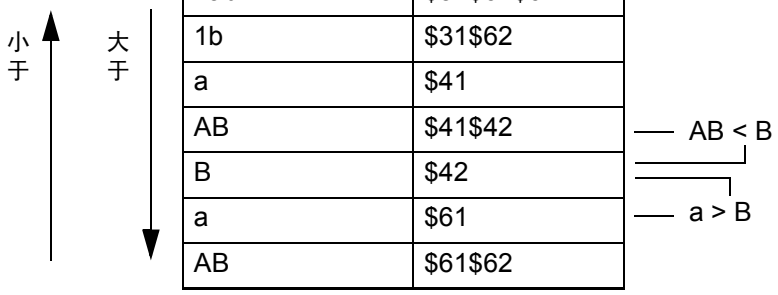
例如：

使用格式：	示例：	
	对于以下情况：	编写：
value1 operator value2	如果 *temp* 是 DINT 标记，并且指定：“如果 *temp* 小于 100°，则……”	`IF temp<100 THEN...`
stringtag1 operator stringtag2	如果 *bar_code* 和 *dest* 是字符串标记，并且指定：“如果 *bar_code* 等于 *dest*，则……”	`IF bar_code=dest THEN...`
char1 operator char2 要将 ASCII 字符直接输入表达式，请输入字符的十进制值。	如果 *bar_code* 是字符串标记，并且指定：“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 标记，*done* 是 BOOL 标记，并且指定“如果 *count* 大于或等于 *length*，则完成了计数。”	`done := (count >= length);`

如何计算字符串

ASCII 字符的十六进制值用于确定一个字符串是小于还是大于另一个字符串。

- 当两个字符串按在电话号码簿中的方式排序时，字符串的顺序决定了哪一个是较大的。



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

- 如果字符串的字符都一致，则字符串相等。
- 字符是区分大小写的。大写 “A” ($41) *不*等于小写 “a” ($61)。

有关字符的十进制值和十六进制代码，请参见本手册的封底。

使用逻辑运算符

使用逻辑运算符可以检查多个条件为 true 还是为 false。逻辑运算的结果为 BOOL 值：

如果比较为：	结果为：
true	1
false	0

使用下列逻辑运算符：

对于：	使用运算符：	数据类型：
逻辑与	&， AND	BOOL
逻辑或	OR	BOOL
逻辑异或	XOR	BOOL
逻辑求补	NOT	BOOL

例如：

使用格式：	示例：	
	对于以下情况：	编写：
BOOLtag	如果 *photoeye* 是 BOOL 标记，并且指定：“如果 *photoeye* 打开，则……”	`IF photoeye THEN...`
NOT *BOOLtag*	如果 *photoeye* 是 BOOL 标记，并且指定：“如果 *photoeye* 关闭，则……”	`IF NOT photoeye THEN...`
expression1 & *expression2*	如果 *photoeye* 是 BOOL 标记， *temp* 是 DINT 标记，并且指定： “如果 *photoeye* 打开且 *temp* 小于 100°，则……”。	`IF photoeye & (temp<100) THEN...`
expression1 OR *expression2*	如果 *photoeye* 是 BOOL 标记， *temp* 是 DINT 标记，并且指定： “如果 *photoeye* 打开或 *temp* 小于 100°，则……”。	`IF photoeye OR (temp<100) THEN...`
expression1 XOR *expression2*	如果 *photoeye1* 和 *photoeye2* 是 BOOL 标记，并且指定：“如果： • *photoeye1* 打开而 *photoeye2* 关闭 或者 • *photoeye1* 关闭而 *photoeye2* 打开 则……”	`IF photoeye1 XOR photoeye2 THEN...`
BOOLtag := *expression1* & *expression2*	如果 *photoeye1* 和 *photoeye2* 是 BOOL 标记， *open* 是 BOOL 标记，并且指定：“如果 *photoeye*1 和 *photoeye2* 都打开，则将 *open* 设置为 true”。	`open := photoeye1 & photoeye2;`

使用位运算符

位运算符根据两个值对一个值中的位进行操作。

对于：	使用运算符：	最佳数据类型：
按位与	&， AND	DINT
按位或	OR	DINT
按位异或	XOR	DINT
按位求补	NOT	DINT

例如：

使用格式：	示例：	
	对于以下情况：	编写：
value1 operator value2	如果 *input1*、*input2* 和 *result1* 是 DINT 标记，并且指定：“计算 *input1* 和 *input2* 的按位结果。将结果存储在 *result1* 中。”	`result1 := input1 AND input2;`

确定执行顺序

您写入表达式的运算按预定顺序执行，不一定是从左到右的顺序。

- 相同顺序的运算是从左到右执行的。
- 如果表达式包含多个运算符或函数，请用括号“( )”组合条件。这可以确保执行顺序正确，且表达式更容易阅读。

顺序：	运算：
1.	()
2.	函数 (…)
3.	**
4.	- （求反）
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。有关结构化文本中可用指令的列表，请参见本手册开始处的 “定位器表”。结构化文本指令在每次被扫描时执行。结构中的结构化文本指令在每当结构条件为 true 时执行。如果结构的条件为 false，则不扫描结构中的语句。没有触发执行的梯级条件或状态转换。

这与使用 EnableIn 触发执行的功能块不同。执行结构化文本指令时，相当于 EnableIn 始终处于置位状态。

这还与使用梯级输入条件触发执行的梯形图指令不同。某些梯形图指令只在梯级输入条件从 false 切换为 true 时执行。这些是转换的梯形图指令。在结构化文本中，指令在每次被扫描时执行，除非您预先限定了结构化文本指令的执行。

例如， ABL 指令是梯形图中的转换指令。在下面的示例中， ABL 指令仅在 *tag_xic* 从清零状态转换为置位状态时在扫描中执行操作。当 *tag_xic* 处于置位状态或 *tag_xic* 处于清零状态时， ABL 指令不执行。



在结构化文本中，如果将此示例写成：

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

则 ABL 指令将执行每一个设置了 *tag_xic* 的扫描，而不仅是在 *tag_xic* 从清零状态转换为置位状态时执行。

如果您希望 ABL 指令仅在 *tag_xic* 从清零状态转换为置位状态时执行，则必须限定结构化文本指令。使用单触发来触发执行。

```
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，也可以嵌套在其他结构中。

如果需要：	使用以下结构：	在以下语言中可用：	请参见页：
如果或当特定条件发生时执行某种操作	IF...THEN	结构化文本	C-13
根据数值选择要执行的操作	CASE...OF	结构化文本	C-16
在执行其他任何操作之前，将某种操作执行特定次	FOR...DO	结构化文本	C-19
只要特定条件为 true，则一直执行某种操作	WHILE...DO	结构化文本	C-22
一直执行某种操作，直到某条件为 true	REPEAT...UNTIL	结构化文本	C-25

某些关键字保留供将来使用

下面的结构不可用：

- GOTO
- REPEAT

RSLogix 5000 软件不允许您将它们作为标记名称或结构使用。

IF...THEN

使用 IF…THEN 在发生特定条件时执行某种操作。

操作数：



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

结构化文本

操作数：	类型：	格式：	输入：
bool_ expression	BOOL	标记 表达式	计算结果为 BOOL 值 （BOOL 表达式）的 BOOL 标记或表达式

说明： 语法为：

```
IF bool_expression1 THEN
      <statement>;          ◄── 当 bool_expression1 为 true 时要执行的语句
                .
                .
                .
可选 ┐ ELSIF bool_expression2 THEN
      <statement>;          ◄── 当 bool_expression2 为 true 时要执行的语句
                .
                .
                .
可选 ┐ ELSE
      <statement>;          ◄── 当两个表达式都为 false 时要执行的语句
                .
                .
                .
END_IF;
```

要使用 ELSIF 或 ELSE，请遵循下列准则：

1. 要从语句的一些可能组中选择，请添加一个或多个 ELSIF 语句。
   - 每个 ELSIF 代表一个可选路径。
   - 指定所需数量的 ELSIF 路径。
   - 控制器执行第一个为 true 的 IF 或 ELSIF，跳过其余的 ELSIF 和 ELSE。
2. 要在所有 IF 或 ELSIF 条件为 false 时执行某种操作，请添加 ELSE 语句。

下表汇总了 IF、 THEN、 ELSIF 和 ELSE 的不同组合。

如果需要：	且：	则使用以下结构：
如果或当条件为 true 时执行某种操作	如果条件为 false，则不执行任何操作	IF…THEN
	如果条件为 false，则执行其他操作	IF…THEN…ELSE
根据输入条件从可选语句（或语句组）中选择	如果条件为 false，则不执行任何操作	IF…THEN…ELSIF
	如果所有条件为 false，则分配默认语句	IF…THEN…ELSIF…ELSE

示例 1： IF…THEN

如果希望：	输入结构化文本：
如果拒绝次数 > 3，则	`IF rejects > 3 THEN`
传送带 = 关闭 (0)	`conveyor := 0;`
警报 = 打开 (1)	`alarm:= 1;`
	`END_IF;`

示例 2： IF…THEN…ELSE

如果希望：	输入结构化文本：
如果传送带方向接触面 = 向前 (1)，则	`IF conveyor_direction THEN`
指示灯 = 关闭	`light := 0;`
否则指示灯 = 打开	`ELSE`
	`light [:=] 1;`
	`END_IF;`

[:=] 告知控制器在控制器处于以下状态时对 *light* 清零：

- 进入 RUN 模式
- 离开 SFC 步骤 （如果您配置 SFC 进行 《自动复位》），这仅在您将赋值操作嵌入步骤操作或在使用步骤操作通过 JSR 指令调用结构化文本例程的情况下才适用。

示例 3： IF…THEN…ELSIF

如果希望：	输入结构化文本：
如果糖度低限开关 = 低 （打开）且糖度高限开关 = 非高 （打开），则	`IF Sugar.Low & Sugar.High THEN`
入口阀 = 打开 （打开）	`Sugar.Inlet [:=] 1;`
直到糖度高限开关 = 高 （关闭）	`ELSIF NOT(Sugar.High) THEN`
	`Sugar.Inlet := 0;`
	`END_IF;`

[:=] 告知控制器在控制器处于以下状态时对 *Sugar.Inlet* 清零：

- 进入 RUN 模式
- 如果您配置 SFC 进行 《自动复位》，则离开 SFC 步骤；这仅在您将赋值操作嵌入步骤操作或在使用步骤操作通过 JSR 指令调用结构化文本例程的情况下才适用。

示例 4： IF…THEN…ELSIF…ELSE

如果希望：	输入结构化文本：
如果箱体温度 > 100	`IF tank.temp > 200 THEN`
则泵 = 慢	`pump.fast :=1; pump.slow :=0; pump.off :=0;`
如果箱体温度 > 200	`ELSIF tank.temp > 100 THEN`
则泵 = 快	`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 可以根据数值选择要执行的操作。

操作数：

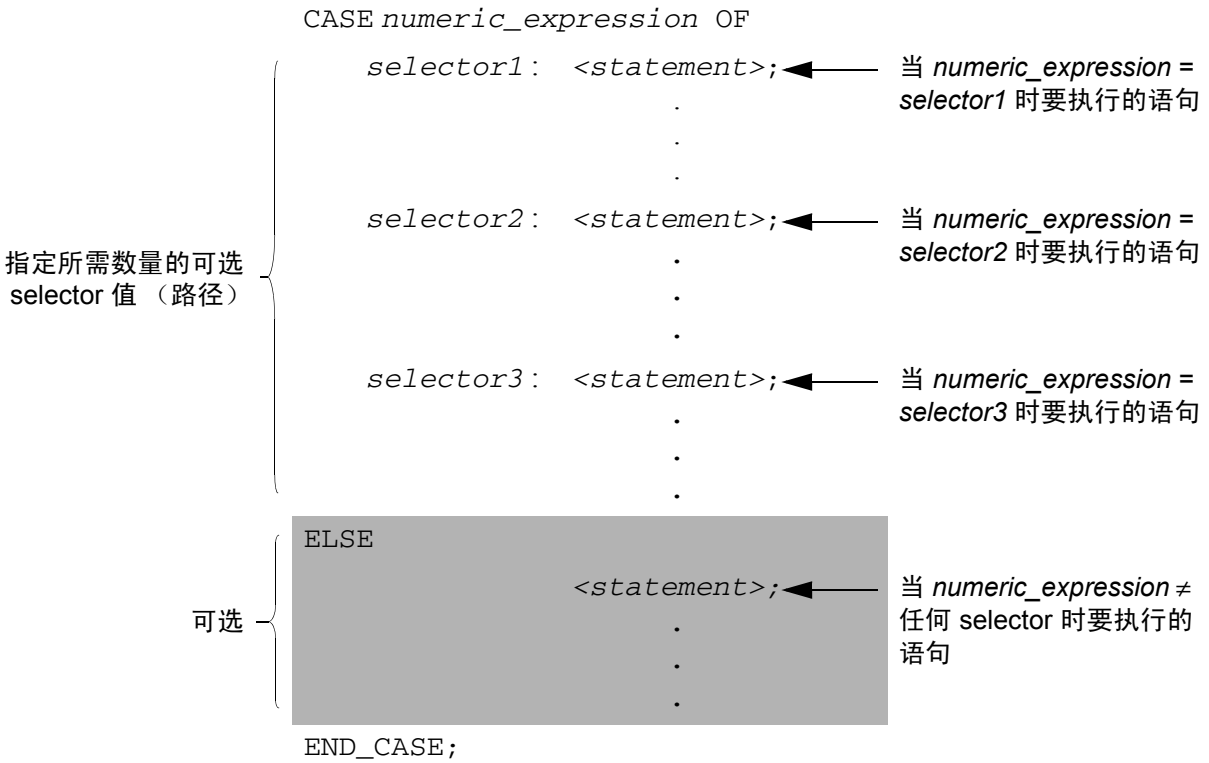
结构化文本

```
CASE numeric_expression OF
     selector1: statement;
     selectorN: statement;
ELSE
     statement;
END_CASE;
```

操作数：	类型：	格式：	输入：
numeric_expression	SINT INT DINT REAL	标记 表达式	计算结果为一个数字 （数值表达式）的标记或表达式
selector	SINT INT DINT REAL	立即数	与 *numeric_expression* 的类型相同

重要事项 如果使用 REAL 值，请对 selector 使用某一范围的值，因为 REAL 值更有可能处于某一范围的值中，而不是一个特定值的精确匹配。

说明： 语法为：



有关有效的 selector 值，请参见下一页中的表。

输入 selector 值的语法为：

当 selector 为：	输入：
一个值	*value*: *statement*
多个不同的值	*value1*, *value2*, *valueN* : <*statement*> 使用逗号 (,) 分隔每个值。
某一范围的值	*value1*..*valueN* : <*statement*> 使用两个句点 (..) 标识范围。
不同的值加上某一范围的值	*valuea, valueb, value1*..*valueN* : <*statement*>

CASE 结构与 C 或 C++ 编程语言中的开关语句类似。但是，使用 CASE 结构，控制器*只*执行与 《第一个匹配》的 selector 值关联的语句。执行 《始终在该选择器的语句后中断》并进入 END_CASE 语句。

示例

如果希望：	输入结构化文本：
如果处方编号 = 1，则	`CASE recipe_number OF`
成份 A 出口 1 = 打开 (1)	`1: Ingredient_A.Outlet_1 :=1;`
Ingredient B outlet 4 = open (1)	`Ingredient_B.Outlet_4 :=1;`
如果处方编号 = 2 或 3，则	`2,3: Ingredient_A.Outlet_4 :=1;`
Ingredient A outlet 4 = open (1)	`Ingredient_B.Outlet_2 :=1;`
Ingredient B outlet 2 = open (1)	
如果处方编号 = 4、 5、 6 或 7，则	`4..7: Ingredient_A.Outlet_4 :=1;`
Ingredient A outlet 4 = open (1)	`Ingredient_B.Outlet_2 :=1;`
Ingredient B outlet 2 = open (1)	
如果处方编号 = 8、 11、 12 或 13，则	`8,11..13 Ingredient_A.Outlet_1 :=1;`
Ingredient A outlet 1 = open (1)	`Ingredient_B.Outlet_4 :=1;`
Ingredient B outlet 4 = open (1)	
否则所有出口 = 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;`

[:=] 告知控制器还在控制器处于以下状态时对出口标记清零：

- 进入 RUN 模式
- 如果您配置 SFC 进行 《自动复位》，离开 SFC 步骤；这仅在您将赋值操作嵌入步骤操作或在使用步骤操作通过 JSR 指令调用结构化文本例程的情况下才适用。

FOR···DO

使用 FOR···DO 循环可以在执行任何其他操作之前，将某种操作执行特定次。

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：
count	SINT INT DINT	标记	当 FOR···DO 执行时用于存储计数位置的标记
initial_value	SINT INT DINT	标记 表达式 立即数	计算结果必须为一个数字 指定计数的初始值
final_value	SINT INT DINT	标记 表达式 立即数	指定计数的最终值，用于确定何时退出循环
increment	SINT INT DINT	标记 表达式 立即数	（可选）每次通过循环后计数递增的量 如果您没有指定递增量，则计数按 1 递增。

重要事项

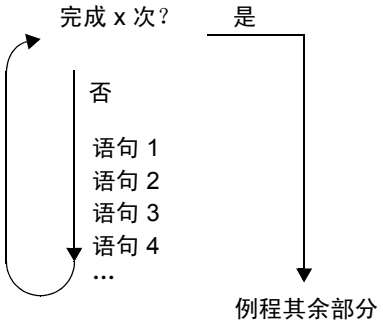
确保单次扫描中 《不》在循环中迭代多次。

- 在完成循环之前，控制器 《不》执行例程中的任何其他语句。
- 如果完成循环的时间大于任务的看门狗定时器，则会出现严重错误。
- 考虑使用其他结构，如 IF...THEN。

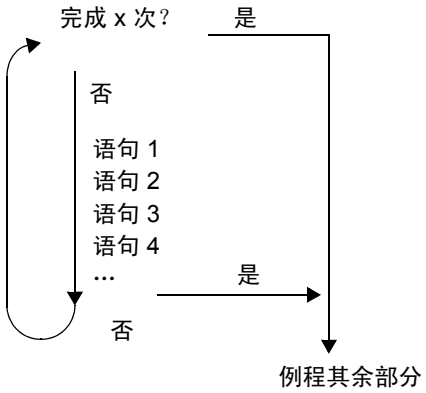
说明： 语法为：

```
FOR count := initial_value
   TO final_value
可选 { BY increment                    如果您没有指定递增量，则循环按 1 递增。
   DO
   <statement>;
可选 { IF bool_expression THEN
          EXIT;                        如果需要提前退出循环，请使用其他语句，如 IF...THEN 结构，作为 EXIT 语句的条件。
       END_IF;
END_FOR;
```

下图显示 FOR...DO 循环如何执行以及 EXIT 语句如何提前离开循环。



FOR…DO 循环执行特定次数。



要在计数达到最后值之前停止循环，请使用 EXIT 语句。

示例 1：

如果希望：	输入结构化文本：

对 BOOL 数组中的 0 - 31 位清零：
1. 将 *subscript* 标记初始化为 0。
2. 对 *array[subscript]* 清零。例如，当 *subscript* = 5 时，对 *array[5]* 清零。
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：

如果希望：	输入结构化文本：

用户自定义的数据类型 （结构）存储有关您的库存项目的以下信息：
• 项目的条形码 ID （字符串数据类型）
• 项目的库存数目 （DINT 数据类型）
以上结构的数组为每个不同库存项目包含一个元素。您想要搜索该数组以查找特定产品（使用其条码）并确定库存量。
1. 获取 *Inventory* 数组的大小 （项目数目），并将结果存储在 *Inventory_Items* （DINT 标记）中。
2. 将 *position* 标记初始化为 0。
3. 如果 *Barcode* 与数组中某项目的 ID 匹配，则：
a.设置 *Quantity* 标记 = *Inventory[position].Qty*。这将生成该项目的库存量。
b.停止。
Barcode 是存储您搜索的项目的条形码的字符串标记。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。由于元素编号从 0 开始，最后一个元素的编号比数组元素数小 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 循环可以在某些条件为 true 的情况下一直执行某种操作。

操作数：



```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

结构化文本

操作数：	类型：	格式：	输入：
bool_ expression	BOOL	标记 表达式	计算结果为 BOOL 值的 BOOL 标记或表达式

重要事项

确保单次扫描中 《不》在循环中迭代多次。

- 在完成循环之前，控制器 《不》执行例程中的任何其他语句。
- 如果完成循环的时间大于任务的看门狗定时器，则会出现严重错误。
- 考虑使用其他结构，如 IF...THEN。

说明： 语法为：

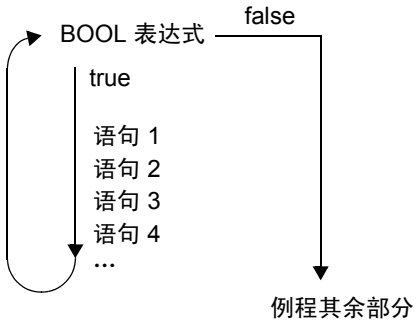
```
WHILE bool_expression1 DO
   <statement>;          <-- 当 bool_expression1 为 true 时要执行的语句

   IF bool_expression2 THEN
     EXIT;               <-- 如果需要提前退出循环，请使用其他语句，如 IF...THEN 结构，作为 EXIT 语句的条件。
   END_IF;

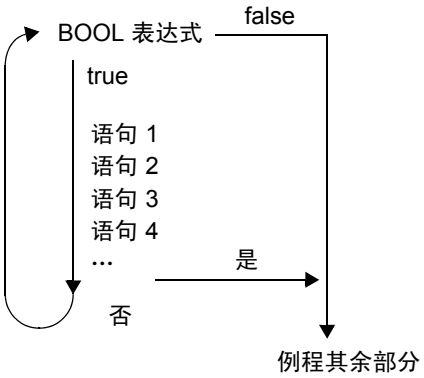
END_WHILE;
```

可选

下图显示 WHILE...DO 循环如何执行和 EXIT 语句如何提前离开循环。



当 *bool_expression* 为 true 时，控制器只执行 WHILE…DO 循环中的语句。



要在条件为 true 之前停止循环，请使用 EXIT 语句。

示例 1：

如果希望：	输入结构化文本：

WHILE...DO 循环先计算其条件。如果条件为 true，控制器则执行循环中的语句。
这与 REPEAT...UNTIL 循环不同，因为 REPEAT...UNTIL 循环先执行结构中的语句，然后在再次执行语句之前确定条件是否为 true。REPEAT...UNTIL 循环中的语句总是至少执行一次。WHILE...DO 循环中的语句可能永远不会执行。
	`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：

如果希望：	输入结构化文本：

将 ASCII 字符从 SINT 数组移动到字符串标记中。（在 SINT 数组中，每个元素都保存一个字符。）到达回车符时停止。
1. 将 *Element_number* 初始化为 0。
2. 计算 *SINT_array*（包含 ASCII 字符的数组）中元素的数目，将结果存储在 *SINT_array_size*（DINT 标记）中。
3. 如果 *SINT_array[element_number]* 处的字符 = 13（回车的十进制值），则停止。
4. 设置 *String_tag[element_number]* = *SINT_array[element_number]* 处的字符。
5. 将 *element_number* 加 1。这样，控制器检查 *SINT_array* 中的下一个字符。
6. 设置 *String_tag* 的 Length 成员 = *element_number*。（此成员记录 *String_tag* 中目前的字符数。）
7. 如果 *element_number* = *SINT_array_size*，则停止。（您位于数组末尾，数组不包含回车符。）
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 循环可以一直执行某种操作，直到条件为 true。

操作数：



```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

结构化文本

操作数：	类型：	格式：	输入：
bool_ expression	BOOL	标记 表达式	计算结果为 BOOL 值 （BOOL 表达式）的 BOOL 标记或表达式

重要事项

确保单次扫描中 《不》在循环中迭代多次。

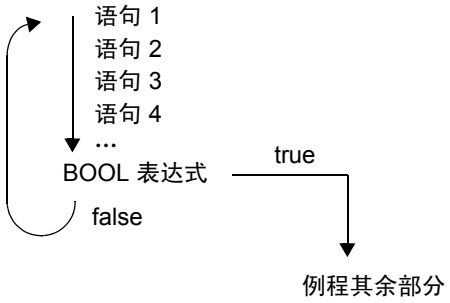
- 在完成循环之前，控制器 《不》执行例程中的任何其他语句。
- 如果完成循环的时间大于任务的看门狗定时器，则会出现严重错误。
- 考虑使用其他结构，如 IF...THEN。

说明： 语法为：

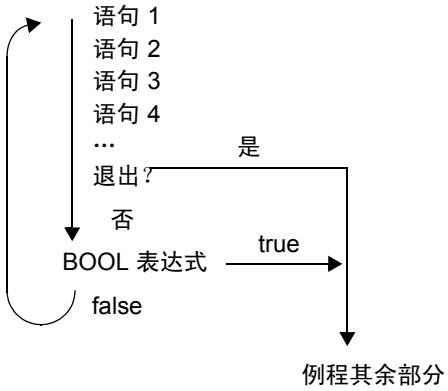
```
REPEAT
   <statement>;          ◄── 当 bool_expression1 为 false 时要执行的语句
   IF bool_expression2 THEN
     EXIT;               ◄── 如果需要提前退出循环，请使用其他语句，如 IF...THEN 结构，作为 EXIT 语句的条件。
   END_IF;
UNTIL bool_expression1
END_REPEAT;
```

可选

下图显示 REPEAT...UNTIL 循环如何执行和 EXIT 语句如何提前离开循环。



当 *bool_expression* 为 false 时，控制器只执行 REPEAT…UNTIL 循环中的语句。



要在条件为 false 之前停止循环，请使用 EXIT 语句。

示例 1：

如果希望：	输入结构化文本：

REPEAT...UNTIL 循环执行结构中的语句，然后在再次执行语句之前确定条件是否为 true。
 	
这与 WHILE...DO 循环不同，因为 WHILE...DO 循环先计算其条件。如果条件为 true，控制器则执行循环中的语句。 REPEAT...UNTIL 循环中的语句总是至少执行一次。 WHILE...DO 循环中的语句可能永远不会执行。
	`pos := -1;`
	`REPEAT`
	`  pos := pos + 2;`
	`UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))`
	`END_REPEAT;`

示例 2：

如果希望：	输入结构化文本：

将 ASCII 字符从 SINT 数组移动到字符串标记中。（在 SINT 数组中，每个元素都保存一个字符。）到达回车符时停止。
1. 将 *Element_number* 初始化为 0。
2. 计算 *SINT_array*（包含 ASCII 字符的数组）中元素的数目，将结果存储在 *SINT_array_size*（DINT 标记）中。
3. 设置 *String_tag[element_number]* = *SINT_array[element_number]* 处的字符。
4. *element_number* 加 1。这样，控制器检查 *SINT_array* 中的下一个字符。
5. 设置 *String_tag* 的 Length 成员 = *element_number*。（此成员记录 *String_tag* 中目前的字符数。）
6. 如果 *element_number* = *SINT_array_size*，则停止。（您位于数组末尾，数组不包含回车符。）
7. 如果 *SINT_array[element_number]* 处的字符 = 13（回车的十进制值），则停止。否则转到 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

要使您的结构化文本更易于解释，请向其添加注释。

- 通过注释，您可以用简单的语言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。

要向结构化文本添加注释，请执行下列操作：

在以下位置添加注释：	请使用下列格式之一：
在单独一行中	// 注释
在一行结构化文本的末尾	(* 注释 *) /* 注释 */
在一行结构化文本之中	(* 注释 *) /* 注释 */
跨多个行	(* 注释开始 . . . 注释结束 *) /* 注释开始 . . . 注释结束 */

例如：

格式：	示例：

// 注释	**在行的开头。**
	// 检查传送带方向
	`IF conveyor_direction THEN...`
	
	在行的末尾。
	`ELSE` // 如果传送带不移动，则设置警报指示灯
	`light := 1;`
	`END_IF;`

(* 注释 *)	`Sugar.Inlet[:=]1;`(* 打开入口 *)
	
	`IF Sugar.Low` (* 低级别 LS*)& Sugar.High (* 高级别 LS*) `THEN...`
	
	（* 控制循环泵的速度。该速度取决于箱体的温度。 *）
	`IF tank.temp > 200 THEN...`

/* 注释 */	`Sugar.Inlet:=0;`/* 关闭入口 */
	
	`IF bar_code=65 /*A*/ THEN...`
	
	/* 获取库存数组中的元素数目，将值存储在 Inventory_Items 标记中 */
	`SIZE(Inventory,0,Inventory_Items);`

A

ABL 指令 16-5
ABS 指令 5-29
ACB 指令 16-8
ACL 指令 16-10
ACS 指令 13-14
ADD 指令 5-6
AFI 指令 10-23
AHL 指令 16-12
AND 指令 6-23
ARD 指令 16-16
ARL 指令 16-19
ASCII
结构化文本赋值 C-4
ASCII 读取 16-16
ASCII 读取行 16-19
ASCII 清除缓冲区 16-10
ASCII 握手行 16-12
ASCII 写入 16-28
ASCII 写入附加 16-23
ASCII 指令
ABL 16-5
ACB 16-8
ACL 16-10
AHL 16-12
ARD 16-16
ARL 16-19
AWA 16-23
AWT 16-28
CONCAT 17-3
DELETE 17-5
DTOS 18-8
FIND 17-7
INSERT 17-9
LOWER 18-14
MID 17-11
RTOS 18-10
STOD 18-4
STOR 18-6
SWPB 6-19
UPPER 18-12
ASN 指令 13-11
ATN 指令 13-17
AVE 指令 7-38
AWA 指令 16-23
AWT 指令 16-28

B

BAND 6-35
BNOT 6-44
BOOL 表达式
结构化文本 C-4
BOR 6-38
BRK 指令 11-5
BSL 指令 8-2
BSR 指令 8-5
BTD 指令 6-11
BTDT 指令 6-14
BXOR 6-41

C

CASE C-16
CLR 指令 6-17
CMP 指令 4-2
COMPARE 4-2
COMPARE 结构 12-3, 12-11
CONCAT 指令 17-3
CONTROL 结构 7-8, 7-19, 7-39, 7-43, 7-48, 8-2, 8-5, 8-8, 8-14, 8-20, 8-26, 9-2, 9-6, 9-10
CONTROLLER 对象 3-37
CONTROLLERDEVICE 对象 3-37
COP 指令 7-28
COS 指令 13-5
COUNTER 结构 2-28, 2-32
CPS 指令 7-28
CPT 指令 5-2
CST 对象 3-39
CTD 指令 2-32
CTU 指令 2-28
CTUD 指令 2-36

D

DDT 指令
操作数 12-10
搜索模式 12-12
DEG 指令 15-2
DELETE 指令 17-5
DF1 对象 3-40
DINT 到字符串 18-8
DIV 指令 5-15
DTOS 指令 18-8
DTR 指令 12-18

E

EOT 指令 10-25
EQU 指令 4-7
EVENT 指令 10-31

F

FAL 指令
操作数 7-7
运算模式 7-2
FAULTLOG 对象 3-43
FBC 指令
操作数 12-2
搜索模式 12-4
FBD_BIT_FIELD_DISTRIBUTE 结构 6-14
FBD_BOOLEAN_AND 结构 6-35
FBD_BOOLEAN_NOT 结构 6-44
FBD_BOOLEAN_OR 结构 6-38
FBD_BOOLEAN_XOR 结构 6-41
FBD_COMPARE 结构 4-8, 4-12, 4-16, 4-20, 4-24, 4-39
FBD_CONVERT 结构 15-6, 15-9
FBD_COUNTER 结构 2-36
FBD_LIMIT 结构 4-28
FBD_LOGICAL 结构 6-24, 6-27, 6-30, 6-32
FBD_MASK_EQUAL 结构 4-34
FBD_MASKED_MOVE 结构 6-8
FBD_MATH 结构 5-7, 5-10, 5-13, 5-16, 5-20, 5-26, 14-7
FBD_MATH_ADVANCED 结构 5-23, 5-29, 13-2, 13-5, 13-8, 13-11, 13-14, 13-17, 14-2, 14-4, 15-2, 15-4
FBD_ONESHOT 结构 1-19, 1-22
FBD_TIMER 结构 2-15, 2-19, 2-23
FBD_TRUNCATE 结构 15-11
FFL 指令 8-8
FFU 指令 8-14
FIFO 加载 8-8
FIFO 卸载 8-14
FIND 指令 17-7
FLL 指令 7-34
FOR 指令 11-2
for/break 指令
BRK 11-5
FOR 11-2
RET 11-6
简介 11-1
FOR…DO C-19
FRD 指令 15-9
FSC 指令
操作数 7-19
运算模式 7-2

G

GEQ 指令 4-11
GRT 指令 4-15
GSV 指令
操作数 3-33
对象 3-36

I

ICON B-1
IF...THEN C-13
INSERT 指令 17-9
IOT 指令 3-57
IREF B-1

J

JMP 指令 10-2
JSR 指令 10-4
JXR 指令
控制结构 10-15

L

LBL 指令 10-2
LEQ 指令 4-19
LES 指令 4-23
LFL 指令 8-20
LFU 指令 8-26
LIFO 加载 8-20
LIFO 卸载 8-26
LIM 指令 4-27
LN 指令 14-2
LOG
以 10 为底 14-4
自然 14-2
LOG 指令 14-4
LOWER 指令 18-14

M

MCR 指令 10-19
MEQ 指令 4-33
MESSAGE 对象 3-44
MESSAGE 结构 3-2

MID 指令 17-11
MOD 指令 5-19
MODULE 对象 3-46
MOTIONGROUP 对象 3-47
MOV 指令 6-3
move 6-3
MSG 指令 3-16
  编程原则 3-32
  操作数 3-2
  错误代码 3-9
  缓存连接 3-30
  结构 3-2
  通信方法 3-29
MUL 指令 5-12
MVM 指令 6-5
MVMT 指令 6-8

N

NEG 指令 5-26
NEQ 指令 4-38
NOP 指令 10-24
NOT 指令 6-32

O

OCON B-1
ONS 指令 1-12
OR 指令 6-26
OREF B-1
OSF 指令 1-17
OSFI 指令 1-22
OSR 指令 1-15
OSRI 指令 1-19
OTE 指令 1-6
OTL 指令 1-8
OTU 指令 1-10

P

PID 结构 12-22
PID 指令
  比例 12-29
  操作数 12-21
  调节 12-27
  警报 12-28
  配置 12-26
  前馈 12-39
  输出偏置 12-39
  死区 12-38
PROGRAM 对象 3-48

R

RAD 指令 15-4
REAL 到字符串 18-10
REPEAT…UNTIL C-25
RES 指令 2-40
RESULT 结构 12-3, 12-11
RET 指令 10-4, 11-6
ROUTINE 对象 3-49
RTO 指令 2-10
RTOR 指令 2-23
RTOS 指令 18-10

S

SBR 指令 10-4
SERIAL_PORT_CONTROL 结构
  16-2, 16-4, 16-5, 16-8, 16-13, 16-17, 16-20, 16-24, 16-29
SERIALPORT 对象 3-49
SFP 指令 10-27
SFR 指令 10-29
SIN 指令 13-2
SIZE 指令 7-53
SQI 指令 9-2
SQL 指令 9-10
SQO 指令 9-6
SQR 指令 5-23
SRT 指令 7-43
SSV 指令
  操作数 3-33
  对象 3-36
STD 指令 7-48
STOD 指令 18-4
STOR 指令 18-6
string
  结构化文本中的计算 C-8
STRING 结构 16-3, 17-2, 18-3
SUB 指令 5-9
SWPB 指令 6-19

T

TAN 指令 13-8
TASK 对象 3-51
TIMER 结构 2-2, 2-6, 2-10
TND 指令 10-17
TOD 指令 15-6
TOF 指令 2-6
TOFR 指令 2-19

TON 指令 2-2
TONR 指令 2-15
TRN 指令 15-11

U

UID 指令 10-21
UIE 指令 10-21
UPPER 指令 18-12

W

WALLCLOCKTIME 对象 3-53
WHILE…DO C-22

X

X 的 Y 次方 14-6
XIO 指令 1-4
XOR 指令 6-29
XPY 指令 14-6

按

按位非 6-32
按位或 6-26
按位异或 6-29
按位与 6-23

保

保持计时器打开 2-10

比

比较指令
- CMP 4-2
- EQU 4-7
- GEQ 4-11
- GRT 4-15
- LEQ 4-19
- LES 4-23
- LIM 4-27
- MEQ 4-33
- NEQ 4-38
- 表达式格式 4-5, 7-25
- 简介 4-1
- 有效运算符 4-4, 7-25
- 运算次序 4-5, 7-26

比例 12-29
比例、积分和微分 12-21

编

编写控制指令
- AFI 10-23
- EOT 10-25
- EVENT 10-31
- JMP 10-2
- JSR 10-4
- LBL 10-2
- MCR 10-19
- NOP 10-24
- RET 10-4
- SBR 10-4
- TND 10-17
- UID 10-21
- UIE 10-21
- 简介 10-1

标

标号 10-2
标准偏差 7-48

表

表达式
- BOOL 表达式
  - 结构化文本 C-4
- 格式 4-5, 5-4, 7-17, 7-25
- 结构化文本
  - 概述 C-4
  - 功能 C-6
  - 关系运算符 C-7
  - 逻辑运算符 C-9
  - 算术运算符 C-6
  - 位运算符 C-10
- 数值表达式
  - 结构化文本 C-4
- 有效运算符 4-4, 5-4, 7-17, 7-25
- 运算次序 4-5, 5-5, 7-18, 7-26
- 执行顺序
  - 结构化文本 C-10

不

不等于 4-38

布

布尔非 6-44
布尔或 6-38
布尔异或 6-41
布尔与 6-35

插

插入字符串 17-9

查

查找字符串 17-7

产

产品代码 3-37

常

常用属性 A-1
- 立即数 A-1
- 转换数据类型 A-1

超

超时
- 配置事件任务 3-51

乘

乘法 5-12

程

程序 / 操作员控制
- 概述 B-13

除

除法 5-15

触

触发事件任务 10-31
触发事件任务指令 10-31

串

串行端口指令
- ABL 16-5
- ACB 16-8
- ACL 16-10
- AHL 16-12
- ARD 16-16
- ARL 16-19
- AWA 16-23
- AWT 16-28
- 简介 16-1

错

错误代码
- ASCII 16-4
- MSG 指令 3-9

大

大写 18-12
大于 4-15
大于或等于 4-11

带

带目标的位域分配 6-14
带目标的掩码移动 6-8
带输入的上升沿单触发 1-19
带输入的下降沿单次触发 1-22
带有复位的保持计时器打开 2-23
带有复位的计时器打开延迟 2-15
带有复位的计时器关闭延迟 2-19

单

单触发 1-12

等

等于 4-7

底

底为 10 的对数 14-4

递

递减 2-32
递增 2-28
递增 / 递减 2-36

调

调节 12-27

定

定序程序加载 9-10
定序程序输出 9-6
定序程序输入 9-2
定序程序指令
- SQI 9-2
- SQL 9-10
- SQO 9-6
- 简介 9-1

对

对象
- CONTROLLER 3-37
- CONTROLLERDEVICE 3-37
- CST 3-39
- DF1 3-40
- FAULTLOG 3-43
- GSV/SSV 指令 3-36
- MESSAGE 3-44
- MODULE 3-46
- MOTIONGROUP 3-47
- PROGRAM 3-48
- ROUTINE 3-49
- SERIALPORT 3-49
- WALLCLOCKTIME 3-53
- 任务 3-51

反

反馈循环
- 功能块图 B-5

反余弦 13-14
反正切 13-17
反正弦 13-11

返

返回 10-4, 11-6

赋

赋值
- ASCII 字符 C-4
- 保持 C-2
- 非保持 C-3

复

复位 2-40
复位 SFC 指令 10-29
复制 7-28

高

高级数学指令
- LN 14-2
- LOG 14-4
- XPY 14-6
- 简介 14-1

更

更新输出 3-57

功

功能
- 结构化文本 C-6

功能块图
- 创建扫描延迟 B-7
- 解析两个块之间的数据流 B-6
- 解析循环 B-5
- 选择元素 B-1

关

关系运算符
- 结构化文本 C-7

恒

恒假指令 10-23

后

后扫描
- 结构化文本 C-3

弧

弧度 15-4

缓

缓冲区行的 ASCII 测试 16-5
缓冲区中的 ASCII 字符 16-8
缓存
- 连接 3-30

混

混用数据类型 A-1

获

获取系统值 3-33

计

计时模式 B-9
计时器打开延迟 2-2
计时器关闭延迟 2-6
计时器指令
- RES 2-40
- RTO 2-10
- RTOR 2-23
- TOF 2-6
- TOFR 2-19

TON 2-2
TONR 2-15
简介 2-1
计数器指令
CTD 2-32
CTU 2-28
CTUD 2-36
RES 2-40
简介 2-1
计算 5-2
计算指令
ABS 5-29
ADD 5-6
CPT 5-2
DIV 5-15
MOD 5-19
MUL 5-12
NEG 5-26
SQR 5-23
SUB 5-9
表达式格式 5-4, 7-17
简介 5-1
有效运算符 5-4, 7-17
运算次序 5-5, 7-18

加

加法 5-6

假

假定数据可用 B-5, B-6, B-7

检

检查是否已打开 1-4

减

减法 5-9

交

交换字节 6-19

角

角度 15-2

截

截断 15-11

结

结构
COMPARE 12-3, 12-11
CONTROL 7-8, 7-19, 7-39, 7-43, 7-48, 8-2, 8-5, 8-8, 8-14, 8-20, 8-26, 9-2, 9-6, 9-10
COUNTER 2-28, 2-32
FBD_BIT_FIELD_DISTRIBUTE 6-14
FBD_BOOLEAN_AND 6-35
FBD_BOOLEAN_NOT 6-44
FBD_BOOLEAN_OR 6-38
FBD_BOOLEAN_XOR 6-41
FBD_COMPARE 4-8, 4-12, 4-16, 4-20, 4-24, 4-39
FBD_CONVERT 15-6, 15-9
FBD_COUNTER 2-36
FBD_LIMIT 4-28
FBD_LOGICAL 6-24, 6-27, 6-30, 6-32
FBD_MASK_EQUAL 4-34
FBD_MASKED_MOVE 6-8
FBD_MATH 5-7, 5-10, 5-13, 5-16, 5-20, 5-26, 14-7
FBD_MATH_ADVANCED 5-23, 5-29, 13-2, 13-5, 13-8, 13-11, 13-14, 13-17, 14-2, 14-4, 15-2, 15-4
FBD_ONESHOT 1-19, 1-22
FBD_TIMER 2-15, 2-19, 2-23
FBD_TRUNCATE 15-11
MESSAGE 3-2
PID 12-22
RES 指令 2-40
Result 12-3, 12-11
SERIAL_PORT_CONTROL 16-2, 16-4, 16-5, 16-8, 16-13, 16-17, 16-20, 16-24, 16-29
STRING 16-3, 17-2, 18-3
string 16-3, 18-3
TIMER 2-2, 2-6, 2-10
结构化文本 C-12
字符串 17-2
结构化文本
CASE C-16
FOR…DO C-19
IF...THEN C-13
REPEAT…UNTIL C-25
WHILE…DO C-22
表达式 C-4
非保持赋值 C-3

赋值 C-2
功能 C-6
关系运算符 C-7
结构 C-12
逻辑运算符 C-9
数值表达式 C-4
算术运算符 C-6
位运算符 C-10
用 ASCII 字符赋值 C-4
注释 C-28
字符串的计算 C-8
组成部分 C-1

禁

禁止
任务 3-51
禁止用户中断 10-21

警

警报 12-28

绝

绝对值 5-29

空

空操作 10-24

控

控制结构 10-15

立

立即输出指令 3-57
立即数 A-1

连

连接
缓存 3-30
连接器
功能块图 B-1

连

临时结束 10-17

逻

逻辑运算符
结构化文本 C-9
逻辑指令
AND 6-23
NOT 6-32
OR 6-26
XOR 6-29
简介 6-1

排

排序 7-43

配

配置 3-16
MSG 指令 3-16
PID 指令 12-26

平

平方根 5-23
平均值 7-38

前

前馈 12-39

清

清零 6-17

求

求反 5-26

取

取余 5-19

任

任务
触发事件任务 10-31
监视 3-51
禁止 3-51
通过使用的标记触发 3-57
以编程方式配置 3-51

三

三角函数指令
- ACS 13-14
- ASN 13-11
- ATN 13-17
- COS 13-5
- SIN 13-2
- TAN 13-8
- 简介 13-1

扫

扫描延迟
- 功能块图 B-7

删

删除字符串 17-5

上

上升单沿触发 1-15
上升沿单次触发 1-15

设

设置系统值 3-33

事

事件任务
- 配置 3-51
- 通过 EVENT 指令触发 10-31
- 通过使用的标记触发 3-57

输

输出
- 立即更新 3-57
- 启用或禁用任务结束处理 3-51

输出参考 B-1
输出激励 1-6
输出解锁 1-10
输出偏置 12-39
输出锁存 1-8
输出线路连接器 B-1
输入 / 输出指令
- GSV 3-33
- IOT 3-57
- MSG 3-2
- SSV 3-33
- 简介 3-1

输入参考 B-1
输入线路连接器 B-1

属

属性
- 立即数 A-1
- 转换数据类型 A-1

数

数据转换 12-18
数学运算符
- 结构化文本 C-6

数学转换指令
- DEG 15-2
- FRD 15-9
- RAD 15-4
- TOD 15-6
- TRN 15-11
- 简介 15-1

数值表达式 C-4
数值模式 7-3
数组指令
- AVE 7-38
- BSL 8-2
- BSR 8-5
- COP 7-28
- CPS 7-28
- DDT 12-10
- FAL 7-7
- FBC 12-2
- FFL 8-8
- FFU 8-14
- FLL 7-34
- FSC 7-19
- LFL 8-20
- LFU 8-26
- RES 2-40
- SIZE 7-53
- SQI 9-2
- SQL 9-10
- SQO 9-6
- SRT 7-43
- STD 7-48

定序程序 9-1
文件 / 综合 7-1
移位 8-1

说

说明
结构化文本 C-28

死

死区 12-38

搜

搜索模式 12-4, 12-12
搜索字符串 17-7

算

算术运算符
结构化文本 C-6
算术状态标志
溢出 B-8

锁

锁存数据 B-2

特

特殊指令
DDT 12-10
DTR 12-18
FBC 12-2
PID 12-21
SFP 10-27
SFR 10-29
简介 12-1

提

提取字符串 17-11

跳

跳转 10-2
跳转到子例程 10-4

同

同步复制 7-28

未

未解析的循环
功能块图 B-5

位

位右移 8-5
位域分配 6-11
位运算符
结构化文本 C-10
位指令
ONS 1-12
OSF 1-17
OSFI 1-22
OSR 1-15
OSRI 1-19
OTE 1-6
OTL 1-8
OTU 1-10
XIO 1-4
简介 1-1
位左移 8-2

文

文档
结构化文本 C-28
文件搜索和比较 7-19
文件算术和逻辑 7-7
文件填充 7-34
文件位比较 12-2
文件指令。请参见数组指令

下

下降沿单次触发 1-17

限

限制 4-27

消

消息 3-2
编程原则 3-32
缓存连接 3-30

小

小写 18-14
小于 4-23
小于或等于 4-19

掩

掩码 12-19
掩码等于 4-33
掩码移动 6-5

移

移动 / 逻辑指令
- BAND 6-35
- BNOT 6-44
- BOR 6-38
- BXOR 6-41

移动指令
- BTD 6-11
- BTDT 6-14
- CLR 6-17
- MOV 6-3
- MVM 6-5
- MVMT 6-8
- 简介 6-1

移位令
- 简介 8-1

移位指令
- BSL 8-2
- BSR 8-5
- FFL 8-8
- FFU 8-14
- LFL 8-20
- LFU 8-26

溢

溢出情况 B-8

余

余弦 13-5

元

元素
- SIZE 指令 7-53

元素大小 7-53

允

允许用户中断 10-21

运

运算次序 4-5, 5-5, 7-18, 7-26
运算符 4-4, 5-4, 7-17, 7-25
- 执行顺序
  - 结构化文本 C-10

运算模式 7-2

暂

暂停 SFC 指令 10-27

增

增量模式 7-5

诊

诊断检测 12-10

整

整体模式 7-2

正

正切 13-8
正弦 13-2

执

执行顺序 B-4
- 结构化文本表达式 C-10

指

指令
- ASCII 串行端口 16-1
- ASCII 转换 18-1
- ASCII 字符串操作 17-1
- COMPARE 4-1
- for/break 11-1
- 程序控制 10-1
- 串行端口 16-1
- 定序程序 9-1
- 高级数学 14-1
- 计时器 2-1
- 计数器 2-1
- 计算 5-1
- 逻辑 6-1
- 三角 13-1
- 输入 / 输出 3-1
- 数学转换 15-1
- 数组
- 特殊 12-1
- 位 1-1
- 移动 6-1
- 移位 8-1
- 转换 15-1
- 字符串操作 17-1
- 字符串转换 18-1

指数 14-6

中

中断 11-5

重

重叠
检查任务重叠 3-51

主

主控复位 10-19

注

注释
结构化文本 C-28

转

转换结束指令 10-25
转换数据类型 A-1
转换为 BCD 15-6
转换为整数 15-9
转换指令
DEG 15-2
FRD 15-9
RAD 15-4
TOD 15-6
TRN 15-11
简介 15-1

状

状态
任务 3-51

子

子例程 10-4
自然对数 14-2
字符串操作指令
CONCAT 17-3
DELETE 17-5
FIND 17-7
INSERT 17-9
MID 17-11
简介 17-1

字

字符串串联 17-3
字符串到 DINT 18-4
字符串到 REAL 18-6
字符串数据类型 16-3, 17-2, 18-3
字符串转换指令
DTOS 18-8
LOWER 18-14
RTOS 18-10
STOD 18-4
STOR 18-6
SWPB 6-19
UPPER 18-12
简介 18-1

How Are We Doing?

Your comments on our technical publications will help us serve you better in the future. Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at www.ab.com/manuals, or email us at RADocumentComments@ra.rockwell.com

Pub. Title/Type Logix5000™ Controllers General Instructions

Cat. No. _____ Pub. No. 1756-RM003H-ZH-P Pub. Date 5 '¬ 2005ƒÍ Part No. 957974-78

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness	1	2	3	How can we make this publication more useful for you?
Completeness (all necessary information is provided)	1	2	3	Can we add more information to help you? procedure/step illustration feature example guideline other explanation definition
Technical Accuracy (all provided information is correct)	1	2	3	Can we be more accurate? text illustration
Clarity (all provided information is easy to understand)	1	2	3	How can we make things clearer?
Other Comments				You can add additional comments on the back of this form.

Your Name _____ Location/Phone _____

Your Title/Function _____

Would you like us to contact you regarding your comments?

___No, there is no need to contact me

___Yes, please call me

___Yes, please email me at _____

___Yes, please contact me via _____

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Phone: 440-646-3176 Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

出版号 1756-RM003H-ZH-P - 2005 年 5 月 PN 957974-78

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Allen-Bradley
RELIANCE ELECTRIC
DODGE
ROCKWELL SOFTWARE
Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

ASCII 字符代码

字符	十进制	十六进制
[ctrl-@] NUL	0	$00
[ctrl-A] SOH	1	$01
[ctrl-B] STX	2	$02
[ctrl-C] ETX	3	$03
[ctrl-D] EOT	4	$04
[ctrl-E] ENQ	5	$05
[ctrl-F] ACK	6	$06
[ctrl-G] BEL	7	$07
[ctrl-H] BS	8	$08
[ctrl-I] HT	9	$09
[ctrl-J] LF	10	$I ($0A)
[ctrl-K] VT	11	$0B
[ctrl-L] FF	12	$0C
[ctrl-M] CR	13	$r ($0D)
[ctrl-N] SO	14	$0E
[ctrl-O] SI	15	$0F
[ctrl-P] DLE	16	$10
[ctrl-Q] DC1	17	$11
[ctrl-R] DC2	18	$12
[ctrl-S] DC3	19	$13
[ctrl-T] DC4	20	$14
[ctrl-U] NAK	21	$15
[ctrl-V] SYN	22	$16
[ctrl-W] ETB	23	$17
[ctrl-X] CAN	24	$18
[ctrl-Y] EM	25	$19
[ctrl-Z] SUB	26	$1A
ctrl-[ESC	27	$1B
[ctrl-\] FS	28	$1C
ctrl-] GS	29	$1D
[ctrl-^] RS	30	$1E
[ctrl-_] US	31	$1F
SPACE	32	$20
!	33	$21
“	34	$22
#	35	$23
$	36	$24
%	37	$25
&	38	$26
‘	39	$27
(	40	$28
)	41	$29
*	42	$2A
+	43	$2B
,	44	$2C
-	45	$2D
.	46	$2E
/	47	$2F
0	48	$30
1	49	$31
2	50	$32
3	51	$33
4	52	$34
5	53	$35
6	54	$36
7	55	$37
8	56	$38
9	57	$39
:	58	$3A
;	59	$3B
<	60	$3C
=	61	$3D
>	62	$3E
?	63	$3F
@	64	$40
A	65	$41
B	66	$42
C	67	$43
D	68	$44
E	69	$45
F	70	$46
G	71	$47
H	72	$48
I	73	$49
J	74	$4A
K	75	$4B
L	76	$4C
M	77	$4D
N	78	$4E
O	79	$4F
P	80	$50
Q	81	$51
R	82	$52
S	83	$53
T	84	$54
U	85	$55
V	86	$56
W	87	$57
X	88	$58
Y	89	$59
Z	90	$5A
[	91	$5B
\	92	$5C
]	93	$5D
^	94	$5E
_	95	$5F
‘	96	$60
a	97	$61
b	98	$62
c	99	$63
d	100	$64
e	101	$65
f	102	$66
g	103	$67
h	104	$68
i	105	$69
j	106	$6A
k	107	$6B
l	108	$6C
m	109	$6D
n	110	$6E
o	111	$6F
p	112	$70
q	113	$71
r	114	$72
s	115	$73
t	116	$74
u	117	$75
v	118	$76
w	119	$77
x	120	$78
y	121	$79
z	122	$7A
{	123	$7B
|	124	$7C
}	125	$7D
~	126	$7E
DEL	127	$7F

Rockwell Automation 支持

Rockwell Automation 的网上技术信息可以为产品的使用提供帮助。在 http://support.rockwellautomation.com 网页上，可以找到技术手册、常见问题解答知识库、技术和应用说明、示例代码、指向软件服务包的链接，以及 MySupport 功能，通过定制 MySupport 功能可以更为有效地使用这些工具。

对于安装、配置和故障排除方面的其他级别的电话技术支持服务，我们推出了 TechConnect 支持计划。有关详细信息，请与当地经销商或 Rockwell Automation 代表联系，或者访问 http://support.rockwellautomation.com。

安装助手

如果在安装后的头 24 小时内遇到硬件模块问题，请参阅本手册中包含的信息。您也可以致电某个客户支持号码，以便在使模块启动并正常运行时获得最初的帮助：

美国	1.440.646.3223 东部时间：周一 – 周五，上午 8 点 – 下午 5 点
美国境外	如有任何技术支持方面的问题，请与当地的 Rockwell Automation 代表联系。

新产品的满意退货处理

Rockwell 对所有产品都进行了测试，旨在确保这些产品在运离制造厂后可以完全正常工作。但是，如果因产品工作异常而需要退货，请通过以下方式进行处理：

美国	与经销商联系。您必须为经销商提供客户支持案例代码（请参见上面的电话号码获得一个代码），以便完成退货过程。
美国境外	有关退货流程，请与当地 Rockwell Automation 代表联系。

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe/Middle East/Africa: Rockwell Automation, Herman-Heinrich-Gossen-Strasse 3, 50858 Köln, Germany, Tel: 49 (0) 2234 379410, Fax: 49 (0) 2234 3794164
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 6356 9077, Fax: (65) 6356 9011

出版号 1756-RM003H-ZH-P - 5 月 2005
替代出版号 1756-RM003G-ZH-P - 2003 年 6 月
PN 957974-78
版权所有 © 2005 Rockwell Automation, Inc. 保留所有权利。美国印刷

Allen-Bradley

Logix5000™ 控制器基本指令参考手册

参考手册