

# SIEMENS

## SIMATIC

### 用于 S7-300 和 S7-400 编程 的梯形图(LAD)

#### 参考手册

前言，  
内容

---

位逻辑指令

---

1

比较指令

---

2

转换指令

---

3

计数器指令

---

4

数据块指令

---

5

逻辑控制指令

---

6

整型数学运算指令

---

7

浮点型数学运算指令

---

8

传送指令

---

9

程序控制指令

---

10

移位和循环指令

---

11

状态位指令

---

12

定时器指令

---

13

字逻辑指令

---

14

附录

---

所有 LAD 指令总览

---

A

编程实例

---

B

使用梯形图

---

C

索引

2007 年 8 月版  
A5E01112993 -01

## 安全指南

本手册包括了保证人身安全及防止财产损失所应遵守的注意事项。在手册中与人身安全有关的注意事项用一个安全警告符号高亮显示，而与财产损失有关的注意事项则没有安全警告符号。这些注意事项根据危险等级标明如下。



### 危险

表示如果不采取适当的预防措施，将导致死亡或严重的人身伤害。



### 警告

表示如果不采取适当的预防措施，可能导致死亡或严重的人身伤害。



### 当心

带安全警告符号：表示如果不采取适当的预防措施，可能导致轻微的人身伤害。

### 当心

不带安全警告符号：表示如果不采取适当的预防措施，可能导致财产损失。

### 须知

表示如果忽略相关注意事项，可能会导致非预期的结果或状态。

如果出现一个以上的危险等级，则将使用表示最高危险等级的警告注意事项。用安全警告符号警告造成人身伤害的注意事项可能还包括与财产损失有关的警告。

## 合格人员

必须按照该文档安装和使用该设备/系统。只有合格人员才允许调试和操作该设备。在本文档的安全注意事项中，合格人员是指被授权按照既定安全惯例和标准，对线路、设备和系统进行调试、接地和标记的人员。

## 规定用法

请注意如下事项：



### 警告

本设备及其部件只能用于产品目录或技术说明书中所描述的范畴，并且只能与由西门子公司认可或推荐的第三方厂商提供的设备或部件一起使用。

只有正确地运输、保管、设置和安装本产品，并且按照推荐的方式操作和维护，产品才能正常、安全地运行。

## 商标

由 © 标识的所有名称是西门子 AG 的注册商标。

本文档中的其它一些标志也是注册商标，如果任何第三方出于个人目的而使用，都会侵犯商标所有者的权利。

## 免责声明

我们已检查过本手册中的内容与所描述的硬件和软件相符。由于差错在所难免，我们不能保证完全一致。我们会定期审查本手册中的内容，并在后续版本中进行必要的更正。

# 前言

## 用途

本手册是以梯形图(LAD)程序语言创建用户程序的指南。

本手册同时也包含了描述梯形图语言元素的语法和函数的参考章节。

## 基础知识要求

本手册供 S7 程序员、操作员以及维护/维修人员使用。

要了解本手册，需要具有自动化技术的常规知识。

此外，要求具备计算机应用能力并了解在操作系统 MS Windows 2000 Professional、MS Windows XP Professional 或 MS Windows Server 2003 下使用 PC 类工作设备(如编程设备)的知识。

## 手册应用范围

本手册适用于 STEP 7 编程软件包 5.4 版本。

## 符合 IEC 1131-3 标准

LAD 符合国际电工技术委员会标准 IEC 1131-3 中定义的“梯形图”语言。欲知更多资料，请参见 STEP 7 文件 NORM\_TBL.WRI 中的标准表。

## 要求

要有效使用本梯形图手册，应已熟悉 S7 程序理论，该理论归档在 STEP 7 在线帮助中。语言包也使用 STEP 7 标准软件，因此应熟悉如何使用该软件并已阅读了随附文档。

本手册是文档包“STEP 7 参考书目”中的一部分。

下表显示了 STEP 7 文档的总览：

文档	用途	订货号
STEP 7 基础信息，包括 <ul style="list-style-type: none"> <li>STEP 7，使用入门手册</li> <li>使用 STEP 7 编程</li> <li>组态硬件和通讯连接，STEP 7 版本</li> <li>从 S5 到 S7，变频器手册</li> </ul>	供技术人员使用的基础信息，描述了使用 STEP 7 和 S7-300/400 可编程控制器来实现控制任务的方法。	6ES7810-4CA08-8BW0
STEP 7 参考书目 <ul style="list-style-type: none"> <li>用于 S7-300/400 的梯形图(LAD)/功能块图(FDB)/语句表(STL)手册</li> <li>用于 S7-300/400 的标准功能和系统功能卷 1 和卷 2</li> </ul>	提供参考信息，并描述编程语言 LAD、FBD、STL、标准功能及系统功能，扩充了 STEP 7 基础信息的范围。	6ES7810-4CA08-8BW1

在线帮助	用途	订货号
STEP 7 帮助	以在线帮助的形式，提供了使用 STEP 7 进行编程和组态硬件的基础信息。	STEP 7 标准软件文档的组成部分。
AWL/KOP/FUP 帮助参考 SFB/SFC 帮助参考 组织块帮助参考	上下文相关参考信息。	STEP 7 标准软件文档的组成部分。

## 在线帮助

集成于软件中的在线帮助是对本手册的补充。提供在线帮助的目的是，在使用该软件时提供详细的支持。

该帮助系统通过一些界面集成于软件中：

- 上下文相关帮助提供关于当前语境的信息，例如，打开的对话框或激活的窗口。可以通过菜单命令**帮助 > 上下文相关的帮助**，或 **F1** 键，或工具栏上的问号符来打开上下文相关的帮助。
- 可以通过使用菜单命令**帮助 > 目录**，或在上下文相关的帮助窗口中按“**STEP 7 帮助**”按钮来调用 **STEP 7** 常规帮助。
- 可以通过按下“词汇表”按钮，调用所有 **STEP7** 应用程序的词汇表。

本手册是“梯形图帮助”的摘要。由于手册和在线帮助具有完全相同的结构，因此非常容易在手册和在线帮助之间切换。

## 更多支持

如果有任何技术问题，请联系西门子代表或代理商。

您可以在下列网页中查找联系人：

<http://www.siemens.com/automation/partner>

可以在下列网址上找到各个 **SIAMTIC** 产品和系统的技术文档指南：

<http://www.siemens.com/simatic-tech-doku-portal>

可以在下列网址上获得在线目录和订货系统：

<http://mall.automation.siemens.com/>

## 培训中心

西门子提供了很多培训教程，帮助您熟悉 **SIMATIC S7** 自动化系统。请联系当地的培训中心，或位于德国纽伦堡(D 90327)的培训总部，以获取详细信息。

电话：+49 (911) 895-3200。

网址：<http://www.sitrain.com>

## 技术支持

您可以获取所有 A&D 产品的技术支持

- 通过网站请求支持  
<http://www.siemens.com/automation/support-request>
- 电话: + 49 180 5050 222
- 传真: + 49 180 5050 223

关于技术支持的更多信息请参见 Internet 网页:

<http://www.siemens.com/automation/service>

## Internet 服务和支持

除文档以外, 还在 Internet 上在线提供了知识产权信息, 网址如下:

<http://www.siemens.com/automation/service&support>

可在其中查找下列内容:

- 公司简讯, 经常提供产品的最新信息。
- 相应文档资料, 可通过“服务和支持”中的搜索功能查找。
- 论坛, 世界各地的用户和专家可以在此交流经验。
- 当地自动化和驱动办事处。
- 在“服务”页面下提供了关于现场服务、维修、备件等信息。

# 内容

<b>1</b>	<b>位逻辑指令</b> .....	<b>1-1</b>
1.1	位逻辑指令概述 .....	1-1
1.2	--   -- 常开触点(地址) .....	1-2
1.3	--  /  -- 常闭触点(地址) .....	1-3
1.4	XOR 位异或运算 .....	1-4
1.5	-- NOT -- 能流取反 .....	1-5
1.6	--( ) 输出线圈 .....	1-6
1.7	--( # )-- 中线输出 .....	1-8
1.8	--( R ) 复位线圈 .....	1-10
1.9	--( S ) 置位线圈 .....	1-12
1.10	RS 置位优先型 RS 双稳态触发器 .....	1-14
1.11	SR 复位优先型 SR 双稳态触发器 .....	1-16
1.12	--( N )-- RLO 负跳沿检测 .....	1-18
1.13	--( P )-- RLO 正跳沿检测 .....	1-19
1.14	--(SAVE) 将 RLO 保存到 BR 存储器中 .....	1-20
1.15	NEG 地址下降沿检测 .....	1-21
1.16	POS 地址上升沿检测 .....	1-22
1.17	立即读取 .....	1-23
1.18	立即写入 .....	1-24
<b>2</b>	<b>比较指令</b> .....	<b>2-1</b>
2.1	比较指令概述 .....	2-1
2.2	CMP ? I 整数比较 .....	2-2
2.3	CMP ? D 长整数比较 .....	2-4
2.4	CMP ? R 实数比较 .....	2-6
<b>3</b>	<b>转换指令</b> .....	<b>3-1</b>
3.1	转换指令概述 .....	3-1
3.2	BCD_I BCD 码转换为整型 .....	3-2
3.3	I_BCD 整型转换为 BCD 码 .....	3-3
3.4	I_DINT 整型转换为长整型 .....	3-4
3.5	BCD_DI BCD 码转换为长整型 .....	3-5
3.6	DI_BCD 长整型转换为 BCD 码 .....	3-6
3.7	DI_REAL 长整型转换为浮点型 .....	3-7
3.8	INV_I 对整数求反码 .....	3-8
3.9	INV_DI 对长整数求反码 .....	3-9
3.10	NEG_I 对整数求补码 .....	3-10
3.11	NEG_DI 对长整数求补码 .....	3-11
3.12	NEG_R 浮点数取反 .....	3-12
3.13	ROUND 取整为长整型 .....	3-13
3.14	TRUNC 截取长整数部分 .....	3-14
3.15	CEIL 向上取整 .....	3-15
3.16	FLOOR 向下取整 .....	3-16

<b>4</b>	<b>计数器指令</b> .....	<b>4-1</b>
4.1	计数器指令概述 .....	4-1
4.2	S_CUD 双向计数器 .....	4-3
4.3	S_CU 升值计数器 .....	4-5
4.4	S_CD 降值计数器 .....	4-7
4.5	--( SC ) 设置计数器值 .....	4-9
4.6	--( CU ) 升值计数器线圈 .....	4-10
4.7	--( CD ) 降值计数器线圈 .....	4-12
<b>5</b>	<b>数据块指令</b> .....	<b>5-1</b>
5.1	---(OPN)打开数据块: DB 或 DI .....	5-1
<b>6</b>	<b>逻辑控制指令</b> .....	<b>6-1</b>
6.1	逻辑控制指令概述 .....	6-1
6.2	---(JMP)--- 无条件跳转 .....	6-2
6.3	---(JMP)--- 条件跳转 .....	6-3
6.4	---(JMPN)若“否”则跳转 .....	6-4
6.5	LABEL 标号 .....	6-5
<b>7</b>	<b>整型数学运算指令</b> .....	<b>7-1</b>
7.1	整数算术指令概述 .....	7-1
7.2	使用整数算术指令时得出状态字的位数值 .....	7-2
7.3	ADD_I 整数加 .....	7-3
7.4	SUB_I 整数减 .....	7-4
7.5	MUL_I 整数乘 .....	7-5
7.6	DIV_I 整数除 .....	7-6
7.7	ADD_DI 长整数加 .....	7-7
7.8	SUB_DI 长整数减 .....	7-8
7.9	MUL_DI 长整数乘 .....	7-9
7.10	DIV_DI 长整数除 .....	7-10
7.11	MOD_DI 返回长整数余数 .....	7-11
<b>8</b>	<b>浮点型数学运算指令</b> .....	<b>8-1</b>
8.1	浮点运算指令概述 .....	8-1
8.2	使用浮点运算指令时得出状态字的位数值 .....	8-2
8.3	基本指令 .....	8-3
8.3.1	ADD_R 实数加 .....	8-3
8.3.2	SUB_R 实数减 .....	8-5
8.3.3	MUL_R 实数乘 .....	8-6
8.3.4	DIV_R 实数除 .....	8-7
8.3.5	ABS 求浮点数的绝对值 .....	8-8
8.4	扩展指令 .....	8-9
8.4.1	SQR 求平方 .....	8-9
8.4.2	SQRT 求平方根 .....	8-10
8.4.3	EXP 求指数值 .....	8-11
8.4.4	LN 求自然对数 .....	8-12
8.4.5	SIN 求正弦值 .....	8-13
8.4.6	COS 求余弦值 .....	8-14
8.4.7	TAN 求正切值 .....	8-15
8.4.8	ASIN 求反正弦值 .....	8-16



8.4.9	ACOS 求反余弦值.....	8-17
8.4.10	ATAN 求反正切值.....	8-18
<b>9</b>	<b>传送指令.....</b>	<b>9-1</b>
9.1	MOVE 分配值.....	9-1
<b>10</b>	<b>程序控制指令.....</b>	<b>10-1</b>
10.1	程序控制指令概述.....	10-1
10.2	--(Call) 调用来自线圈的 FC SFC (不带参数).....	10-2
10.3	CALL_FB 调用来自框的 FB.....	10-4
10.4	CALL_FC 调用来自框的 FC.....	10-6
10.5	CALL_SFB 调用来自框的系统 FB.....	10-8
10.6	CALL_SFC 调用来自框的系统 FC.....	10-10
10.7	调用多重实例.....	10-12
10.8	从库中调用块.....	10-13
10.9	使用 MCR 功能的重要注意事项.....	10-13
10.10	---(MCR<) 主控制继电器打开.....	10-14
10.11	---(MCR>) 主控制继电器关闭.....	10-16
10.12	---(MCRA) 主控制继电器激活.....	10-18
10.13	--(MCRD) 主控制继电器取消激活.....	10-19
10.14	--(RET) 返回.....	10-20
<b>11</b>	<b>移位和循环指令.....</b>	<b>11-1</b>
11.1	移位指令.....	11-1
11.1.1	移位指令概述.....	11-1
11.1.2	SHR_I 整数右移.....	11-2
11.1.3	SHR_DI 长整数右移.....	11-4
11.1.4	SHL_W 字左移.....	11-5
11.1.5	SHR_W 字右移.....	11-7
11.1.6	SHL_DW 双字左移.....	11-8
11.1.7	SHR_DW 双字右移.....	11-9
11.2	循环移位指令.....	11-11
11.2.1	循环移位指令概述.....	11-11
11.2.2	ROL_DW 双字循环左移.....	11-11
11.2.3	ROR_DW 双字循环右移.....	11-13
<b>12</b>	<b>状态位指令.....</b>	<b>12-1</b>
12.1	状态位指令概述.....	12-1
12.2	OV ---   --- 异常位溢出.....	12-2
12.3	OS ---   --- 存储的异常位溢出.....	12-3
12.4	UO ---   --- 无序异常位.....	12-5
12.5	BR ---   --- 异常位二进制结果.....	12-6
12.6	==0 ---   --- 结果位等于 0.....	12-7
12.7	<>0 ---   --- 结果位不等于 0.....	12-8
12.8	>0 ---   --- 结果位大于 0.....	12-9
12.9	<0 ---   --- 结果位小于 0.....	12-10
12.10	>=0 ---   --- 结果位大于等于 0.....	12-11
12.11	<=0 ---   --- 结果位小于等于 0.....	12-12
<b>13</b>	<b>定时器指令.....</b>	<b>13-1</b>
13.1	定时器指令概述.....	13-1

13.2	定时器在存储器中的位置与定时器组件 .....	13-2
13.3	S_PULSE 脉冲 S5 定时器 .....	13-5
13.4	S_PEXT 扩展脉冲 S5 定时器 .....	13-7
13.5	S_ODT 接通延时 S5 定时器 .....	13-9
13.6	S_ODTS 保持接通延时 S5 定时器 .....	13-11
13.7	S_OFFDT 断开延时 S5 定时器 .....	13-13
13.8	--( SP ) 脉冲定时器线圈 .....	13-15
13.9	--( SE ) 扩展脉冲定时器线圈 .....	13-17
13.10	--( SD ) 接通延时定时器线圈 .....	13-19
13.11	--( SS ) 保持性接通延时定时器线圈 .....	13-21
13.12	--( SF ) 断开延时定时器线圈 .....	13-23
<b>14</b>	<b>字逻辑指令 .....</b>	<b>14-1</b>
14.1	字逻辑指令概述 .....	14-1
14.2	WAND_W (字)单字与运算 .....	14-2
14.3	WOR_W (字)单字或运算 .....	14-3
14.4	WAND_DW (字)双字与运算 .....	14-4
14.5	WOR_DW (字)双字或运算 .....	14-5
14.6	WXOR_W (字)单字异或运算 .....	14-6
14.7	WXOR_DW (字)双字异或运算 .....	14-7
<b>A</b>	<b>所有 LAD 指令总览 .....</b>	<b>A-1</b>
A.1	按英语助记符(国际)排序的 LAD 指令 .....	A-1
A.2	按德语助记符(SIMATIC)排序的 LAD 指令 .....	A-4
<b>B</b>	<b>编程实例 .....</b>	<b>B-1</b>
B.1	编程实例概述 .....	B-1
B.2	实例：位逻辑指令 .....	B-2
B.3	实例：定时器指令 .....	B-6
B.4	实例：计数器和比较指令 .....	B-10
B.5	实例：整型数学运算指令 .....	B-13
B.6	实例：字逻辑指令 .....	B-14
<b>C</b>	<b>使用梯形图 .....</b>	<b>C-1</b>
C.1	EN/ENO 机制 .....	C-1
C.1.1	加法器连接了 EN 和 ENO .....	C-3
C.1.2	加法器连接了 EN 但未连接 ENO .....	C-4
C.1.3	加法器未连接 EN 但连接了 ENO .....	C-5
C.1.4	加法器未连接 EN 和 ENO .....	C-6
C.2	参数传送 .....	C-7
<b>索引</b>	<b>.....索引-1</b>	

# 1 位逻辑指令

## 1.1 位逻辑指令概述

### 描述

位逻辑指令使用两个数字 1 和 0。这两个数字构成二进制系统的基础。这两个数字 1 和 0 称为二进制数字或位。对于触点和线圈而言，1 表示已激活或已励磁，0 表示未激活或未励磁。

位逻辑指令解释信号状态 1 和 0，并根据布尔逻辑将其组合。这些组合产生称为“逻辑运算结果” (RLO)的结果 1 或 0。

由位逻辑指令触发的逻辑运算可以执行各种功能。

有可以执行下列功能的位逻辑指令：

- --| |-- 常开触点(地址)
- --| / |-- 常闭触点(地址)
- --(SAVE) 将 RLO 状态保存到 BR
- XOR 位异或运算
- --- ( ) 输出线圈
- --- (#)--- 中间输出
- ---|NOT|--- 取反使能位

RLO 为 1 时将触发下列指令：

- ---( S ) 置位线圈
- ---( R ) 复位线圈
- SR 复位优先型 SR 双稳态触发器
- RS 置位优先型 RS 双稳态触发器

其它指令将对上升沿或下降沿过渡做出反应，执行下列功能：

- ---(N)--- RLO 负跳沿检测
- ---(P)--- RLO 正跳沿检测
- NEG 地址下降沿检测
- POS 地址上升沿检测
- 立即读取
- 立即写入

## 1.2 ---| |--- 常开触点(地址)

### 符号

<地址>

---| |---

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D、 T、C	选中的位

### 描述

---| |--- 存储在指定<地址>的位值为“1”时，(常开触点)处于闭合状态。触点闭合时，梯形图轨道能流流过触点，逻辑运算结果(RLO) = “1”。

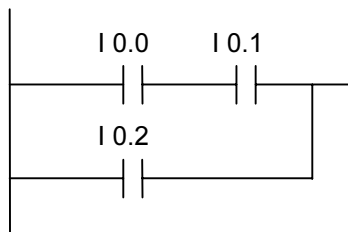
否则，如果指定<地址>的信号状态为“0”，触点将处于断开状态。触点断开时，能流不流过触点，逻辑运算结果(RLO) = “0”。

串联使用时，通过 AND 逻辑将---| |--- 与 RLO 位进行链接。并联使用时，通过 OR 逻辑将其与 RLO 位进行链接。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	X	X	X	1

### 实例



满足下列条件之一时，将会通过能流：

输入端 I0.0 和 I0.1 的信号状态为“1”时

或输入端 I0.2 的信号状态为“1”时

### 1.3 ---| / |--- 常闭触点(地址)

#### 符号

<地址>

---| / |---

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D、 T、C	选中的位

#### 描述

---| / |--- 存储在指定<地址>的位值为“0”时，(常闭触点)处于闭合状态。触点闭合时，梯形图轨道能流流过触点，逻辑运算结果(RLO) = “1”。

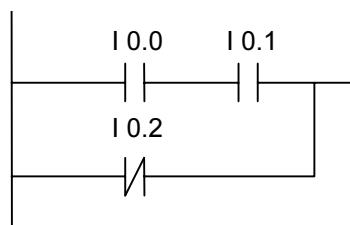
否则，如果指定<地址>的信号状态为“1”，将断开触点。触点断开时，能流不流过触点，逻辑运算结果(RLO) = “0”。

串联使用时，通过 AND 逻辑将 ---| / |--- 与 RLO 位进行链接。并联使用时，通过 OR 逻辑将其与 RLO 位进行链接。

#### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	X	X	X	1

#### 实例



满足下列条件之一时，将会通过能流：

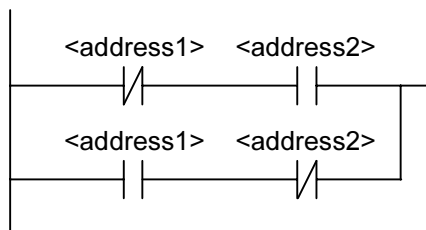
输入端 I0.0 和 I0.1 的信号状态为“1”时

或输入端 I0.2 的信号状态为“1”时

## 1.4 XOR 位异或运算

对于 XOR 函数，必须按以下所示创建由常开触点和常闭触点组成的程序段。

### 符号

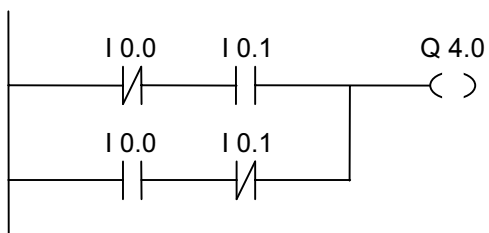


参数	数据类型	存储区	描述
<address1>	BOOL	I、Q、M、L、 D、T、C	扫描的位
<address2>	BOOL	I、Q、M、L、 D、T、C	扫描的位

### 描述

**XOR** (逻辑“异或”)如果两个指定位的信号状态不同，则创建状态为“1”的 RLO。

### 实例



如果(I0.0 = “0” 且 I0.1 = “1”)或者(I0.0 = “1” 且 I0.1 = “0”), 输出 Q4.0 将是“1”。

## 1.5 --|NOT|-- 能流取反

符号

---| NOT |---

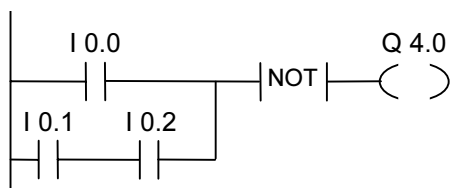
描述

---|NOT|--- (能流取反)取反 RLO 位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	-	1	X	-

实例



满足下列条件之一时，输出端 Q4.0 的信号状态将是“0”：

输入端 I0.0 的信号状态为“1”时

或当输入端 I0.1 和 I0.2 的信号状态为“1”时。

## 1.6 ---( ) 输出线圈

### 符号

<地址>

---( )

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D	分配位

### 描述

---( ) (输出线圈)的工作方式与继电器逻辑图中线圈的工作方式类似。如果有能流通过线圈(RLO = 1)，将置位<地址>位置的位为“1”。如果没有能流通过线圈(RLO = 0)，将置位<地址>位置的位为“0”。只能将输出线圈置于梯级的右端。可以有多个(最多 16 个)输出单元(请参见实例)。使用 ---|NOT|--- (能流取反)单元可以创建取反输出。

### MCR (主站控制继电器)依存

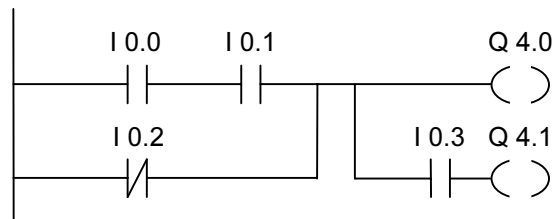
只有在将输出线圈置于激活的 MCR 区内时，才会激活 MCR 依存关系。在激活的 MCR 区内，如果 MCR 处于接通状态并且输出线圈有能流通过，将把寻址位设置为能流的当前状态。如果 MCR 处于断开状态，则无论能流状态如何，都会将逻辑“0”写入指定地址。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	X	-	0



## 实例



满足下列条件之一时，输出端 Q4.0 的信号状态将是“1”：

输入端 I0.0 和 I0.1 的信号状态为“1”时

或输入端 I0.2 的信号状态为“0”时。

满足下列条件之一时，输出端 Q4.1 的信号状态将是“1”：

输入端 I0.0 和 I0.1 的信号状态为“1”时

或输入端 I0.2 的信号状态为“0”、输入端 I0.3 的信号状态为“1”时

**如果实例梯级在激活的 MCR 区之内：**

MCR 处于接通状态时，将按照上述能流状态置位 Q4.0 和 Q4.1。

MCR 处于断开状态(=0)时，无论是否有能流通过，都将 Q4.0 和 Q4.1 复位为 0。

## 1.7 ---(# )--- 中线输出

### 符号

<地址>

---(# )---

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、*L、D	分配位

\* 只有在逻辑块(FC、FB、OB)的变量声明表中将 L 区地址声明为 TEMP 时，才能使用 L 区地址。

### 描述

---(# )--- (中间输出)是中间分配单元，它将 RLO 位状态(能流状态)保存到指定<地址>。中间输出单元保存前面分支单元的逻辑结果。以串联方式与其它触点连接时，可以像插入触点那样插入 ---(# )---。不能将 ---(# )--- 单元连接到电源轨道、直接连接在分支连接的后面或连接在分支的尾部。使用 ---|NOT|--- (能流取反)单元可以创建取反 ---(# )---。

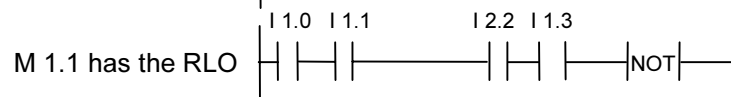
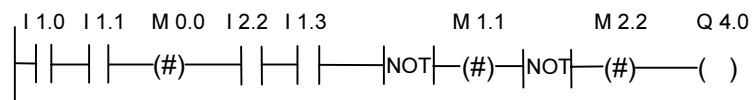
### MCR (主站控制继电器)依存

只有在将中间输出线圈置于激活的 MCR 区内时，才会激活 MCR 依存关系。在激活的 MCR 区内，如果 MCR 处于接通状态并且中间输出线圈有能流通过，将把寻址位设置为能流的当前状态。如果 MCR 处于断开状态，则无论能流状态如何，都会将逻辑“0”写入指定地址。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	X	-	1

实例



M 2.2 has the RLO of the entire bit logic combination

## 1.8 ---( R ) 复位线圈

### 符号

<地址>

---( R )

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D、 T、C	复位

### 描述

只有在前面指令的 RLO 为“1” (能流通过线圈)时, 才会执行 ---( R ) (复位线圈)。如果能流通过线圈(RLO 为“1”), 将把单元的指定<地址>复位为“0”。RLO 为“0” (没有能流通过线圈)将不起作用, 单元指定地址的状态将保持不变。<地址>也可以是值复位为“0”的定时器(T 编号)或值复位为“0”的计数器(C 编号)。

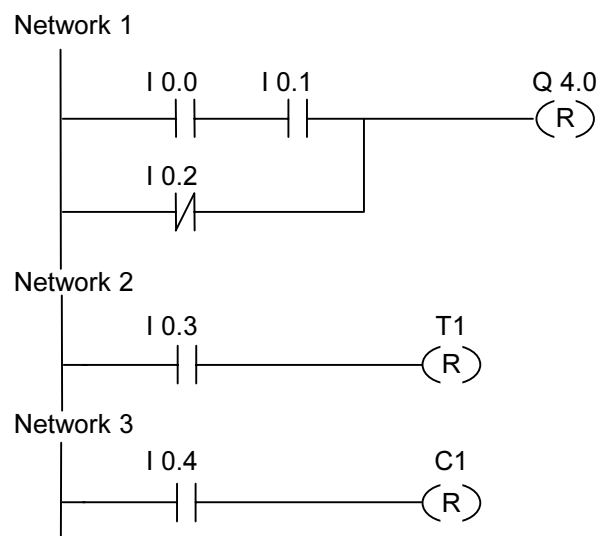
### MCR (主站控制继电器)依存

只有将复位线圈置于激活的 MCR 区内时, 才会激活 MCR 依存。在激活的 MCR 区内, 如果 MCR 处于接通状态并且复位线圈有能流通过, 将把寻址位状态复位为“0”。如果 MCR 处于断开状态, 则无论能流状态如何, 单元指定地址的当前状态均保持不变。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	X	-	0

## 实例



满足下列条件之一时，将把输出端 Q4.0 的信号状态复位为“0”：

输入端 I0.0 和 I0.1 的信号状态为“1”时

或输入端 I0.2 的信号状态为“0”时。

如果 RLO 为“0”，输出端 Q4.0 的信号状态将保持不变。

满足下列条件时才会复位定时器 T1 的信号状态：

输入端 I0.3 的信号状态为“1”时。

满足下列条件时才会复位计数器 C1 的信号状态：

输入端 I0.4 的信号状态为“1”时。

**如果实例梯级在激活的 MCR 区之内：**

MCR 处于接通状态时，将按以上所述复位 Q4.0、T1 和 C1。

MCR 处于断开状态时，无论 RLO 的状态(能流状态)如何，Q4.0、T1 和 C1 的状态均保持不变。

## 1.9 ---( S ) 置位线圈

### 符号

<地址>

---( S )

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D	置位

### 描述

只有在前面指令的 RLO 为“1” (能流通过线圈)时，才会执行 ---( S ) (置位线圈)。如果 RLO 为“1”，将把单元的指定<地址>置位为“1”。

RLO = 0 将不起作用，单元的指定地址的当前状态将保持不变。

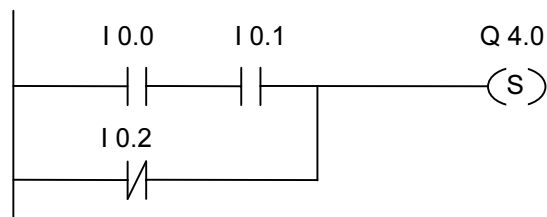
### MCR (主站控制继电器)依存

只有将置位线圈置于激活的 MCR 区内时，才会激活 MCR 依存关系。在激活的 MCR 区内，如果 MCR 处于接通状态并且置位线圈有能流通过，将把寻址位的状态置位为“1”。如果 MCR 处于断开状态，则无论能流状态如何，单元指定地址的当前状态均保持不变。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	X	-	0

## 实例



满足下列条件之一时，输出端 Q4.0 的信号状态将是“1”：

输入端 I0.0 和 I0.1 的信号状态为“1”时

或输入端 I0.2 的信号状态为“0”时。

如果 RLO 为“0”，输出端 Q4.0 的信号状态将保持不变。

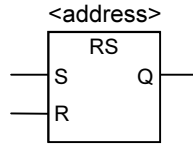
**如果实例梯级在激活的 MCR 区之内：**

MCR 处于接通状态时，则按以上所述置位 Q4.0。

MCR 处于断开状态时，无论 RLO 状态(能流状态)如何，Q4.0 状态均保持不变。

## 1.10 RS 置位优先型 RS 双稳态触发器

符号



参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D	置位或复位位
S	BOOL	I、Q、M、L、D	启用复位指令
R	BOOL	I、Q、M、L、D	启用复位指令
Q	BOOL	I、Q、M、L、D	<地址>的信号状态

描述

如果 R 输入端的信号状态为“1”，S 输入端的信号状态为“0”，则复位 RS (置位优先型 RS 双稳态触发器)。否则，如果 R 输入端的信号状态为“0”，S 输入端的信号状态为“1”，则置位触发器。如果两个输入端的 RLO 状态均为“1”，则指令的执行顺序是最重要的。RS 触发器先在指定<地址>执行复位指令，然后执行置位指令，以使该地址在执行余下的程序扫描过程中保持置位状态。

只有在 RLO 为“1”时，才会执行 S (置位)和 R (复位)指令。这些指令不受 RLO “0”的影响，指令中指定的地址保持不变。

### MCR (主站控制继电器)依存

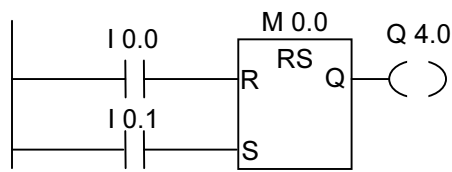
只有将 RS 触发器置于激活的 MCR 区内时，才会激活 MCR 依存关系。在激活的 MCR 区内，如果 MCR 处于接通状态，则按以上所述将寻址位复位为“0”或置位为“1”。如果 MCR 处于断开状态，则无论输入状态如何，指定地址的当前状态均保持不变。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1



## 实例



如果输入端 I0.0 的信号状态为“1”，I0.1 的信号状态为“0”，则置位存储器位 M0.0，输出 Q4.0 将是“0”。否则，如果输入端 I0.0 的信号状态为“0”，I0.1 的信号状态为“1”，则复位存储器位 M0.0，输出 Q4.0 将是“1”。如果两个信号状态均为“0”，则不会发生任何变化。如果两个信号状态均为“1”，将因顺序关系执行置位指令；置位 M0.0，Q4.0 将是“1”。

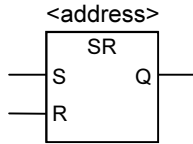
**如果实例在激活的 MCR 区之内：**

MCR 处于接通状态时，将按以上所述复位或置位 Q4.0。

MCR 处于断开状态时，无论输入状态如何，Q4.0 均保持不变。

## 1.11 SR 复位优先型 SR 双稳态触发器

符号



参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D	置位或复位位
S	BOOL	I、Q、M、L、D	启用置位指令
R	BOOL	I、Q、M、L、D	启用复位指令
Q	BOOL	I、Q、M、L、D	<地址>的信号状态

描述

如果 S 输入端的信号状态为“1”，R 输入端的信号状态为“0”，则置位 SR (复位优先型 SR 双稳态触发器)。否则，如果 S 输入端的信号状态为“0”，R 输入端的信号状态为“1”，则复位触发器。如果两个输入端的 RLO 状态均为“1”，则指令的执行顺序是最重要的。SR 触发器先在指定<地址>执行置位指令，然后执行复位指令，以使该地址在执行余下的程序扫描过程中保持复位状态。

只有在 RLO 为“1”时，才会执行 S (置位)和 R (复位)指令。这些指令不受 RLO “0”的影响，指令中指定的地址保持不变。

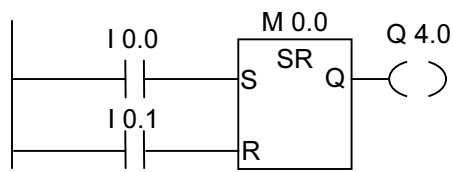
### MCR (主站控制继电器)依存

只有将 SR 触发器置于激活的 MCR 区内时，才会激活 MCR 依存关系。在激活的 MCR 区内，如果 MCR 处于接通状态，则按以上所述将寻址位置位为“1”或复位为“0”。如果 MCR 处于断开状态，则无论输入状态如何，指定地址的当前状态均保持不变。

状态字

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

## 实例



如果输入端 I0.0 的信号状态为“1”，I0.1 的信号状态为“0”，则置位存储器位 M0.0，输出 Q4.0 将是“1”。否则，如果输入端 I0.0 的信号状态为“0”，I0.1 的信号状态为“1”，则复位存储器位 M0.0，输出 Q4.0 将是“0”。如果两个信号状态均为“0”，则不会发生任何变化。如果两个信号状态均为“1”，将因顺序关系执行复位指令；复位 M0.0，Q4.0 将是“0”。

**如果实例在激活的 MCR 区之内：**

MCR 处于接通状态时，将按以上所述置位或复位 Q4.0。

MCR 处于断开状态时，无论输入状态如何，Q4.0 均保持不变。

## 1.12 ---( N )--- RLO 负跳沿检测

### 符号

<地址>

---( N )

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D	边沿存储位，存储 RLO 的上一信号状态

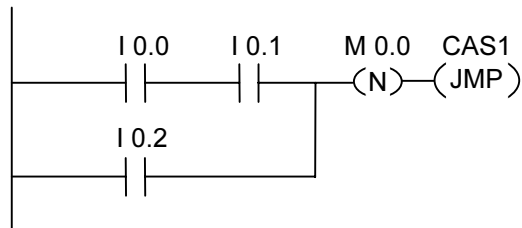
### 描述

---( N )--- (RLO 负跳沿检测)检测地址中“1”到“0”的信号变化，并在指令后将其显示为 RLO = “1”。将 RLO 中的当前信号状态与地址的信号状态(边沿存储位)进行比较。如果在执行指令前地址的信号状态为“1”，RLO 为“0”，则在执行指令后 RLO 将是“1”(脉冲)，在所有其它情况下将是“0”。指令执行前的 RLO 状态存储在地址中。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	x	x	1

### 实例



边沿存储位 M0.0 保存 RLO 的先前状态。RLO 的信号状态从“1”变为“0”时，程序将跳转到标号 CAS1。

## 1.13 ---( P )--- RLO 正跳沿检测

### 符号

<地址>

---( P )---

参数	数据类型	存储区	描述
<地址>	BOOL	I、Q、M、L、D	边沿存储位，存储 RLO 的上一信号状态

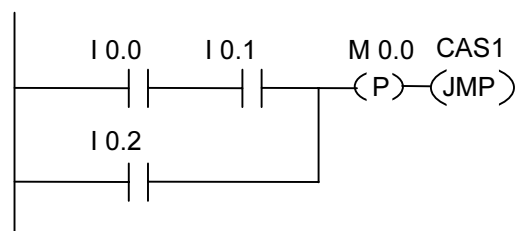
### 描述

---( P )--- (RLO 正跳沿检测)检测地址中“0”到“1”的信号变化，并在指令后将其显示为 RLO = “1”。将 RLO 中的当前信号状态与地址的信号状态(边沿存储位)进行比较。如果在执行指令前地址的信号状态为“0”，RLO 为“1”，则在执行指令后 RLO 将是“1” (脉冲)，在所有其它情况下将是“0”。指令执行前的 RLO 状态存储在地址中。

### 状态字

	BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	X	X	1

### 实例



边沿存储位 M0.0 保存 RLO 的先前状态。RLO 的信号状态从“0”变为“1”时，程序将跳转到标号 CAS1。

## 1.14 ---(SAVE) 将 RLO 保存到 BR 存储器中

符号

---( SAVE )

描述

---(SAVE) (将 RLO 状态保存到 BR)将 RLO 保存到状态字的 BR 位。第一个校验位 /FC 不复位。因此，BR 位的状态包括在下一程序段中的与逻辑运算内。

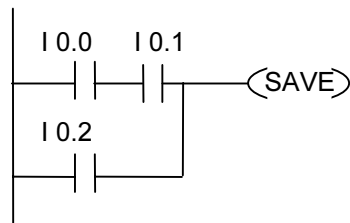
指令“SAVE” (LAD、FBD、STL)适用下列规则，手册及在线帮助中提供的建议用法并不适用：

建议用户不要在使用 SAVE 后在同一块或从属块中校验 BR 位，因为这期间执行的指令中有许多会对 BR 位进行修改。建议用户在退出块前使用 SAVE 指令，因为 ENO 输出(= BR 位)届时已设置为 RLO 位的值，所以可以检查块中是否有错误。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	X	-	-	-	-	-	-	-	-

实例

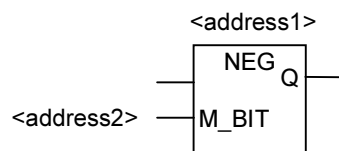


将梯级(=RLO)的状态保存到 BR 位。

BR 二进制结果位(状态字，第 8 位)

## 1.15 NEG 地址下降沿检测

符号



参数	数据类型	存储区	描述
<address1>	BOOL	I、Q、M、L、D	已扫描信号
<address2>	BOOL	I、Q、M、L、D	M_BIT 边沿存储位, 存储 <address1> 的前一个信号状态
Q	BOOL	I、Q、M、L、D	单触发输出

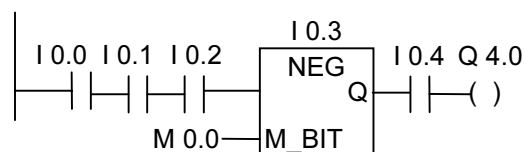
描述

**NEG** (地址下降沿检测)比较<address1>的信号状态与前一次扫描的信号状态(存储在<address2>中)。如果当前 RLO 状态为“1”且其前一状态为“0”(检测到上升沿), 执行此指令后 RLO 位将是“1”。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	1	x	1

实例

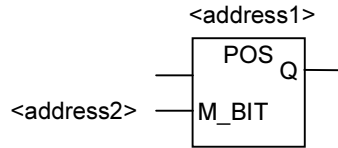


满足下列条件时, 输出 Q4.0 的信号状态将是“1”:

- 输入 I0.0、I0.1 和 I0.2 的信号状态是“1”
- 输入 I0.3 有下降沿
- 输入 I0.4 的信号状态为“1”

## 1.16 POS 地址上升沿检测

符号



参数	数据类型	存储区	描述
<address1>	BOOL	I、Q、M、L、D	已扫描信号
<address2>	BOOL	I、Q、M、L、D	M_BIT 边沿存储位, 存储 <address1> 的前一个信号状态
Q	BOOL	I、Q、M、L、D	单触发输出

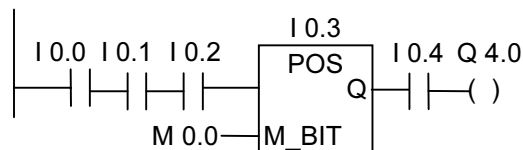
描述

**POS** (地址上升沿检测)比较<address1>的信号状态与前一次扫描的信号状态(存储在<address2>中)。如果当前 RLO 状态为“1”且其前一状态为“0”(检测到上升沿), 执行此指令后 RLO 位将是“1”。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	1	x	1

实例



满足下列条件时, 输出 Q4.0 的信号状态将是“1”:

- 输入 I0.0、I0.1 和 I0.2 的信号状态是“1”
- 输入 I0.3 有上升沿
- 输入 I0.4 的信号状态为“1”



## 1.17 立即读取

### 描述

对于“立即读取”功能，必须按以下实例所示创建符号程序段。

对于对时间要求苛刻的应用程序，对数字输入的当前状态的读取可能要比正常情况下每 OB1 扫描周期一次的速度快。“立即读取”在扫描“立即读取”梯级时从输入模块中获取数字输入的状态。否则，必须等到下一 OB1 扫描周期结束，届时将以 P 存储器状态更新 I 存储区。

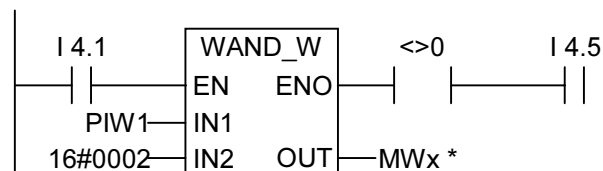
要从输入模块立即读取一个输入(或多个输入)，请使用外设输入(PI)存储区来代替输入(I)存储区。可以字节、字或双字形式读取外设输入存储区。因此，不能通过触点(位)元素读取单一数字输入。

**根据立即输入的状态有条件地传递电压：**

1. CPU 读取包含相关输入数据的 PI 存储器的字。
2. 如果输入位处于接通状态(为“1”)，将对 PI 存储器的字与某个常数执行产生非零结果的 AND 运算。
3. 测试累加器的非零条件。

### 实例

可以立即读取外设输入 I1.1 的梯形图程序段



必须指定\* MWx，才能存储程序段。x 可以是允许的任何数。

**WAND\_W 指令说明：**

PIW1	0000000000101010
W#16#0002	0000000000000010
结果	0000000000000010

在此实例中，立即输入 I1.1 与 I4.1 和 I4.5 串联。

字 PIW1 包含 I1.1 的立即状态。对 PIW1 与 W#16#0002 执行 AND 运算。如果 PB1 中的 I1.1 (第二位)为真(“1”)，则结果不等于零。如果 WAND\_W 指令的结果不等于零，触点 A<>0 时将传递电压。

## 1.18 立即写入

### 描述

对于“立即写入”功能，必须按以下实例所示创建符号程序段。

对于对时间要求苛刻的应用程序，将数字输出的当前状态发送给输出模块的速度可能必须快于正常情况下在 OB1 扫描周期结束时发送一次的速度。“立即写入”将在扫描“立即写入”梯级时将数字输出写入输入模块。否则，必须等到下一 OB1 扫描周期结束，届时将以 P 存储器状态更新 Q 存储区。

要将一个输出(或多个输出)立即写入输出模块，请使用外设输出(PQ)存储区来代替输出(Q)存储区。可以字节、字或双字形式读取外设输出存储区。因此，不能通过线圈单元更新单一数字输出。要立即向输出模块写入数字输出的状态，将根据条件把包含相关位的 Q 存储器的字节、字或双字复制到相应的 PQ 存储器(直接输出模块地址)中。



### 当心

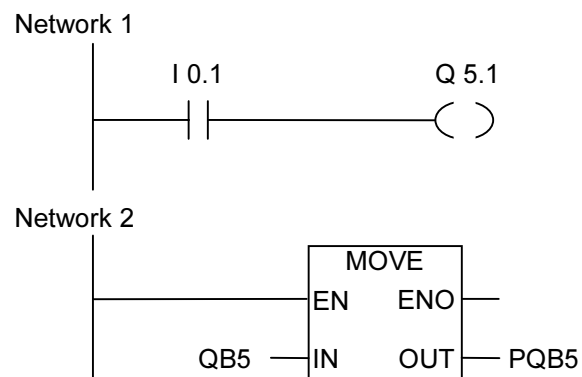
- 由于 Q 存储器的整个字节都写入了输出模块，因此在执行立即输出时，将更新该字节中的所有输出位。
- 如果输出位在程序各处产生了多个中间状态(1/0)，而这些状态不应发送给输出模块，则执行“立即写入”可能会导致危险情况(输出端产生瞬态脉冲)发生。
- 作为常规设计原则，在程序中只能以线圈形式对外部输出模块引用一次。如果用户遵循此设计原则，则可以避免使用立即输出时的大多数潜在问题。

## 实例

立即写入外设数字输出模块 5 通道 1 的等价梯形图程序段。

可以修改寻址输出 Q 字节(QB5)的状态位，也可以将其保持不变。程序段 1 中给 Q5.1 分配 I0.1 信号状态。将 QB5 复制到相应的直接外设输出存储区(PQB5)。

字 PIW1 包含 I1.1 的立即状态。对 PIW1 与 W#16#0002 执行 AND 运算。如果 PB1 中的 I1.1 (第二位)为真(“1”), 则结果不等于零。如果 WAND\_W 指令的结果不等于零, 触点 A<>0 时将传递电压。



在此实例中，Q5.1 为所需的立即输出位。

字节 PQB5 包含 Q5.1 位的立即输出状态。

MOVE(复制)指令还会更新 PQB5 的其它 7 位。



## 2 比较指令

### 2.1 比较指令概述

#### 描述

根据用户选择的比较类型比较 IN1 和 IN2:

== IN1 等于 IN2  
<> IN1 不等于 IN2  
> IN1 大于 IN2  
< IN1 小于 IN2  
>= IN1 大于等于 IN2  
<= IN1 小于等于 IN2

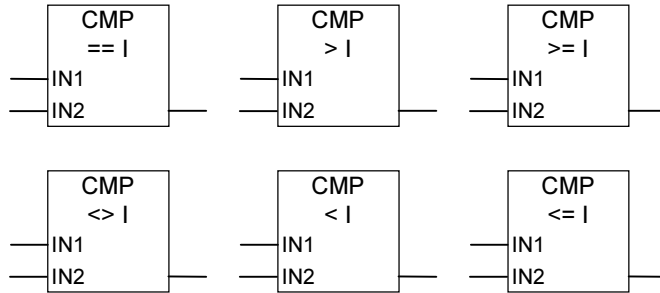
如果比较结果为 true, 则此函数的 RLO 为“1”。如果以串联方式使用比较单元, 则使用“与”运算将其链接至梯级程序段的 RLO; 如果以并联方式使用该框, 则使用“或”运算将其链接至梯级程序段的 RLO。

可使用下列比较指令:

- CMP ? I 整数比较
- CMP ? D 长整数比较
- CMP ? R 实数比较

## 2.2 CMP ? I 整数比较

符号



参数	数据类型	存储区	描述
输入框	BOOL	I、Q、M、L、D	上一逻辑运算的结果
输出框	BOOL	I、Q、M、L、D	比较的结果，仅在输入框的 RLO = 1 时才进一步处理
IN1	INT	I、Q、M、L、D 或常数	要比较的第一个值
IN2	INT	I、Q、M、L、D 或常数	要比较的第二个值

描述

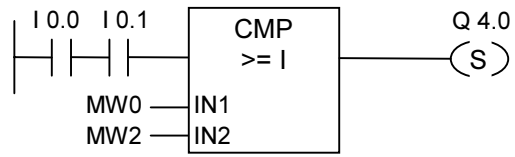
**CMP ? I** (整数比较)的使用方法 与标准触点类似。它可位于任何可放置标准触点的位置。可根据用户选择的比较类型比较 IN1 和 IN2。

如果比较结果为 true，则此函数的 RLO 为“1”。如果以串联方式使用该框，则使用“与”运算将其链接至整个梯级程序段的 RLO；如果以并联方式使用该框，则使用“或”运算将其链接至整个梯级程序段的 RLO。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	0	-	0	x	x	1

## 实例

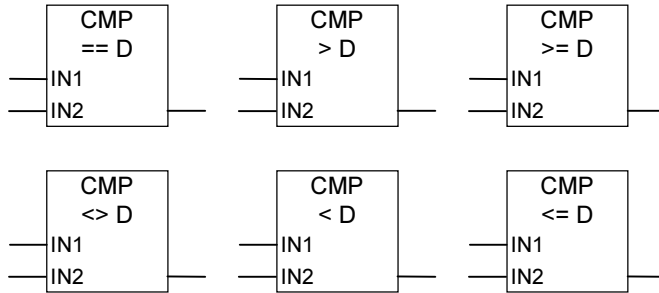


如果满足下列条件，则输出 Q4.0 置位：

- 输入 I0.0 和 I0.1 的信号状态为 “1”
- 并且 MW0 >= MW2

## 2.3 CMP ? D 长整数比较

符号



参数	数据类型	存储区	描述
输入框	BOOL	I、Q、M、L、D	上一逻辑运算的结果
输出框	BOOL	I、Q、M、L、D	比较的结果，仅在输入框的 RLO = 1 时才进一步处理
IN1	DINT	I、Q、M、L、D 或常数	要比较的第一个值
IN2	DINT	I、Q、M、L、D 或常数	要比较的第二个值

描述

**CMP ? D** (长整数比较)的使用方法 与标准触点类似。它可位于任何可放置标准触点的位置。可根据用户选择的比较类型比较 IN1 和 IN2。

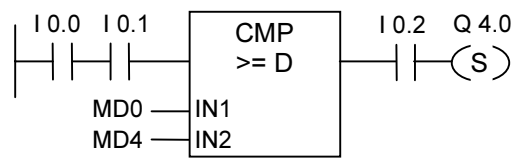
如果比较结果为 true，则此函数的 RLO 为“1”。如果以串联方式使用比较单元，则使用“与”运算将其链接至梯级程序段的 RLO；如果以并联方式使用该框，则使用“或”运算将其链接至梯级程序段的 RLO。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	0	-	0	x	x	1



## 实例

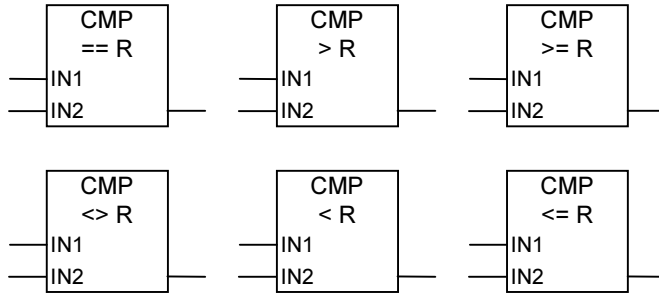


如果满足下列条件，则输出 Q4.0 置位：

- 输入 I0.0 和 I0.1 的信号状态为“1”
- 并且 MD0 >= MD4
- 同时输入 I0.2 的信号状态为“1”

## 2.4 CMP ? R 实数比较

符号



参数	数据类型	存储区	描述
输入框	BOOL	I、Q、M、L、D	上一逻辑运算的结果
输出框	BOOL	I、Q、M、L、D	比较的结果，仅在输入框的 RLO = 1 时才进一步处理
IN1	REAL	I、Q、M、L、D 或常数	要比较的第一个值
IN2	REAL	I、Q、M、L、D 或常数	要比较的第二个值

描述

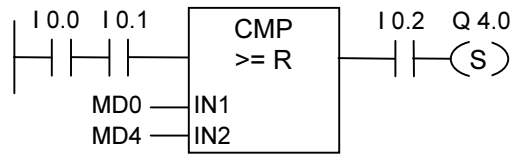
**CMP ? R** (比较实数)的使用方法类似标准触点。它可位于任何可放置标准触点的位置。可根据用户选择的比较类型比较 IN1 和 IN2。

如果比较结果为 true，则此函数的 RLO 为“1”。如果以串联方式使用该框，则使用“与”运算将其链接至整个梯级程序段的 RLO；如果以并联方式使用该框，则使用“或”运算将其链接至整个梯级程序段的 RLO。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

## 实例



如果满足下列条件，则输出 Q4.0 置位：

- 输入 I0.0 和 I0.1 的信号状态为 “1”
- 并且 MD0 >= MD4
- 同时输入 I0.2 的信号状态为 “1”



## 3 转换指令

### 3.1 转换指令概述

#### 描述

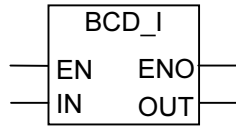
转换指令读取参数 IN 的内容，然后进行转换或改变其符号。可通过参数 OUT 查询结果。

用户可使用下列转换指令：

- BCD\_I      BCD 码转换为整型
- I\_BCD      整型转换为 BCD 码
- BCD\_DI     BCD 码转换为长整型
- I\_DINT     整型转换为长整型
- DI\_BCD     长整型转换为 BCD 码
- DI\_REAL    长整型转换为浮点型
  
- INV\_I      对整数求反码
- INV\_DI     对长整数求反码
- NEG\_I      对整数求补码
- NEG\_DI     对长整数求补码
  
- NEG\_R     浮点数取反
- ROUND     取整为长整型
- TRUNC     截取长整数部分
- CEIL      向上取整
- FLOOR     向上取整

### 3.2 BCD\_I BCD 码转换为整型

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	WORD	I、Q、M、L、D	BCD 码数字
OUT	INT	I、Q、M、L、D	BCD 码数字的整型值

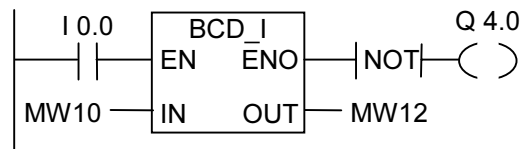
描述

**BCD\_I** (BCD 码转换为整型)将参数 IN 的内容以三位 BCD 码数字(+/- 999)读取，并将其转换为整型值(16 位)。整型值的结果通过参数 OUT 输出。ENO 始终与 EN 的信号状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

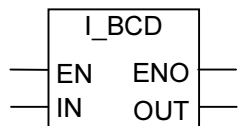
实例



如果输入 I0.0 的状态为“1”，则将 MW10 中的内容以三位 BCD 码数字读取，并将其转换为整型值。结果存储在 MW12 中。如果未执行转换(ENO = EN = 0)，则输出 Q4.0 的状态为“1”。

### 3.3 I\_BCD 整型转换为 BCD 码

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	INT	I、Q、M、L、D	整数
OUT	WORD	I、Q、M、L、D	整数的 BCD 码值

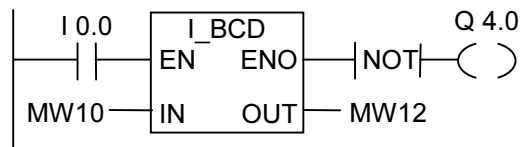
描述

**I\_BCD** (整型转换为 BCD 码)将参数 IN 的内容以整型值(16 位)读取，并将其转换为三位 BCD 码数字(+/- 999)。结果由参数 OUT 输出。如果产生溢出，ENO 的状态为“0”。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	-	-	x	x	0	x	x	1

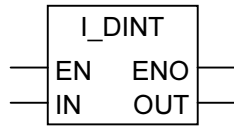
实例



如果 I0.0 的状态为“1”，则将 MW10 的内容以整型值读取，并将其转换为三位 BCD 码数字。结果存储在 MW12 中。如果产生溢出或未执行指令(I0.0 = 0)，则输出 Q4.0 的状态为“1”。

### 3.4 I\_DINT 整型转换为长整型

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	INT	I、Q、M、L、D	要转换的整型值
OUT	DINT	I、Q、M、L、D	长整型结果

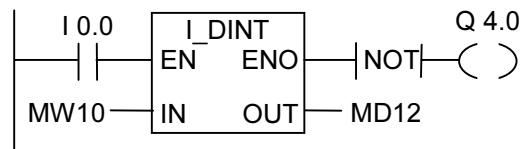
描述

**I\_DINT** (整型转换为长整型)将参数 IN 的内容以整型(16 位)读取, 并将其转换为长整型(32 位)。结果由参数 OUT 输出。ENO 始终与 EN 的信号状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

实例

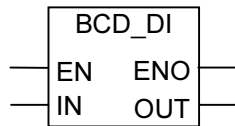


如果 I0.0 为“1”，则 MW10 的内容以整型读取，并将其转换为长整型。结果存储在 MD12 中。如果未执行转换(ENO = EN = 0)，则输出 Q4.0 的状态为“1”。



### 3.5 BCD\_DI BCD 码转换为长整型

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DWORD	I、Q、M、L、D	BCD 码数字
OUT	DINT	I、Q、M、L、D	BCD 码数字的长整型值

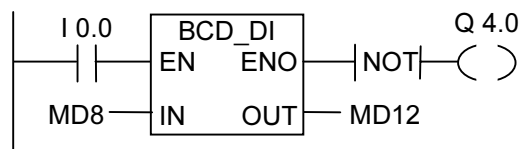
描述

**BCD\_DI** (将 BCD 码转换为长整型)将参数 **IN** 的内容以七位 BCD 码(+/- 9999999)数字读取，并将其转换为长整型值(32 位)。长整型值的结果通过参数 **OUT** 输出。**ENO** 始终与 **EN** 的信号状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

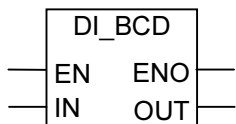
实例



如果 **I0.0** 的状态为“1”，则将 **MD8** 的内容以七位 BCD 码数字读取，并将其转换为长整型值。结果存储在 **MD12** 中。如果未执行转换(**ENO = EN = 0**)，则输出 **Q4.0** 的状态为“1”。

### 3.6 DI\_BCD 长整型转换为 BCD 码

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DINT	I、Q、M、L、D	长整数
OUT	DWORD	I、Q、M、L、D	长整数的 BCD 码值

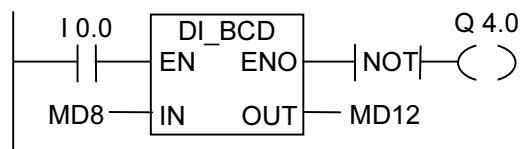
描述

**DI\_BCD** (长整型转换为 BCD 码)将参数 IN 的内容以长整型值(32 位)读取，并将其转换为七位 BCD 码数字 (+/- 9999999)。结果由参数 OUT 输出。如果产生溢出，ENO 的状态为“0”。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	-	-	x	x	0	x	x	1

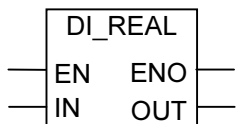
实例



如果 I0.0 的状态为“1”，则将 MD8 的内容以长整型读取，并将其转换为七位 BCD 码数字。结果存储在 MD12 中。如果产生溢出或未执行指令(I0.0 = 0)，则输出 Q4.0 的状态为“1”。

### 3.7 DI\_REAL 长整型转换为浮点型

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DINT	I、Q、M、L、D	要转换的长整型值
OUT	REAL	I、Q、M、L、D	浮点数结果

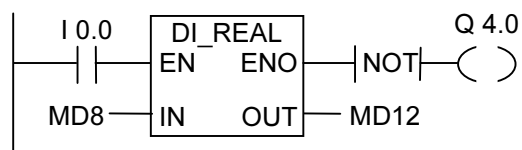
描述

**DI\_REAL** (长整型转换为浮点型)将参数 **IN** 的内容以长整型读取，并将其转换为浮点数。结果由参数 **OUT** 输出。**ENO** 始终与 **EN** 的信号状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

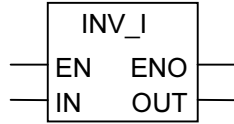
实例



如果 **I 0.0** 的状态为“1”，则将 **MD8** 中的内容以长整型读取，并将其转换为浮点数。结果存储在 **MD12** 中。如果未执行转换(**ENO = EN = 0**)，则输出 **Q 4.0** 的状态为“1”。

### 3.8 INV\_I 对整数求反码

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	INT	I、Q、M、L、D	整型输入值
OUT	INT	I、Q、M、L、D	整型 IN 的二进制反码

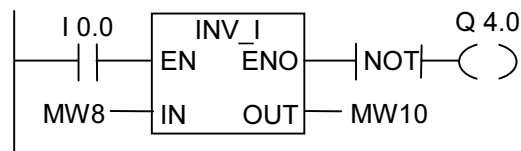
描述

**INV\_I** (对整数求反码)读取 **IN** 参数的内容, 并使用十六进制掩码 **W#16#FFFF** 执行布尔“异或”运算。此指令将每一位变成相反状态。**ENO** 始终与 **EN** 的信号状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

实例



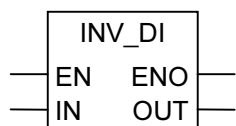
如果 I0.0 为“1”，则将 MW8 的每一位都取反，例如：

MW8 = 01000001 10000001 取反结果为 MW10 = 10111110 01111110。

如果未执行转换(ENO = EN = 0)，则输出 Q4.0 的状态为“1”。

### 3.9 INV\_DI 对长整数求反码

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DINT	I、Q、M、L、D	长整型输入值
OUT	DINT	I、Q、M、L、D	长整型 IN 的二进制反码

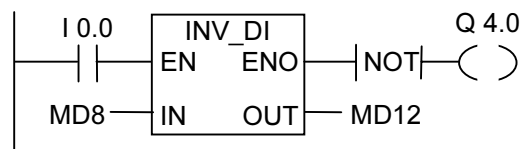
描述

**INV\_DI** (对长整数求反码)读取 IN 参数的内容, 并使用十六进制掩码 **W#16#FFFF FFFF** 执行布尔“异或”运算。此指令将每一位转换为相反状态。ENO 始终与 EN 的信号状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

实例



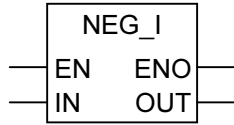
如果 I0.0 为“1”，则 MD8 的每一位都取反，例如：

MD8 = F0FF FFF0 取反结果为 MD12 = 0F00 000F。

如果未执行转换(ENO = EN = 0)，则输出 Q4.0 的状态为“1”。

### 3.10 NEG\_I 对整数求补码

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	INT	I、Q、M、L、D	整型输入值
OUT	INT	I、Q、M、L、D	整型 IN 的二进制补码

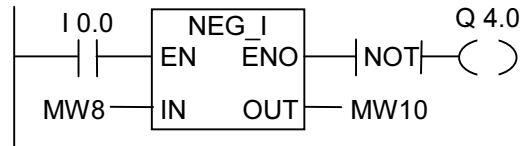
描述

**NEG\_I** (对整数求补码)读取 **IN** 参数的内容并执行求二进制补码指令。二进制补码指令等同于乘以(-1)后改变符号(例如: 从正值变为负值)。**ENO** 始终与 **EN** 的信号状态相同, 以下情况例外: 如果 **EN** 的信号状态 = 1 并产生溢出, 则 **ENO** 的信号状态 = 0。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

实例



如果 I0.0 为“1”, 则由 OUT 参数将 MW8 的值(符号相反)输出到 MW10。

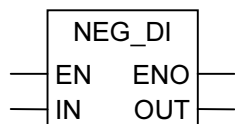
MW8 = + 10 结果为 MW10 = - 10。

如果未执行转换(ENO = EN = 0), 则输出 Q4.0 的状态为“1”。

如果 EN 的信号状态 = 1 并产生溢出, 则 ENO 的信号状态 = 0。

### 3.11 NEG\_DI 对长整数求补码

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DINT	I、Q、M、L、D	长整型输入值
OUT	DINT	I、Q、M、L、D	IN 值的二进制补码

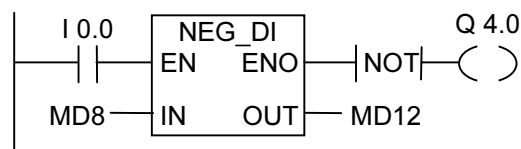
描述

**NEG\_DI** (对长整数求补码)读取参数 **IN** 的内容并执行二进制补码指令。二进制补码指令等同于乘以(-1)后改变符号(例如：从正值变为负值)。**ENO** 始终与 **EN** 的信号状态相同，以下情况例外：如果 **EN** 的信号状态 = 1 并产生溢出，则 **ENO** 的信号状态 = 0。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

实例



如果 I0.0 为“1”，则由 OUT 参数将 MD8 的值(符号相反)输出到 MD12。

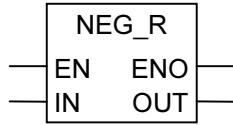
MD8 = + 1000 结果为 MD12 = - 1000。

如果未执行转换(ENO = EN = 0)，则输出 Q4.0 的状态为“1”。

如果 EN 的信号状态 = 1 并产生溢出，则 ENO 的信号状态 = 0。

### 3.12 NEG\_R 浮点数取反

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D	浮点数输入值
OUT	REAL	I、Q、M、L、D	浮点数 IN, 带负号

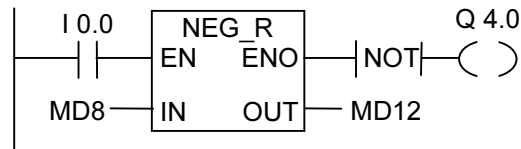
描述

**NEG\_R** (取反浮点)读取参数 IN 的内容并改变符号。指令等同于乘以(-1)后改变符号(例如: 从正值变为负值)。ENO 始终与 EN 的信号状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	-	-	-	-	0	x	x	1

实例



如果 I0.0 为“1”，则由 OUT 参数将 MD8 的值(符号相反)输出到 MD12。

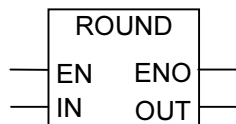
MD8 = + 6.234 结果为 MD12 = - 6.234。

如果未执行转换(ENO = EN = 0)，则输出 Q4.0 的状态为“1”。



### 3.13 ROUND 取整为长整型

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D	要取整的值
OUT	DINT	I、Q、M、L、D	将 IN 取整至最接近的整数

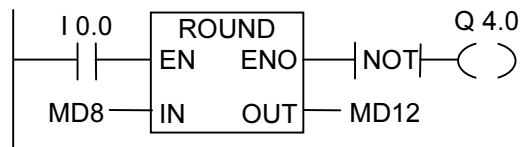
描述

**ROUND** (取整为长整型)将参数 **IN** 的内容以浮点数读取，并将其转换为长整型(32位)。结果为最接近的整数(“取整到最接近值”)。如果浮点数介于两个整数之间，则返回偶数。结果由参数 **OUT** 输出。如果产生溢出，**ENO** 的状态为“0”。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	-	-	x	x	0	x	x	1

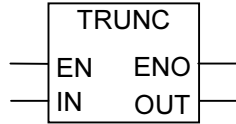
实例



如果 **I0.0** 的状态为“1”，则将 **MD8** 中的内容以浮点数读取，并将其转换为最接近的长整数。函数“取整为最接近值”的结果存储在 **MD12** 中。如果产生溢出或未执行指令(**I0.0** = 0)，则输出 **Q4.0** 的状态为“1”。

### 3.14 TRUNC 截取长整数部分

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D	要转换的浮点型数值
OUT	DINT	I、Q、M、L、D	IN 值的所有数字部分

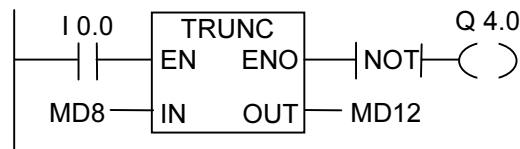
描述

**TRUNC** (截断长整型)将参数 IN 的内容以浮点数读取，并将其转换为长整型(32 位)。(“向零取整模式”)的长整型结果由参数 OUT 输出。如果产生溢出，ENO 的状态为“0”。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	-	-	x	x	0	x	x	1

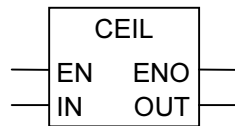
实例



如果 I0.0 的状态为“1”，则将 MD8 中的内容以实型数字读取，并将其转换为长整型值。结果为浮点数的整型部分，并存储在 MD12 中。如果产生溢出或未执行指令 (I0.0 = 0)，则输出 Q4.0 的状态为“1”。

### 3.15 CEIL 向上取整

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D	要转换的浮点型数值
OUT	DINT	I、Q、M、L、D	大于长整型的最小值

描述

**CEIL** (上取整)将参数 **IN** 的内容以浮点数读取，并将其转换为长整型(32 位)。结果为大于该浮点数的最小整数(“取整到 + 无穷大”)。如果产生溢出，**ENO** 的状态为“0”。

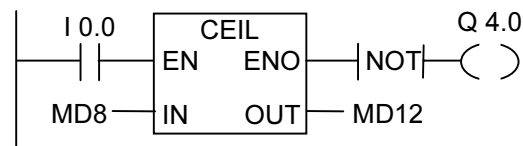
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写*:	X	-	-	X	X	0	X	X	1
写**:	0	-	-	-	-	0	0	0	1

\* 函数执行(EN = 1)

\*\* 函数不执行(EN = 0)

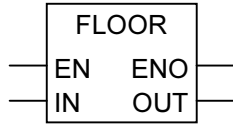
实例



如果 **I0.0** 为 1，则将 **MD8** 的内容以浮点数读取，并使用**取整**函数将其转换为长整型。结果存储在 **MD12** 中。如果出现溢出或未处理指令(**I0.0** = 0)，则输出 **Q4.0** 的状态为“1”。

### 3.16 FLOOR 向下取整

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D	要转换的浮点型数值
OUT	DINT	I、Q、M、L、D	小于长整型的最大值

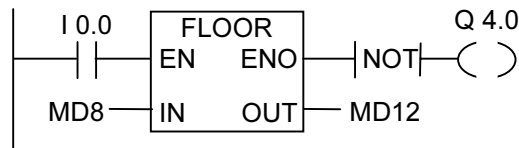
描述

**FLOOR** (下取整)将参数 IN 的内容以浮点数读取，并将其转换为长整型(32 位)。结果为小于该浮点数的最大整数部分(“取整为 - 无穷大”)。如果产生溢出，ENO 的状态为“0”。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	-	-	x	x	0	x	x	1

实例



如果 I0.0 为“1”，则将 MD8 的内容以浮点数读取，并按取整到负无穷大模式将其转换为长整型。结果存储在 MD12 中。如果产生溢出或未执行指令(I0.0 = 0)，则输出 Q4.0 的状态为“1”。

## 4 计数器指令

### 4.1 计数器指令概述

#### 存储器中的区域

在 CPU 存储器中，有为计数器保留的区域。此存储区为每个计数器地址保留一个 16 位字。梯形图指令集支持 256 个计数器。

计数器指令是仅有的可访问计数器存储区的函数。

#### 计数值

计数器字中的 0 至 9 位包含二进制代码形式的计数值。当设置某个计数器时，计数值移至计数器字。计数值的范围为 0 至 999。

可使用下列计数器指令在此范围内改变计数值：

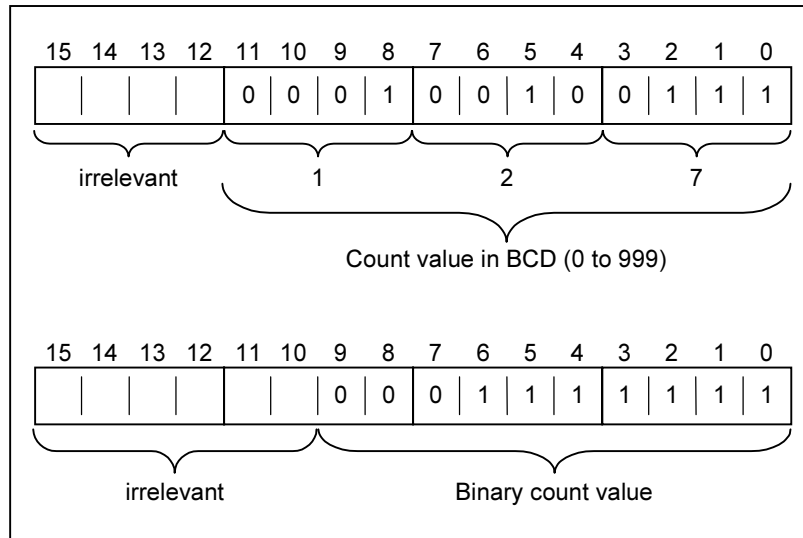
- S\_CUD 双向计数器
- S\_CD 降值计数器
- S\_CU 升值计数器
- ---( SC ) 设置计数器线圈
- ---( CU ) 升值计数器线圈
- ---( CD ) 降值计数器线圈

### 计数器中的位组态

输入从 0 至 999 的数字，用户可为计数器提供预设值，例如，使用下列格式输入 127: C#127。其中 C#代表二进制编码十进制格式(BCD 格式：由四位组成的每一组都包含一个十进制值的二进制代码)。

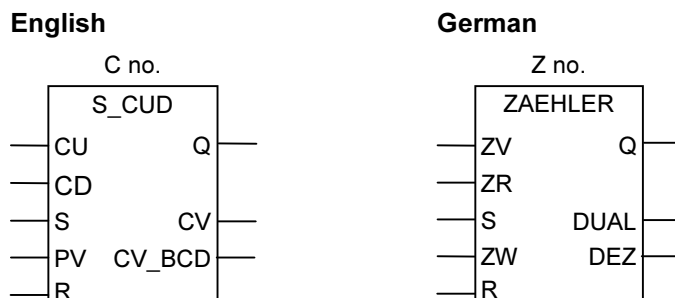
计数器中的 0 至 11 位包含二进制编码十进制格式的计数值。

下图显示了加载计数值 127 之后计数器的内容，以及设置计数器之后计数器单元中的内容。



## 4.2 S\_CUD 双向计数器

符号



参数英语	参数德语	数据类型	存储区	描述
C 编号	Z 编号	COUNTER	C	计数器标识号，其范围依赖于 CPU
CU	ZV	BOOL	I、Q、M、 L、D	升值计数输入
CD	ZR	BOOL	I、Q、M、 L、D	降值计数输入
S	S	BOOL	I、Q、M、 L、D	为预设计数器设置输入
PV	ZW	WORD	I、Q、M、 L、D 或常数	将计数器值以“C#<值>”的格式输入(范围 0 至 999)
PV	ZW	WORD	I、Q、M、 L、D	预设计数器的值
R	R	BOOL	I、Q、M、 L、D	复位输入
CV	DUAL	WORD	I、Q、M、 L、D	当前计数器值，十六进制数字
CV_BCD	DEZ	WORD	I、Q、M、 L、D	当前计数器值，BCD 码
Q	Q	BOOL	I、Q、M、 L、D	计数器状态

### 描述

如果输入 S 有上升沿，S\_CUD(双向计数器)预置为输入 PV 的值。如果输入 R 为 1，则计数器复位，并将计数值设置为零。如果输入 CU 的信号状态从“0”切换为“1”，并且计数器的值小于“999”，则计数器的值增 1。如果输入 CD 有上升沿，并且计数器的值大于“0”，则计数器的值减 1。

如果两个计数输入都有上升沿，则执行两个指令，并且计数值保持不变。

如果已设置计数器，并且输入 CU/CD 的 RLO = 1，则即使没有从上升沿到下降沿或下降沿到上升沿的切换，计数器也会在下一个扫描周期进行相应的计数。

如果计数值大于等于零(“0”)，则输出 Q 的信号状态为“1”。

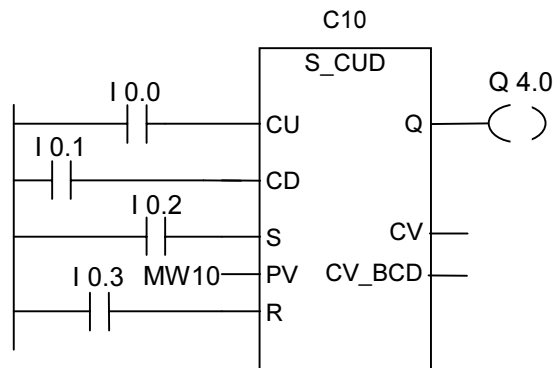
### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

### 注意

避免在多个程序点使用同一计数器(可能出现计数出错)。

### 实例

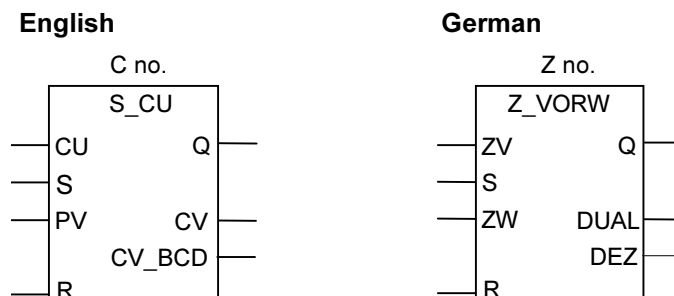


如果 I0.2 从“0”改变为“1”，则计数器预置为 MW10 的值。如果 I0.0 的信号状态从“0”改变为“1”，则计数器 C10 的值将增加 1 - 当 C10 的值等于“999”时除外。如果 I0.1 从“0”改变为“1”，则 C10 减少 1 - 但当 C10 的值为“0”时除外。如果 C10 不等于零，则 Q4.0 为“1”。



## 4.3 S\_CU 升值计数器

符号



参数英语	参数德语	数据类型	存储区	描述
C 编号	Z 编号	COUNTER	C	计数器标识号，其范围依赖于 CPU
CU	ZV	BOOL	I、Q、M、L、D	升值计数输入
S	S	BOOL	I、Q、M、L、D	为预设计数器设置输入
PV	ZW	WORD	I、Q、M、L、D 或常数	将计数器值以“C#<值>”的格式输入(范围 0 至 999)
PV	ZW	WORD	I、Q、M、L、D	预设计数器的值
R	R	BOOL	I、Q、M、L、D	复位输入
CV	DUAL	WORD	I、Q、M、L、D	当前计数器值，十六进制数字
CV_BCD	DEZ	WORD	I、Q、M、L、D	当前计数器值，BCD 码
Q	Q	BOOL	I、Q、M、L、D	计数器状态

描述

如果输入 S 有上升沿，则 **S\_CU**(升值计数器)预置为输入 PV 的值。

如果输入 R 为“1”，则计数器复位，并将计数值设置为零。

如果输入 CU 的信号状态从“0”切换为“1”，并且计数器的值小于“999”，则计数器的值增 1。

如果已设置计数器，并且输入 CU 的 RLO = 1，则即使没有从上升沿到下降沿或下降沿到上升沿的切换，计数器也会在下一个扫描周期进行相应的计数。

如果计数值大于等于零(“0”)，则输出 Q 的信号状态为“1”。

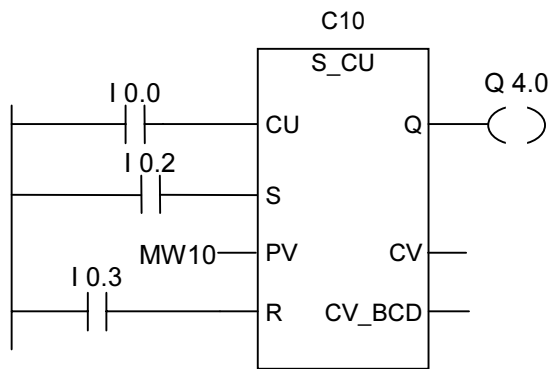
### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

### 注意

避免在多个程序点使用同一计数器(可能出现计数出错)。

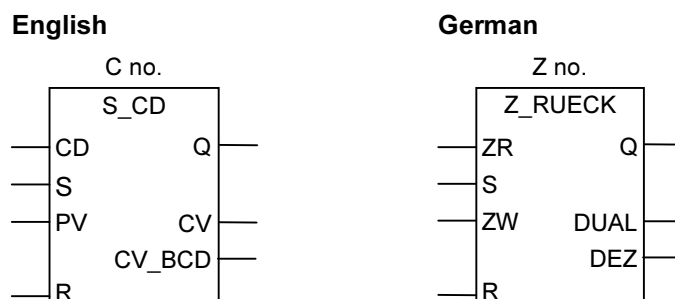
### 实例



如果 I0.2 从“0” 改变为“1”，则计数器预置为 MW10 的值。如果 I0.0 的信号状态从“0” 改变为“1”，则计数器 C10 的值将增加 1 - 当 C10 的值等于“999” 时除外。如果 C10 不等于零，则 Q4.0 为“1”。

## 4.4 S\_CD 降值计数器

符号



参数英语	参数德语	数据类型	存储区	描述
C 编号	Z 编号	COUNTER	C	计数器标识号, 其范围依赖于 CPU
CD	ZR	BOOL	I、Q、M、L、D	降值计数输入
S	S	BOOL	I、Q、M、L、D	为预设计数器设置输入
PV	ZW	WORD	I、Q、M、L、D 或常数	将计数器值以“C#<值>”的格式输入(范围 0 至 999)
PV	ZW	WORD	I、Q、M、L、D	预设计数器的值
R	R	BOOL	I、Q、M、L、D	复位输入
CV	DUAL	WORD	I、Q、M、L、D	当前计数器值, 十六进制数字
CV_BCD	DEZ	WORD	I、Q、M、L、D	当前计数器值, BCD 码
Q	Q	BOOL	I、Q、M、L、D	状态计数器

描述

如果输入 S 有上升沿, 则 S\_CD (降值计数器) 设置为输入 PV 的值。

如果输入 R 为 1, 则计数器复位, 并将计数值设置为零。

如果输入 CD 的信号状态从“0”切换为“1”, 并且计数器的值大于零, 则计数器的值减 1。

如果已设置计数器, 并且输入 CD 的 RLO = 1, 则即使没有从上升沿到下降沿或下降沿到上升沿的改变, 计数器也会在下一个扫描周期进行相应的计数。

如果计数值大于等于零(“0”), 则输出 Q 的信号状态为“1”。

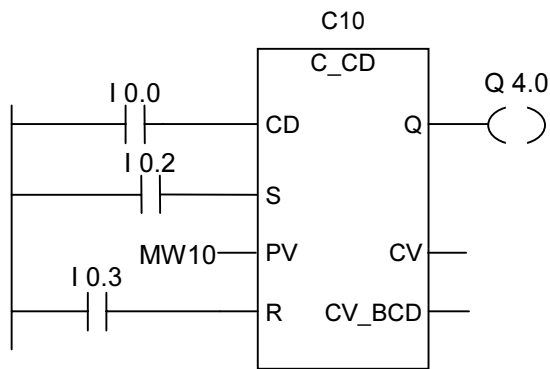
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

注意

避免在多个程序点使用同一计数器(可能出现计数出错)。

实例



如果 I0.2 从“0” 改变为“1”，则计数器预置为 MW10 的值。如果 I0.0 的信号状态从“0” 改变为“1”，则计数器 C10 的值将减 1 - 当 C10 的值等于“0” 时除外。如果 C10 不等于零，则 Q4.0 为“1”。

## 4.5 ---( SC ) 设置计数器值

### 符号

英语	德语
<C 编号>	<Z 编号>
---( SC )	---( SZ )
<预设值>	<预设值>

参数英语	参数德语	数据类型	存储区	描述
<C 编号>	<Z 编号>	COUNTER	C	要预置的计数器编号
<预设值>	<预设值>	WORD	I、Q、M、 L、D 或常数	预置 BCD 的值 (0 至 999)

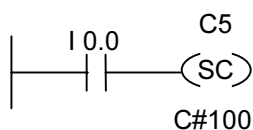
### 描述

仅在 RLO 中有上升沿时，---( SC ) (设置计数器值) 才会执行。此时，预设值被传送至指定的计数器。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	-	-	-	-	0	x	-	0

### 实例



如在 I0.0 有上升沿(从“0”改变为“1”), 则计数器 C5 预置为 100。如果没有上升沿, 则计数器 C5 的值保持不变。

## 4.6 ---( CU ) 升值计数器线圈

### 符号

英语	德语
<C 编号>	<Z 编号>
---( CU )	---( ZV )

参数英语	参数德语	数据类型	存储区	描述
<C 编号>	<Z 编号>	COUNTER	C	计数器标识号，其范围依赖于 CPU

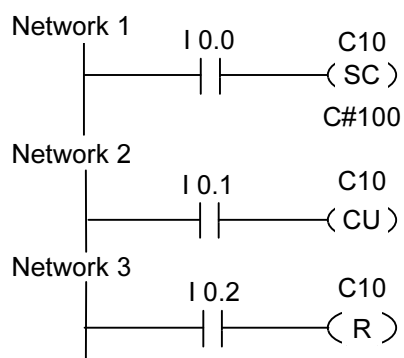
### 描述

如在 RLO 中有上升沿，并且计数器的值小于“999”，则---( CU ) (升值计数器线圈) 将指定计数器的值加 1。如果 RLO 中没有上升沿，或者计数器的值已经是“999”，则计数器值不变。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	-	-	0

## 实例



如果输入 I0.0 的信号状态从“0”改变为“1” (RLO 中有上升沿), 则将预设值 100 载入计数器 C10。

如果输入 I0.1 的信号状态从“0”改变为“1” (在 RLO 中有上升沿), 则计数器 C10 的计数值将增加 1, 但当 C10 的值等于“999”时除外。如果 RLO 中没有上升沿, 则 C10 的值保持不变。

如果 I0.2 的信号状态为“1”, 则计数器 C10 复位为“0”。

## 4.7 ---( CD ) 降值计数器线圈

### 符号

英语	德语
<C 编号>	<Z 编号>
---( CD )	---( ZD )

参数英语	参数德语	数据类型	存储区	描述
<C 编号>	<Z 编号>	COUNTER	C	计数器标识号，其范围依赖于 CPU

### 描述

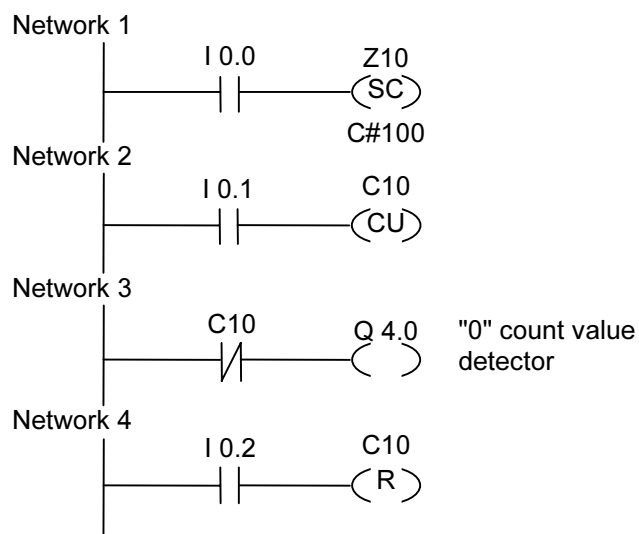
如果 RLO 状态中有上升沿，并且计数器的值大于“0”，则---( CD ) (降值计数器线圈)将指定计数器的值减 1。如果 RLO 中没有上升沿，或者计数器的值已经是“0”，则计数器值不变。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	-	-	0



## 实例



如果输入 I0.0 的信号状态从“0”改变为“1” (RLO 中有上升沿), 则将预设值 100 载入计数器 C10。

如果输入 I0.1 的信号状态从“0”改变为“1” (在 RLO 中有上升沿), 则计数器 C10 的计数值将减 1, 但当 C10 的值等于“0”时除外。如果 RLO 中没有上升沿, 则 C10 的值保持不变。

如果计数值 = 0, 则接通 Q4.0。

如果输入 I0.2 的信号状态为“1”, 则计数器 C10 复位为“0”。



## 5 数据块指令

### 5.1 ---(OPN)打开数据块：DB 或 DI

#### 符号

<DB 编号> 或 <DI 编号>

---(OPN)

参数	数据类型	存储区	描述
<DB 编号> <DI 编号>	BLOCK_DB	DB、DI	DB/DI 编号；编号范围取决于 CPU

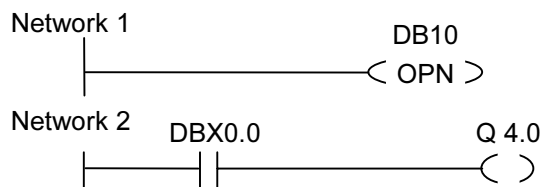
#### 描述

---(OPN) (打开数据块)打开共享数据块(DB)或实例数据块(DI)。---(OPN)函数是一种对数据块的无条件调用。将数据块的编号传送到 DB 或 DI 寄存器中。后续的 DB 和 DI 命令根据寄存器内容访问相应的块。

#### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	-	-	-	-

#### 实例



打开数据块 10 (DB10)。触点地址(DBX0.0)引用包含在 DB10 中的当前数据记录的数据字节零的第零位。将此位的信号状态分配给输出 Q4.0。



## 6 逻辑控制指令

### 6.1 逻辑控制指令概述

#### 描述

可以在所有逻辑块 (组织块(OB)、功能块(FB)和功能(FC))中使用逻辑控制指令。

有可以执行下列功能的逻辑控制指令：

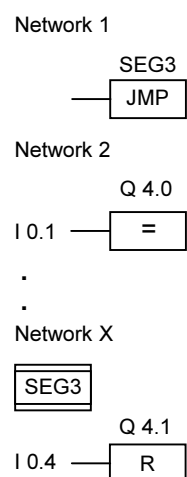
- ---(JMP)--- 无条件跳转
- ---(JMP)--- 条件跳转
- ---(JMPN)--- 若“否”则跳转

#### 标号作为地址

跳转指令的地址是标号。标号最多可以包含四个字符。第一个字符必须是字母表中的字母，其它字符可以是字母或数字(例如，SEG3)。跳转标号指示程序将要跳转到的目标。

#### 标号作为目标

目标标号必须位于程序段的开头。可以通过从梯形图浏览器中选择 LABEL，在程序段的开头输入目标标号。在显示的空框中，键入标号的名称。



## 6.2 ---(JMP)--- 无条件跳转

### 符号

<标号名称>

---(JMP)

### 描述

---(JMP) (为 1 时在块内跳转)当左侧电源轨道与指令间没有其它梯形图元素时执行的是绝对跳转。(请参见示例)。

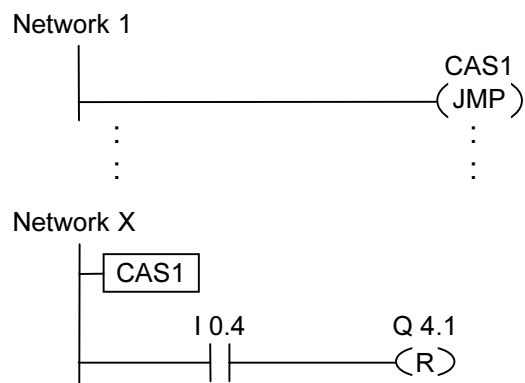
每一个 ---(JMP)都还必须有与之对应的目标(LABEL)。

跳转指令和标号间的所有指令都不予执行。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	-	-	-	-

### 实例



始终执行跳转，并忽略跳转指令和跳转标号间的指令。

## 6.3 ---(JMP)--- 条件跳转

### 符号

<标号名称>

---(JMP)

### 描述

---(JMP) (为 1 时在块内跳转)当前一逻辑运算的 RLO 为 “1” 时执行的是条件跳转。

每一个 ---(JMP)都还必须有与之对应的目标(LABEL)。

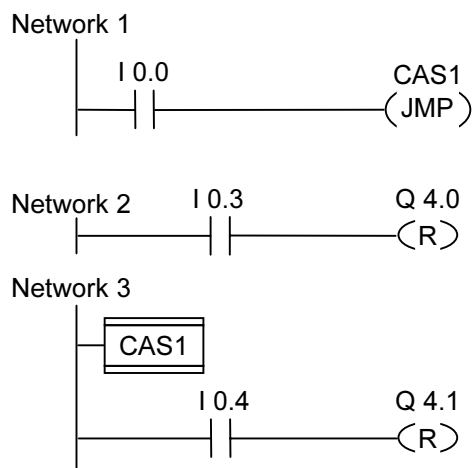
跳转指令和标号间的所有指令都不予执行。

如果未执行条件跳转，RLO 将在执行跳转指令后变为 “1”。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	1	1	0

### 实例



如果 I0.0 = “1”，则执行跳转到标号 CAS1。由于此跳转的存在，即使 I0.3 处有逻辑 “1”，也不会执行复位输出 Q4.0 的指令。

## 6.4 ---(JMPN)若“否”则跳转

### 符号

<标号名称>

---(JMPN)

### 描述

---(JMPN) (若“否”则跳转)相当于在 RLO 为“0”时执行的“转到标号”功能。

每一个 ---(JMPN)都还必须有与之对应的目标(LABEL)。

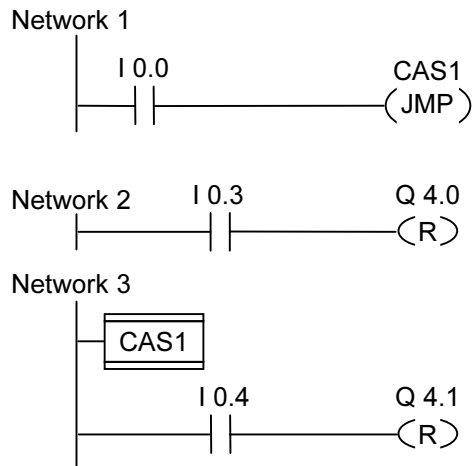
跳转指令和标号间的所有指令都不予执行。

如果未执行条件跳转，RLO 将在执行跳转指令后变为“1”。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	1	1	0

### 实例



如果 I0.0 = “0”，则执行跳转到标号 CAS1。由于此跳转的存在，即使 I0.3 处有逻辑“1”，也不会执行复位输出 Q4.0 的指令。



## 6.5 LABEL 标号

符号



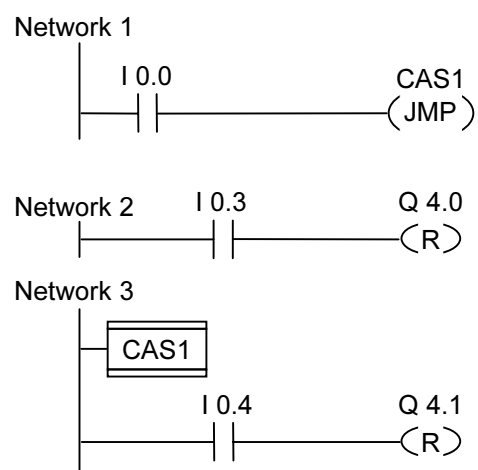
描述

**LABEL** 是跳转指令目标的标识符。

第一个字符必须是字母表中的字母；其它字符可以是字母或数字(例如，CAS1)。

每个 ---(**JMP**)或 ---(**JMPN**)都还必须有与之对应的跳转标号(**LABEL**)。

实例



如果 I0.0 = “1”，则执行跳转到标号 CAS1。由于此跳转的存在，即使 I0.3 处有逻辑 “1”，也不会执行复位输出 Q4.0 的指令。



## 7 整型数学运算指令

### 7.1 整数算术指令概述

#### 描述

使用整数运算，您可以对两个整数(16 和 32 位)执行以下运算：

- ADD\_I 整数加
- SUB\_I 整数减
- MUL\_I 整数乘
- DIV\_I 整数除
  
- ADD\_DI 长整数加
- SUB\_DI 长整数减
- MUL\_DI 长整数乘
- DIV\_DI 长整数除
- MOD\_DI 返回长整数余数

## 7.2 使用整数算术指令时得出状态字的位数值

### 描述

整数运算指令影响状态字中的以下位：CC1 和 CC0，OV 和 OS。

下表显示指令结果为整数(16 位和 32 位)时状态字中各位的信号状态：

有效的结果范围	CC 1	CC 0	OV	OS
0(零)	0	0	0	*
16 位: -32 768 <= 结果 < 0 (负数) 32 位: -2 147 483 648 <= 结果 < 0 (负数)	0	1	0	*
16 位: 32 767 >= 结果 > 0 (正数) 32 位: 2 147 483 647 >= 结果 > 0 (正数)	1	0	0	*

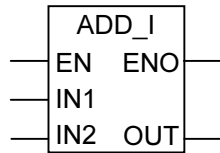
\* 指令结果不影响 OS 位。

无效的结果范围	A1	A0	OV	OS
下溢(加法) 16 位: 结果 = -65536 32 位: 结果 = -4 294 967 296	0	0	1	1
下溢(乘法) 16 位: 结果 < -32 768 (负数) 32 位: 结果 < -2 147 483 648 (负数)	0	1	1	1
溢出(加法、减法) 16 位: 结果 > 32 767 (正数) 32 位: 结果 > 2 147 483 647 (正数)	0	1	1	1
溢出(乘法、除法) 16 位: 结果 > 32 767 (正数) 32 位: 结果 > 2 147 483 647 (正数)	1	0	1	1
下溢(加法、减法) 16 位: 结果 < -32. 768 (负数) 32 位: 结果 < -2 147 483 648 (负数)	1	0	1	1
被 0 除	1	1	1	1

运算	A1	A0	OV	OS
+D: 结果 = -4 294 967 296	0	0	1	1
/D 或 MOD: 除以 0	1	1	1	1

### 7.3 ADD\_I 整数加

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	INT	I、Q、M、L、D 或常数	被加数
IN2	INT	I、Q、M、L、D 或常数	加数
OUT	INT	I、Q、M、L、D	加法结果

描述

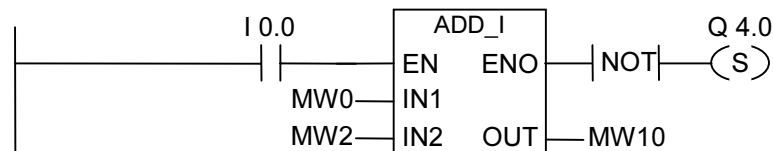
在启用(EN)输入端通过一个逻辑“1”来激活 **ADD\_I** (整数加)。IN1 和 IN2 相加，其结果通过 OUT 来查看。如果该结果超出了整数(16 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

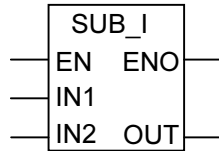
实例



如果 I0.0 = “1”，则激活 **ADD\_I** 框。MW0 + MW2 相加的结果输出到 MW10。如果结果超出整数的允许范围，则设置输出 Q4.0。

## 7.4 SUB\_I 整数减

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	INT	I、Q、M、L、D 或常数	被减数
IN2	INT	I、Q、M、L、D 或常数	减数
OUT	INT	I、Q、M、L、D	减法结果

描述

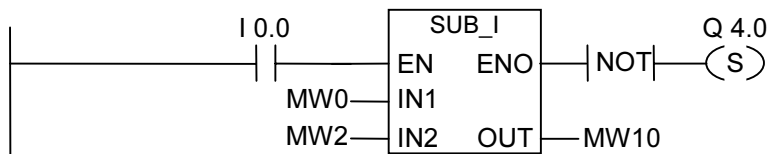
在启用(EN)输入端通过逻辑“1”激活 **SUB\_I** (整数减)。从 IN1 中减去 IN2，并通过 OUT 查看结果。如果该结果超出了整数(16 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

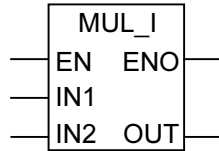
实例



如果 I0.0 = “1”，则激活 SUB\_I 框。MW0 - MW2 相减的结果输出到 MW10。如果结果超出整数允许范围，或者 I0.0 信号状态 = 0，则设置输出 Q4.0。

## 7.5 MUL\_I 整数乘

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	INT	I、Q、M、L、D 或常数	被乘数
IN2	INT	I、Q、M、L、D 或常数	第二个乘运算值
OUT	INT	I、Q、M、L、D	乘运算结果

描述

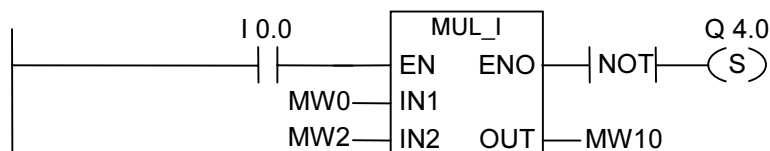
在启用(EN)输入端通过逻辑“1”激活 **MUL\_I** (整数乘)。IN1 和 IN2 相乘，结果通过 OUT 查看。如果该结果超出了整数(16 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

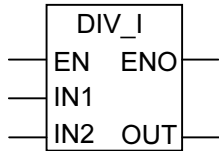
实例



如果 I0.0 = “1”，则激活 **MUL\_I** 框。MW0 x MW2 相乘的结果输出到 MD10。如果结果超出整数的允许范围，则设置输出 Q4.0。

## 7.6 DIV\_I 整数除

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	INT	I、Q、M、L、D 或常数	被除数
IN2	INT	I、Q、M、L、D 或常数	除数
OUT	INT	I、Q、M、L、D	除法结果

描述

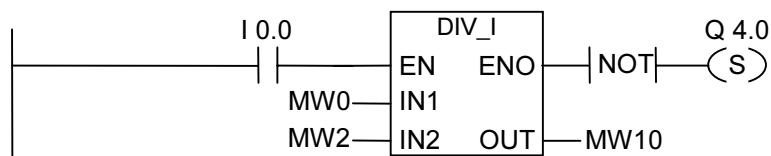
在启用(EN)输入端通过逻辑“1”激活 **DIV\_I** (整数除)。IN1 除以 IN2，结果可通过 OUT 查看。如果该结果超出了整数(16 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

实例

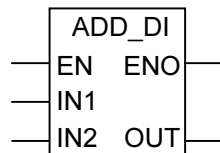


如果 I0.0 = “1”，则激活 DIV\_I 框。MW0 除以 MW2 的结果输出到 MW10。如果结果超出整数的允许范围，则设置输出 Q4.0。



## 7.7 ADD\_DI 长整数加

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DINT	I、Q、M、L、D 或常数	被加数
IN2	DINT	I、Q、M、L、D 或常数	加数
OUT	DINT	I、Q、M、L、D	加法结果

描述

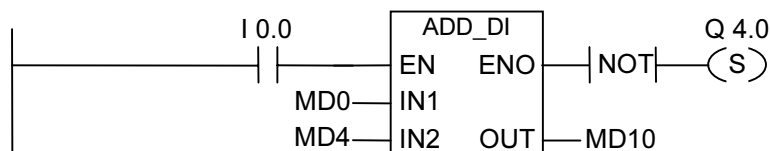
在启用(EN)输入端通过逻辑“1”激活 **ADD\_DI** (长整数加)。IN1 和 IN2 相加，其结果通过 OUT 来查看。如果该结果超出了长整数(32 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

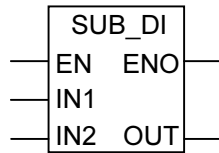
实例



如果 I0.0 = “1”，则激活 ADD\_DI 框。MD0 + MD4 相加的结果输出到 MD10。如果结果超出长整数的允许范围，则设置输出 Q4.0。

## 7.8 SUB\_DI 长整数减

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DINT	I、Q、M、L、D 或常数	被减数
IN2	DINT	I、Q、M、L、D 或常数	减数
OUT	DINT	I、Q、M、L、D	减法结果

描述

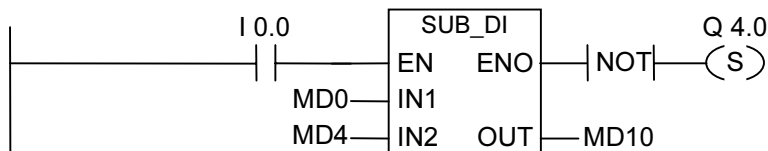
在启用(EN)输入端通过逻辑“1”激活 **SUB\_DI** (长整数减)。从 IN1 中减去 IN2，并通过 OUT 查看结果。如果该结果超出了长整数(32 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

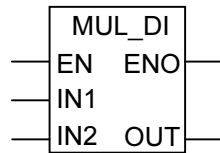
实例



如果 I0.0 = “1”，则激活 SUB\_DI 框。MD0 - MD4 相减的结果输出到 MD10。如果结果超出长整数的允许范围，则设置输出 Q4.0。

## 7.9 MUL\_DI 长整数乘

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DINT	I、Q、M、L、D 或常数	被乘数
IN2	DINT	I、Q、M、L、D 或常数	第二个乘运算值
OUT	DINT	I、Q、M、L、D	乘运算结果

描述

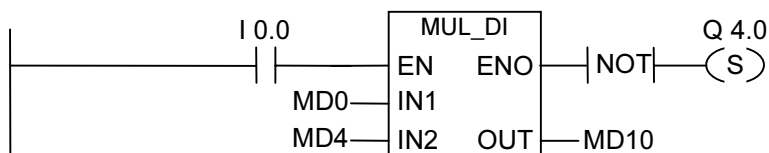
在启用(EN)输入端通过逻辑“1”激活 **MUL\_DI** (长整数乘)。IN1 和 IN2 相乘，结果通过 OUT 查看。如果该结果超出了长整数(32位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

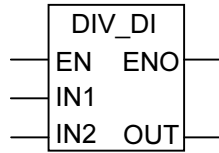
实例



如果 I0.0 = “1”，则激活 MUL\_DI 框。MD0 x MD4 相乘的结果输出到 MD10。如果结果超出长整数的允许范围，则设置输出 Q4.0。

## 7.10 DIV\_DI 长整数除

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DINT	I、Q、M、L、D 或常数	被除数
IN2	DINT	I、Q、M、L、D 或常数	除数
OUT	DINT	I、Q、M、L、D	除法的整数结果

描述

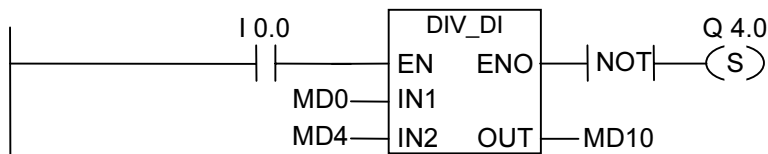
在启用(EN)输入端通过逻辑“1”激活 **DIV\_DI** (长整数除)。IN1 除以 IN2，结果可通过 OUT 查看。长整型除法不产生余数。如果该结果超出了长整数(32 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

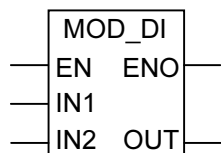
实例



如果 I0.0 = “1”，则激活 DIV\_DI 框。MD0 : MD4 的相除结果输出到 MD10。如果结果超出长整数的允许范围，则设置输出 Q4.0。

## 7.11 MOD\_DI 返回长整数余数

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DINT	I、Q、M、L、D 或常数	被除数
IN2	DINT	I、Q、M、L、D 或常数	除数
OUT	DINT	I、Q、M、L、D	除运算的余数

描述

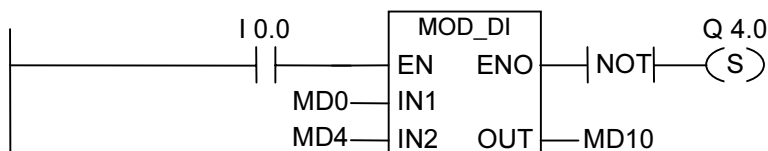
在启用(EN)输入端通过逻辑“1”激活 **MOD\_DI** (返回长整数余数)。IN1 除以 IN2，余数可通过 OUT 查看。如果该结果超出了长整数(32 位)允许的范围，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见使用整数算术指令时得出状态字的位数值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

实例



如果 I0.0 = “1”，则激活 **MOD\_DI** 框。MD0:MD4 相除的余数输出到 MD10。如果余数超出长整数的允许范围，则设置输出 Q4.0。



## 8 浮点型数学运算指令

### 8.1 浮点运算指令概述

#### 描述

IEEE 32 位浮点数属于称作实数(REAL)的数据类型。您可使用浮点运算指令通过两个 32 位 IEEE 浮点数来执行下列数学运算指令：

- ADD\_R 实数加
- SUB\_R 实数减
- MUL\_R 实数乘
- DIV\_R 实数除

利用浮点运算，可用一个 32 位 IEEE 浮点数执行下列运算：

- 求绝对值 (ABS)
- 求平方(SQR)和平方根 (SQRT)
- 求自然对数 (LN)
- 求指数值(EXP)以 e (= 2,71828)为底
- 求下列 32 位 IEEE 浮点数表示的角度的三角函数
  - 正弦(SIN)和反正弦(ASIN)
  - 余弦(COS)和反余弦(ACOS)
  - 正切(TAN)和反正切(ATAN)

参见判断状态字的位。

## 8.2 使用浮点运算指令时得出状态字的位数值

### 描述

浮点指令影响状态字中的下列位：CC 1 和 CC 0、OV 和 OS。

下表说明了指令结果为浮点数(32 位)时状态字中各位的信号状态：

结果的有效范围	CC 1	CC 0	OV	OS
+0, -0(零)	0	0	0	*
$-3.402823E+38 < \text{结果} < -1.175494E-38$ (负数)	0	1	0	*
$+1.175494E-38 < \text{结果} < 3.402824E+38$ (正数)	1	0	0	*

\* 指令结果不影响 OS 位。

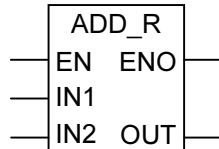
结果的无效范围	CC 1	CC 0	OV	OS
下溢 $-1.175494E-38 < \text{结果} < -1.401298E-45$ (负数)	0	0	1	1
下溢 $+1.401298E-45 < \text{结果} < +1.175494E-38$ (正数)	0	0	1	1
溢出 结果 $< -3.402823E+38$ (负数)	0	1	1	1
溢出 结果 $> 3.402823E+38$ (正数)	1	0	1	1
无效浮点数或非法指令 (输入值不在有效范围内)	1	1	1	1



## 8.3 基本指令

### 8.3.1 ADD\_R 实数加

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	REAL	I、Q、M、L、D 或常数	被加数
IN2	REAL	I、Q、M、L、D 或常数	加数
OUT	REAL	I、Q、M、L、D	加法结果

描述

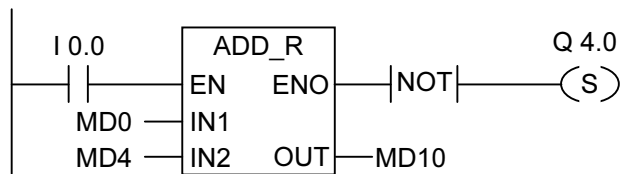
在启用(EN)输入端通过一个逻辑“1”来激活 **ADD\_R** (实数加)。IN1 和 IN2 相加，其结果通过 OUT 来查看。如果结果超出了浮点数允许的范围(溢出或下溢)，OV 位和 OS 位将为“1”并且 ENO 为“0”，这样便不执行此数学框后由 ENO 连接的其它功能(层叠排列)。

参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

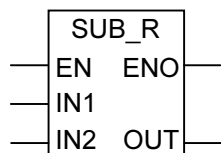
实例



由 I0.0 处的逻辑“1”激活 ADD\_R 框。MD0 + MD4 相加的结果输出到 MD10。如果结果超出了浮点数的允许范围，或者如果没有处理该程序语句(I0.0 = 0)，则设置输出 Q4.0。

### 8.3.2 SUB\_R 实数减

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	REAL	I、Q、M、L、D 或常数	被减数
IN2	REAL	I、Q、M、L、D 或常数	减数
OUT	REAL	I、Q、M、L、D	减法结果

描述

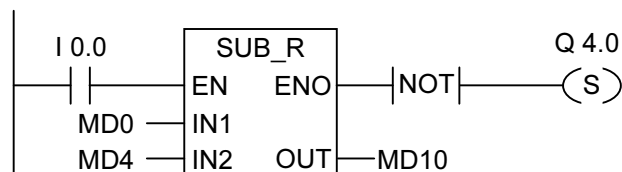
在启用(EN)输入端通过一个逻辑“1”来激活 SUB\_R (实数减)。从 IN1 中减去 IN2，并通过 OUT 查看结果。如果该结果超出了浮点数允许的范围(溢出或下溢)，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

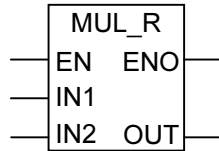
实例



在 I0.0 处由逻辑“1”激活 SUB\_R 框。MD0 - MD4 相减的结果输出到 MD10。如果结果超出了浮点数的允许范围，或者如果没有处理该程序语句，则设置输出 Q4.0。

### 8.3.3 MUL\_R 实数乘

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	REAL	I、Q、M、L、D 或常数	被乘数
IN2	REAL	I、Q、M、L、D 或常数	第二个乘运算值
OUT	REAL	I、Q、M、L、D	乘运算结果

描述

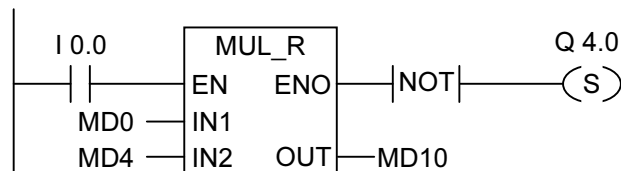
在启用(EN)输入端通过一个逻辑“1”来激活 **MUL\_R** (实数乘)。IN1 和 IN2 相乘，结果通过 OUT 查看。如果该结果超出了浮点数允许的范围(溢出或下溢)，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

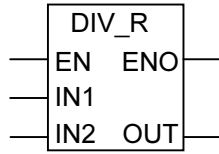
实例



在 I0.0 处由逻辑“1”激活 **MUL\_R** 框。MD0 x MD4 相乘的结果输出到 MD0。如果结果超出了浮点数的允许范围，或者如果没有处理该程序语句，则设置输出 Q4.0。

### 8.3.4 DIV\_R 实数除

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	REAL	I、Q、M、L、D 或常数	被除数
IN2	REAL	I、Q、M、L、D 或常数	除数
OUT	REAL	I、Q、M、L、D	除法结果

描述

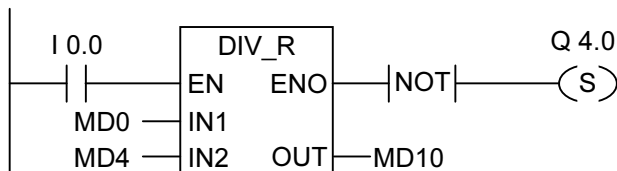
在启用(EN)输入端通过一个逻辑“1”来激活 DIV\_R (实数除)。IN1 除以 IN2，结果可通过 OUT 查看。如果该结果超出了浮点数允许的范围(溢出或下溢)，OV 位和 OS 位将为“1”并且 ENO 为逻辑“0”，这样便不执行此数学框后由 ENO 连接的其它函数(层叠排列)。

参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

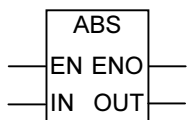
实例



由 I0.0 处的逻辑“1”激活 DIV\_R 框。MD0 除以 MD4 的结果输出到 MD10。如果结果超出了浮点数的允许范围，或者如果没有处理该程序语句，则设置输出 Q4.0。

### 8.3.5 ABS 求浮点数的绝对值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的绝对值

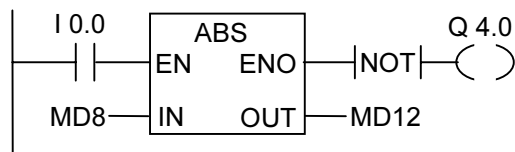
描述

**ABS** 求浮点数的绝对值。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

实例



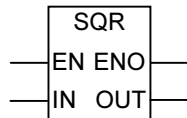
如果 I0.0 = “1”，则 MD8 的绝对值在 MD12 输出。

MD8 = + 6.234 得到 MD12 = 6.234。如果未执行该转换(ENO = EN = 0)，则输出 Q4.0 为 “1”。

## 8.4 扩展指令

### 8.4.1 SQR 求平方

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的平方

描述

**SQR** 求浮点数的平方

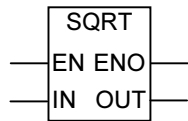
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

## 8.4.2 SQRT 求平方根

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的平方根

描述

**SQRT** 求浮点数的平方根。当地址大于“0”时，此指令得出一个正的结果。唯一例外的是：-0 的平方根是 -0。

参见判断状态字的位。

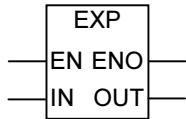
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1



### 8.4.3 EXP 求指数值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的指数值

描述

**EXP** 求浮点数的以  $e(=2,71828\dots)$  为底的指数值。

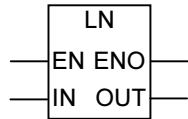
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

### 8.4.4 LN 求自然对数

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的自然对数

描述

**LN** 求浮点数的自然对数。

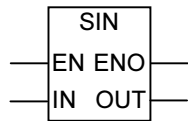
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

### 8.4.5 SIN 求正弦值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的正弦

描述

**SIN** 求浮点数的正弦值。这里浮点数代表以弧度为单位的一个角度。

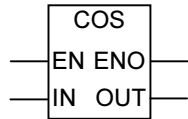
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

### 8.4.6 COS 求余弦值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的余弦

描述

**COS** 求浮点数的余弦值。这里浮点数代表以弧度为单位的一个角度。

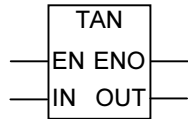
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

### 8.4.7 TAN 求正切值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的正切

描述

**TAN** 求浮点数的正切值。这里浮点数代表以弧度为单位的一个角度。

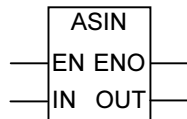
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

### 8.4.8 ASIN 求反正弦值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的反正弦

描述

**ASIN** 求一个定义在  $-1 \leq \text{输入值} \leq 1$  范围内的浮点数的反正弦值。结果代表下式范围内的一个以弧度为单位的角度

$$-\pi/2 \leq \text{output value} \leq +\pi/2$$

where  $\pi = 3.1415\dots$

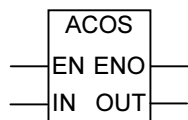
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

### 8.4.9 ACOS 求反余弦值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点 或常数
OUT	REAL	I、Q、M、L、D	输出值：浮点数的反余弦

描述

**ACOS** 求一个定义在  $-1 \leq \text{输入值} \leq 1$  范围内的浮点数的反余弦值。结果代表下式范围内的一个以弧度为单位的角度

$$0 \leq \text{output value} \leq +\pi$$

where  $\pi = 3.1415\dots$

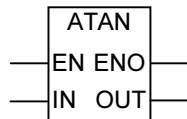
参见判断状态字的位。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1

### 8.4.10 ATAN 求反正切值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	REAL	I、Q、M、L、D 或常数	输入值：浮点
OUT	REAL	I、Q、M、L、D	输出值：浮点数的反正切

描述

**ATAN** 求浮点数的反正切值。结果代表下式范围内的一个以弧度为单位的角度

$$-\pi/2 \leq \text{output value} \leq +\pi/2$$

where  $\pi = 3.1415\dots$

参见判断状态字的位。

状态字

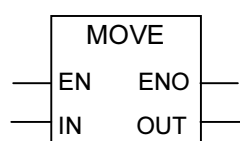
	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	x	0	x	x	1



## 9 传送指令

### 9.1 MOVE 分配值

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	所有长度为 8、16 或 32 位的基本数据类型	I、Q、M、L、D 或常数	源值
OUT	所有长度为 8、16 或 32 位的基本数据类型	I、Q、M、L、D	目标地址

描述

**MOVE** (分配值)通过启用 **EN** 输入来激活。在 **IN** 输入端指定的值将复制到在 **OUT** 输出端指定的地址。**ENO** 与 **EN** 的逻辑状态相同。**MOVE** 只能复制 **BYTE**、**WORD** 或 **DWORD** 数据对象。用户自定义数据类型(如数组或结构)必须使用系统功能“**BLKMOVE**” (SFC 20)来复制。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	-	-	-	-	0	1	1	1

#### MCR (主站控制继电器)依存

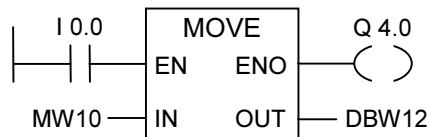
只有当“传送”框位于激活的 **MCR** 区内时，才会激活 **MCR** 依存。在激活的 **MCR** 区内，如果开启了 **MCR**，同时有通往启用输入的电，则按如上所述复制寻址的数据。如果 **MCR** 关闭，并执行了 **MOVE**，则无论当前 **IN** 状态如何，均会将逻辑“0”写入到指定的 **OUT** 地址。

注意

将某个值传送给另一长度的数据类型时，将根据需要截断或以零填充高位字节：

<b>实例：双字</b>	<b>1111 1111</b>	<b>0000 1111</b>	<b>1111 0000</b>	<b>0101 0101</b>
<b>传送</b>	<b>结果</b>			
到双字：	1111 1111	0000 1111	1111 0000	0101 0101
到字节：				0101 0101
到字：			1111 0000	0101 0101
<b>实例：字节</b>				<b>1111 0000</b>
<b>传送</b>	<b>结果</b>			
到字节：				1111 0000
到字：			0000 0000	1111 0000
到双字：	0000 0000	0000 0000	0000 0000	1111 0000

实例



如果 I0.0 为“1”，则执行指令。把 MW10 的内容复制到当前打开 DB 的数据字 12。

如果执行了指令，则 Q4.0 为“1”。

如果实例梯级在激活的 MCR 区之内：

- 如果 MCR 开启，则按如上所述将 MW10 数据复制到 DBW12。
- 如果 MCR 关闭，则将“0”写入到 DBW12。

## 10 程序控制指令

### 10.1 程序控制指令概述

#### 描述

可使用下列程序控制指令：

- ---(CALL) 调用来自线圈的 FC SFC (不带参数)
- CALL\_FB 调用来自框的 FB
- CALL\_FC 调用来自框的 FC
- CALL\_SFB 调用来自框的系统 FB
- CALL\_SFC 调用来自框的系统 FC
- 调用多重实例
- 从库中调用块
  
- 使用 MCR 功能的重要注意事项
- ---(MCR<) 主控制继电器打开
- ---(MCR>) 主控制继电器关闭
- ---(MCRA) 主控制继电器激活
- ---(MCRD) 主控制继电器取消激活
  
- RET 返回

## 10.2 ---(Call) 调用来自线圈的 FC SFC (不带参数)

### 符号

<FC/SFC 编号>

---(CALL)

参数	数据类型	存储区	描述
<FC/SFC 编号>	BLOCK_FC BLOCK_SFC	-	FC/SFC 编号, 范围取决于 CPU

### 描述

---(Call) (不带参数调用 FC 或 SFC)用于调用没有传递参数的功能(FC)或系统功能(SFC)。只有在 CALL 线圈上 RLO 为“1”时,才执行调用。当执行---(Call)时,

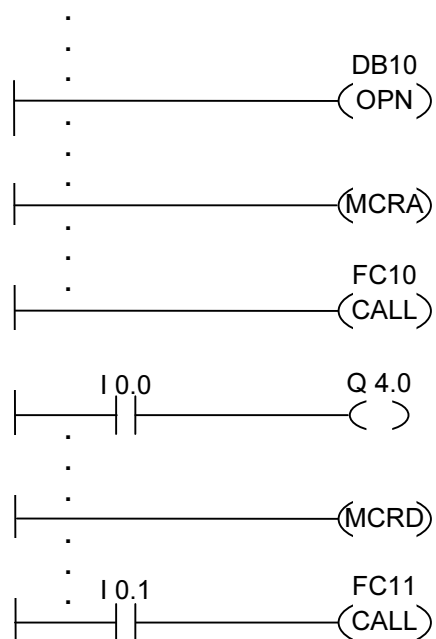
- 存储调用块的返回地址,
- 由当前的本地数据区代替以前的本地数据区,
- 将 MA 位(有效 MCR 位)移位到 B 堆栈中,
- 为被调用的功能创建一个新的本地数据区。

之后,在被调用的 FC 或 SFC 中继续进行程序处理。

### 状态字

		BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
无条件:	写:	-	-	-	-	0	0	1	-	0
有条件:	写:	-	-	-	-	0	0	1	1	0

## 实例



上面所示的梯形图的梯级是由用户编写的功能块中的程序段。在该 **FB** 中，打开 **DB10**，并激活 **MCR** 功能。当执行无条件调用 **FC10** 时，发生下面情况：

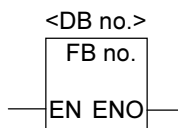
保存调用 **FB** 的返回地址，并保存 **DB10** 和调用 **FB** 的实例数据块的选择数据。在 **MCRA** 指令中设置成“1”的 **MA** 位被推入到 **B** 堆栈中，然后为被调用块(**FC10**)将 **MA** 位设置成“0”。继续在 **FC10** 中进行程序处理。当 **FC10** 要求 **MCR** 功能时，必须在 **FC10** 内重新激活该功能。当完成 **FC10** 时，程序处理返回调用 **FB**。不管 **FC10** 使用了哪个 **DB**，都恢复 **MA** 位，**DB10** 和用户编写 **FB** 的实例数据块重新成为当前 **DB**。通过将 **I0.0** 的逻辑状态分配给输出 **Q4.0**，程序继续处理下一个梯级。**FC11** 的调用为条件调用。只有在 **I0.1** 为“1”时，才执行调用。执行调用后，将程序控制传递给 **FC11** 并从 **FC11** 返回程序控制的过程，与已经描述过的 **FC10** 的过程相同。

**注意**

返回调用块后，不是总会再次打开以前打开的 **DB**。请一定要阅读自述文件中的注意事项。

## 10.3 CALL\_FB 调用来自框的 FB

### 符号



该符号取决于 FB (是否带参数以及带多少个参数)。它必须具有 EN、ENO，以及 FB 的名称或编号。

参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
FB 编号	BLOCK_FB	-	FB/DB 编号，范围取决于 CPU
DB 号	BLOCK_DB	-	

### 描述

当 EN 为“1”时，执行 **CALL\_FB** (调用来自框的功能块)。当执行 **CALL\_FB** 时，

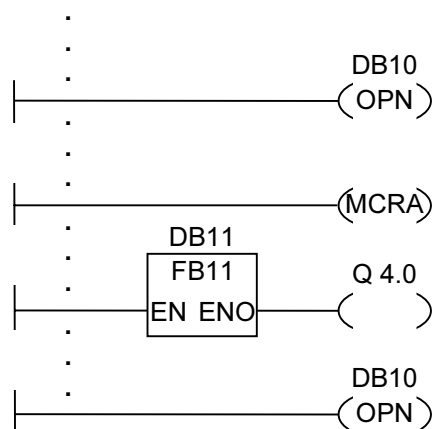
- 存储调用块的返回地址，
- 存储两个当前数据块(DB 和实例 DB)的选择数据，
- 由当前的本地数据区代替以前的本地数据区，
- 将 MA 位(有效 MCR 位)移位到 B 堆栈中，
- 为被调用的功能块创建一个新的本地数据区。

之后，在被调用的功能块中继续进行程序处理。扫描 BR 位，以查找 ENO。用户必须使用---(SAVE)为调用块中的 BR 位分配所需的状态(错误评估)。

### 状态字

		BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
无条件:	写:	x	-	-	-	0	0	x	x	x
有条件:	写:	-	-	-	-	0	0	x	x	x

## 实例



上面所示的梯形图的梯级是由用户编写的功能块中的程序段。在该 **FB** 中，打开 **DB10**，并激活 **MCR** 功能。当执行无条件调用 **FB11** 时，发生下面情况：

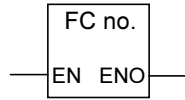
保存调用 **FB** 的返回地址，并保存 **DB10** 和调用 **FB** 的实例数据块的选择数据。在 **MCRA** 指令中设置成“1”的 **MA** 位被推入到 **B** 堆栈中，然后为被调用块(**FB11**)将 **MA** 位设置成“0”。继续在 **FB11** 中进行程序处理。当 **FB11** 要求 **MCR** 功能时，必须在 **FB11** 内重新激活该功能。必须使用指令---(**SAVE**)在 **BR** 位中保存 **RLO** 的状态，以便判断调用 **FB** 中的错误。当完成 **FB11** 时，程序处理返回调用 **FB**。恢复 **MA** 位，并重新打开用户编写的 **FB** 的实例数据块。当正确处理 **FB11** 时，**ENO** = “1”，因此 **Q4.0** = “1”。

**注意**

打开 **FB** 或 **SFB** 时，会丢失以前打开的 **DB** 的编号。必须重新打开所要求的 **DB**。

## 10.4 CALL\_FC 调用来自框的 FC

### 符号



该符号取决于 FC (是否带参数以及带多少个参数)。它必须具有 EN、ENO，以及 FC 的名称或编号。

参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
FC 编号	BLOCK_FC	-	FCC 编号，范围取决于 CPU

### 描述

**CALL\_FC** (调用来自框的功能)用于调用一个功能(FC)。当 EN 为“1”时，执行调用。如果执行了 CALL\_FC，

- 存储调用块的返回地址，
- 由当前的本地数据区代替以前的本地数据区，
- 将 MA 位(有效 MCR 位)移位到 B 堆栈中，
- 为被调用的功能创建一个新的本地数据区。

之后，在被调用的功能中继续进行程序处理。

扫描 BR 位，以查找 ENO。用户必须使用---(SAVE)为调用块中的 BR 位分配所需的状态(错误评估)。

当调用一个功能且被调用块的变量声明表中含有 IN、OUT 和 IN\_OUT 声明时，这些变量以形式参数列表添加到调用块的程序中。

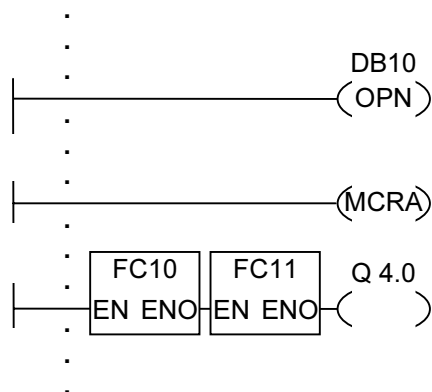
当调用功能时，**必须**在调用位置处将实际参数分配给形式参数。功能声明中的任何初始值都没有意义。

### 状态字

		BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
无条件:	写:	x	-	-	-	0	0	x	x	x
有条件:	写:	-	-	-	-	0	0	x	x	x



## 实例



上面所示的梯形图的梯级是由用户编写的功能块中的程序段。在该 FB 中，打开 DB10，并激活 MCR 功能。当执行无条件调用 FC10 时，发生下面情况：

保存调用 FB 的返回地址，并保存 DB10 和调用 FB 的实例数据块的选择数据。在 MCRA 指令中设置成“1”的 MA 位被推入到 B 堆栈中，然后为被调用块(FC10)将 MA 位设置成“0”。继续在 FC10 中进行程序处理。当 FC10 要求 MCR 功能时，必须在 FC10 内重新激活该功能。必须使用指令---(SAVE)在 BR 位中保存 RLO 的状态，以便判断调用 FB 中的错误。当完成 FC10 时，程序处理返回调用 FB。恢复 MA 位。执行 FC10 后，根据 ENO，在调用 FB 中继续进行程序处理：

ENO = “1”            FC11 已处理

ENO = “0”            在下一个程序段中开始处理

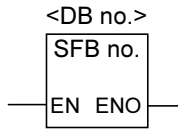
如果也正确处理了 FC11，则 ENO = “1”，因此 Q4.0 = “1”。

**注意**

返回调用块后，不是总会再次打开以前打开的 DB。请一定要阅读自述文件中的注意事项。

## 10.5 CALL\_SFB 调用来自框的系统 FB

### 符号



该符号取决于 SFB (是否带参数以及带多少个参数)。它必须具有 EN、ENO，以及 SFB 的名称或编号。

参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
SFB 编号	BLOCK_SFB	-	SFB 编号，范围取决于 CPU
DB 号	BLOCK_DB	-	

### 描述

当 EN 为“1”时，执行 CALL\_SFB (调用来自框的系统功能块)。执行 CALL\_SFB 时，

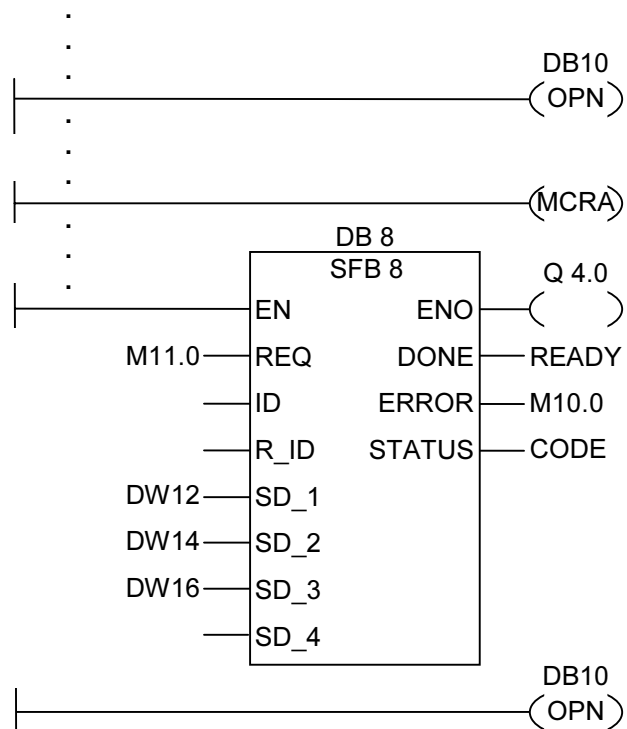
- 存储调用块的返回地址，
- 存储两个当前数据块(DB 和实例 DB)的选择数据，
- 由当前的本地数据区代替以前的本地数据区，
- 将 MA 位(有效 MCR 位)移位到 B 堆栈中，
- 为被调用的系统功能块创建一个新的本地数据区。

然后在被调用的 SFB 中继续进行程序处理。当调用 SFB (EN = “1”)且没有发生错误时，ENO 为“1”。

### 状态字

		BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
无条件:	写:	x	-	-	-	0	0	x	x	x
有条件:	写:	-	-	-	-	0	0	x	x	x

## 实例



上面所示的梯形图的梯级是由用户编写的功能块中的程序段。在该 **FB** 中，打开 **DB10**，并激活 **MCR** 功能。当执行无条件调用 **SFB8** 时，发生下面情况：

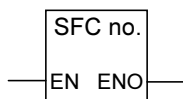
保存调用 **FB** 的返回地址，并保存 **DB10** 和调用 **FB** 的实例数据块的选择数据。在 **MCRA** 指令中设置成“1”的 **MA** 位被推入到 **B** 堆栈中，然后为被调用块(**SFB8**)将 **MA** 位设置成“0”。继续在 **SFB8** 中进行程序处理。当完成 **SFB8** 时，程序处理返回调用 **FB**。恢复 **MA** 位，且用户编写的 **FB** 的实例数据块成为当前实例 **DB**。当正确处理 **SFB8** 时，**ENO** = “1”，因此 **Q4.0** = “1”。

**注意**

打开 **FB** 或 **SFB** 时，会丢失以前打开的 **DB** 的编号。必须重新打开所要求的 **DB**。

## 10.6 CALL\_SFC 调用来自框的系统 FC

### 符号



该符号取决于 SFC (是否带参数以及带多少个参数)。它必须具有 EN、ENO, 以及 SFC 的名称或编号。

参数	数据类型	存储区	描述
EN	BOOL	-	启用输入
ENO	BOOL	-	启用输出
SFC 编号	BLOCK_SFC	-	SFC 编号; 范围取决于 CPU

### 描述

**CALL\_SFC** (调用来自框的系统功能)用于调用一个 SFC。当 EN 为“1”时, 执行调用。当执行 CALL\_SFC 时,

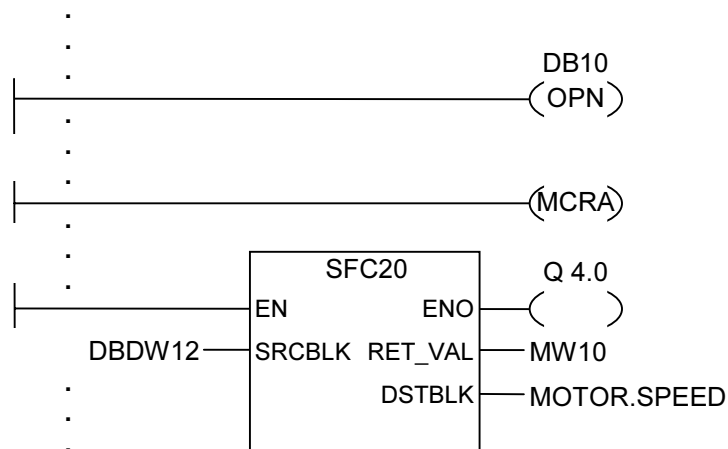
- 存储调用块的返回地址,
- 由当前的本地数据区代替以前的本地数据区,
- 将 MA 位(有效 MCR 位)移位到 B 堆栈中,
- 为被调用的系统功能创建一个新的本地数据区。

之后, 在被调用的 SFC 中继续进行程序处理。当调用 SFC (EN = “1”)且没有发生错误时, ENO 为“1”。

### 状态字

		BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
无条件:	写:	x	-	-	-	0	0	x	x	x
有条件:	写:	-	-	-	-	0	0	x	x	x

## 实例



上面所示的梯形图的梯级是由用户编写的功能块中的程序段。在该 FB 中，打开 DB10，并激活 MCR 功能。当执行无条件调用 SFC20 时，发生下面情况：

保存调用 FB 的返回地址，并保存 DB10 和调用 FB 的实例数据块的选择数据。在 MCRA 指令中设置成“1”的 MA 位被推入到 B 堆栈中，然后为被调用块(SFC20)将 MA 位设置成“0”。继续在 SFC20 中进行程序处理。当完成 SFC20 时，程序处理返回调用 FB。恢复 MA 位。

处理 SFC20 后，根据 ENO，在调用 FB 中继续进行程序处理：

ENO = “1”          Q4.0 = “1”

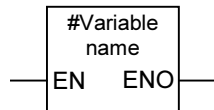
ENO = “0”          Q4.0 = “0”

**注意**

返回调用块后，不是总会再次打开以前打开的 DB。请一定要阅读自述文件中的注意事项。

## 10.7 调用多重实例

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
# 变量名	FB、SFB	-	多重实例的名称

描述

通过声明一个具有功能块数据类型的静态变量，创建一个多重实例。只在程序元素目录中包括已经声明的多重实例。多重实例的符号改变取决于是否带参数以及带多少个参数。始终显示 EN、ENO 和变量名。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	0	0	x	x	x

## 10.8 从库中调用块

在此，可使用 SIMATIC 管理器中可供使用的库来选择下列块：

- 集成在 CPU 操作系统中的块(对于 V3 版本的 STEP 7 项目，为“标准库”，对于 V2 版本的 STEP 7 项目，为“stdlibs (V2)”)
- 由于要多次使用而自行在库中保存的块。

## 10.9 使用 MCR 功能的重要注意事项



**注意在其中使用 MCRA 激活主控制继电器的块：**

- 去活 MCR 时，为---(MCR<)和---(MCR>)之间的程序段中的所有赋值写入数值 0。这对包含一个赋值的所有框都有效，包括传递到块的参数在内。
- 当 MCR<指令前的 RLO = 0 时，取消激活 MCR。



**危险：PLC 处于 STOP 状态或未定义的运行特征！**

编译器还对 VAR\_TEMP 中定义的临时变量后的本地数据进行写访问，以便计算地址。这表示下列命令序列将把 PLC 设置成 STOP，或导致未定义的运行系统特征：

**形式参数访问**

- 访问 STRUCT、UDT、ARRAY、STRING 类型的复杂 FC 参数的组件
- 访问来自具有多重实例能力的块(V2 版本的块)的 IN\_OUT 区域的、STRUCT、UDT、ARRAY、STRING 类型的复杂 FB 参数的组件。
- 当地址高于 8180.0 时，访问具有多重实例能力(V2 版本的块)的功能块的参数。
- 在具有多重实例能力(V2 版本的块)的功能块访问类型为 BLOCK\_DB 的参数，打开 DB0。任何后继数据存取都会将 CPU 设置成 STOP 状态。T 0、C 0、FC0 或 FB0 也经常用于 TIMER、COUNTER、BLOCK\_FC 和 LOCK\_FB。

**参数传递**

- 调用可传递参数的功能。

**LAD/FBD**

- 梯形图或功能块中的 T 分支和中线输出以 RLO = 0 开始。

**纠正方法**

从上述命令对 MCR 的依赖性，来解决问题：

- 在所述语句或程序段之前，使用**主控制继电器取消激活**指令，取消激活主控制继电器。
- 在所述语句或程序段之后，使用**主控制继电器激活**指令，重新激活主控制继电器。

## 10.10 ---(MCR<) 主控制继电器打开

使用 MCR 功能的重要注意事项

### 符号

---(MCR<)

### 描述

---(MCR<) (打开主控制继电器区域)在 MCR 堆栈中保存 RLO。MCR 嵌套堆栈为 LIFO (后入先出)堆栈，且只能有 8 个堆栈条目(嵌套级别)。当堆栈已满时，---(MCR<)功能产生一个 MCR 堆栈故障(MCRF)。下列元素与 MCR 有关，并在打开 MCR 区域时，受保存在 MCR 堆栈中的 RLO 状态的影响：

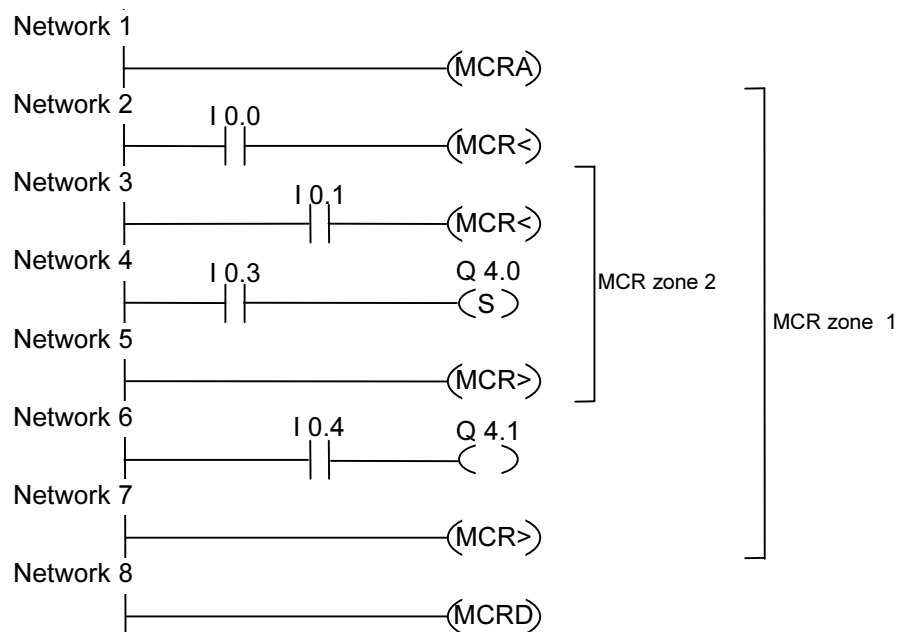
- --( # )        中线输出
- --( )         输出
- --( S )       置位输出
- --( R )       复位输出
- RS            复位触发器
- SR            置位触发器
- MOVE         分配值

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	1	-	0



## 实例



MCRA 梯级激活 MCR 功能。然后可以创建至多 8 个嵌套 MCR 区域。在此实例中，有两个 MCR 区域。按如下执行该功能：

I0.0 = “1” (区域 1 的 MCR 打开)：将 I0.4 的逻辑状态分配给 Q4.1

I0.0 = “0” (区域 1 的 MCR 关闭)：无论输入 I0.4 的逻辑状态如何，Q4.1 都为 0。

I0.1 = “1” (区域 2 的 MCR 打开)：当 I0.3 为 “1” 时，将 Q4.0 设置成 “1”

I0.1 = “0” (区域 2 的 MCR 关闭)：无论 I0.3 的逻辑状态如何，Q4.0 都保持不变，

## 10.11 ---(MCR>) 主控制继电器关闭

使用 MCR 功能的重要注意事项

### 符号

---(MCR>)

### 描述

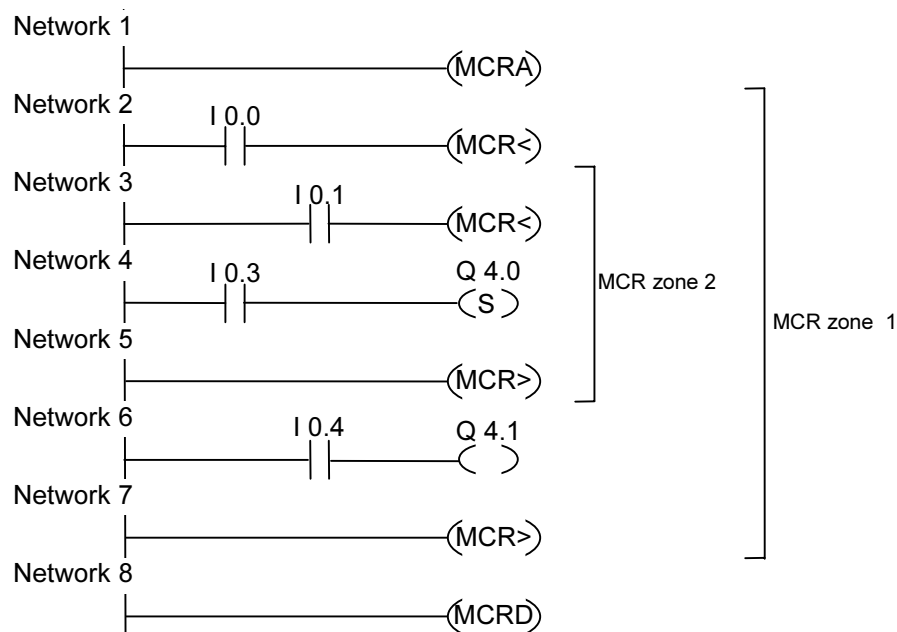
---(MCR>) (关闭最后打开的 MCR 区域)从 MCR 堆栈中删除一个 RLO 条目。MCR 嵌套堆栈为 LIFO (后入先出)堆栈，且只能有 8 个堆栈条目(嵌套级别)。当堆栈已空时，---(MCR>)产生一个 MCR 堆栈故障(MCRF)。下列元素与 MCR 有关，并在打开 MCR 区域时，受保存在 MCR 堆栈中的 RLO 状态的影响：

- --( # )        中线输出
- --( )         输出
- --( S )       置位输出
- --( R )       复位输出
- RS            复位触发器
- SR            置位触发器
- MOVE         分配值

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	1	-	0

## 实例



---(MCRA)梯级激活 MCR 功能。然后可以创建至多 8 个嵌套 MCR 区域。在此实例中，有两个 MCR 区域。第一个---(MCR>) (MCR 关闭)梯级属于第二个---(MCR<) (MCR 打开)梯级。两者之间的所有梯级都属于 MCR 区域 2。按如下执行这些功能：

I0.0 = “1”：将 I0.4 的逻辑状态分配给 Q4.1

I0.0 = “0”：无论输入 I0.4 的逻辑状态如何，Q4.1 都为 0。

I0.1 = “1”：当 I0.3 为 “1” 时，将 Q4.0 设置成 “1”

I0.1 = “0”：无论 I0.3 的逻辑状态如何，Q4.0 都保持不变。

## 10.12 ---(MCRA) 主控制继电器激活

使用 MCR 功能的重要注意事项

### 符号

---(MCRA)

### 描述

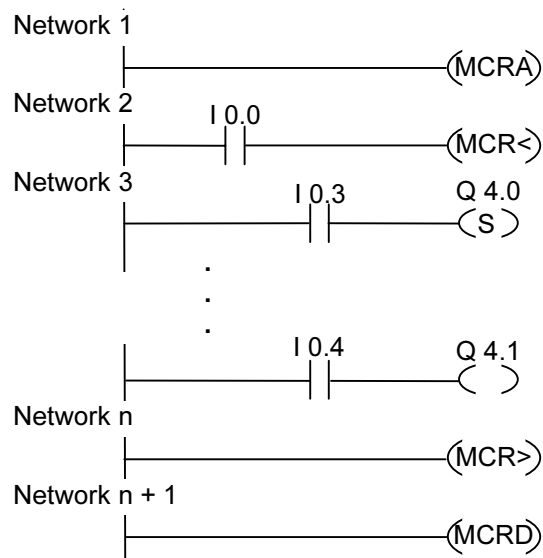
---(MCRA) (激活主控制继电器)激活主控制继电器功能。在该命令后，可以使用下列命令编程 MCR 区域：

- ---(MCR<)
- ---(MCR>)

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	-	-	-	-

### 实例



MCRA 梯级激活 MCR 功能。MCR<和 MCR> (输出 Q4.0、Q4.1)之间的梯级按如下执行：

**I0.0 = “1” (MCR 打开)：**当 I0.3 为逻辑“1”时，将 Q4.0 设置成“1”，或当 I0.3 为“0”时，Q4.0 保持不变，并将 I0.4 的逻辑状态分配给 Q4.1

**I0.0 = “0” (MCR 关闭)：**无论 I0.3 的逻辑状态如何，Q4.0 保持不变，无论 I0.4 的逻辑状态如何，Q4.1 为“0”

在下一个梯级中，指令---(MCRD)取消激活 MCR。这表示不能再使用指令对---(MCR<)和---(MCR>)编程更多的 MCR 区域。

## 10.13 ---(MCRD) 主控制继电器取消激活

使用 MCR 功能的重要注意事项

符号

---(MCRD)

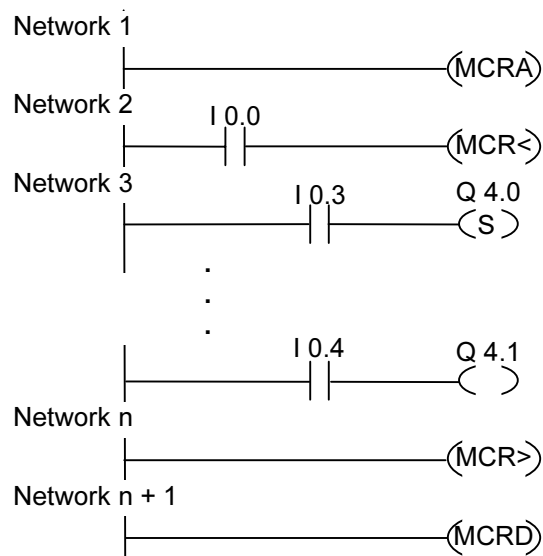
描述

---(MCRD) (取消激活主控制继电器)取消激活 MCR 功能。在该命令后，不能编程 MCR 区域。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	-	-	-	-

实例



MCRA 梯级激活 MCR 功能。MCR<和 MCR> (输出 Q4.0、Q4.1)之间的梯级按如下执行：

I0.0 = “1” (MCR 打开)：当 I0.3 为逻辑“1”时，将 Q4.0 设置成“1”，并将 I0.4 的逻辑状态分配给 Q4.1。

I0.0 = “0” (MCR 关闭)：无论 I0.3 的逻辑状态如何，Q4.0 保持不变，无论 I0.4 的逻辑状态如何，Q4.1 为“0”。

在下一个梯级中，指令---(MCRD)取消激活 MCR。这表示不能再使用指令对---(MCR<)和---(MCR>)编程更多的 MCR 区域。

## 10.14 ---(RET) 返回

符号

---(RET)

描述

**RET** (返回)用于有条件地退出块。对于该输出，要求在前面使用一个逻辑运算。

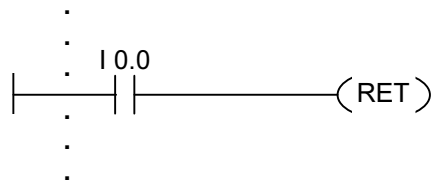
状态字

条件返回(当 RLO = “1” 时，返回):

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	*	-	-	-	0	0	1	1	0

\* 操作 **RET** 在内部以 “SAVE; BEC,” 序列显示。这还影响 BR 位。

实例



当 I0.0 为 “1” 时，退出块。

# 11 移位和循环指令

## 11.1 移位指令

### 11.1.1 移位指令概述

#### 描述

可使用移位指令向左或向右逐位移动输入 IN 的内容(另请参阅 CPU 寄存器)。向左移  $n$  位会将输入 IN 的内容乘以 2 的  $n$  次幂( $2^n$ )；向右移  $n$  位则会将输入 IN 的内容除以 2 的  $n$  次幂( $2^n$ )。例如，如果将十进制值 3 的等效二进制数向左移 3 位，则在累加器中将得到十进制值 24 的等效二进制数。如果将十进制值 16 的等效二进制数向右移 2 位，则在累加器中将得到十进制值 4 的等效二进制数。

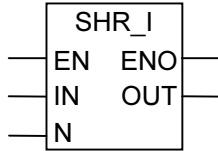
您为输入参数 N 提供的数值指示要移动的位数。由零或符号位的信号状态(0 代表正数、1 代表负数)填充移位指令空出的位。最后移动的位的信号状态会被载入状态字的 CC 1 位中。复位状态字的 CC 0 和 OV 位为 0。可使用跳转指令来判断 CC 1 位。

下列移位指令可用：

- SHR\_I 整数右移
- SHR\_DI 长整数右移
- SHL\_W 字左移
- SHR\_W 字右移
- SHL\_DW 双字左移
- SHR\_DW 双字右移

### 11.1.2 SHR\_I 整数右移

符号

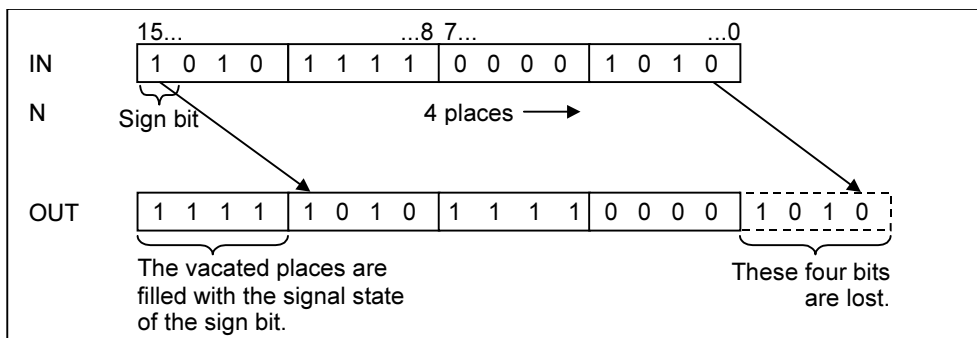


参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	INT	I、Q、M、L、D	要移位的值
N	WORD	I、Q、M、L、D	要移动的位数
OUT	INT	I、Q、M、L、D	移位指令的结果

描述

**SHR\_I** (整数右移)指令通过使能(EN)输入位置上的逻辑“1”来激活。**SHR\_I**指令用于将输入 IN 的 0 至 15 位逐位向右移动。16 到 31 位不受影响。输入 N 用于指定移位的位数。如果 N 大于 16，命令将按照 N 等于 16 的情况执行。自左移入的、用于填补空出位的位位置将被赋予位 15 的逻辑状态(整数的符号位)。这意味着，当该整数为正时，这些位将被赋值“0”，而当该整数为负时，则被赋值为“1”。可在输出 OUT 位置扫描移位指令的结果。如果 N 不等于 0，则 **SHR\_I** 会将 CC 0 位和 OV 位设为“0”。

ENO 与 EN 具有相同的信号状态。

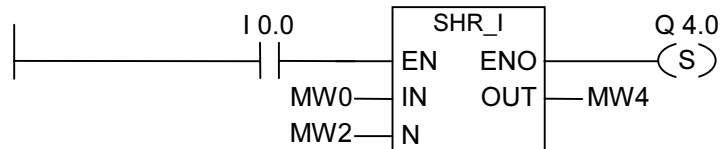




## 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

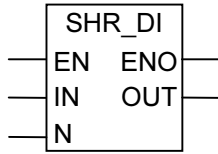
## 实例



SHR\_I 框由 I0.0 位置上的逻辑“1”激活。装载 MW0 并将其右移由 MW2 指定的位数。结果将被写入 MW4。置位 Q4.0。

### 11.1.3 SHR\_DI 长整数右移

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DINT	I、Q、M、L、D	要移位的值
N	WORD	I、Q、M、L、D	要移动的位数
OUT	DINT	I、Q、M、L、D	移位指令的结果

描述

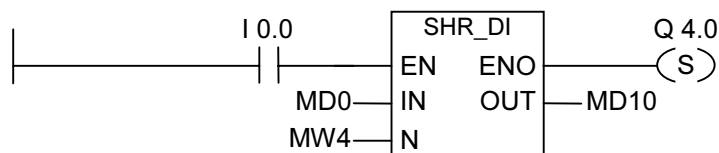
**SHR\_DI** (右移长整数)指令通过使能(EN)输入位置上的逻辑“1”来激活。**SHR\_DI**指令用于将输入 **IN** 的 0 至 31 位逐位向右移动。输入 **N** 用于指定移位的位数。如果 **N** 大于 32，命令将按照 **N** 等于 32 的情况执行。自左移入的、用于填补空出位的位置将被赋予位 31 的逻辑状态(整数的符号位)。这意味着，当该整数为正时，这些位将被赋值“0”，而当该整数为负时，则被赋值为“1”。可在输出 **OUT** 位置扫描移位指令的结果。如果 **N** 不等于 0，则 **SHR\_DI** 会将 **CC 0** 位和 **OV** 位设为“0”。

**ENO** 与 **EN** 具有相同的信号状态。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

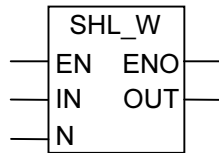
实例



**SHR\_DI** 框由 **I0.0** 位置上的逻辑“1”激活。装载 **MD0** 并将其右移由 **MW4** 指定的位数。结果将被写入 **MD10**。置位 **Q4.0**。

### 11.1.4 SHL\_W 字左移

符号

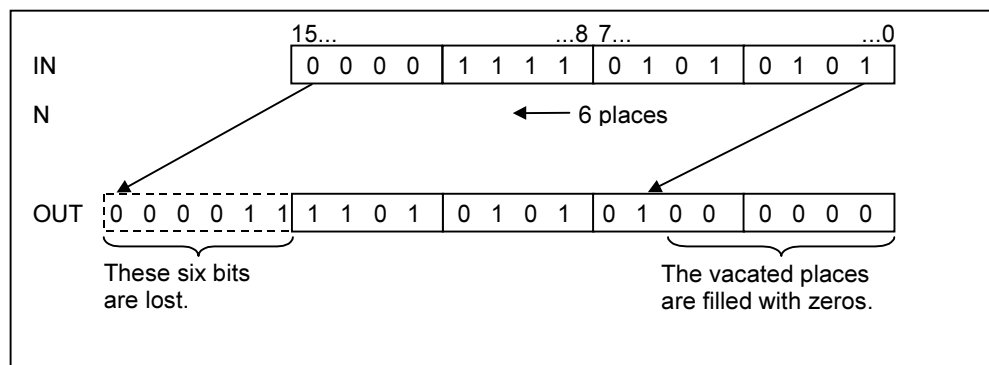


参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	WORD	I、Q、M、L、D	要移位的值
N	WORD	I、Q、M、L、D	要移动的位数
OUT	WORD	I、Q、M、L、D	移位指令的结果

描述

**SHL\_W** (字左移)指令通过使能(EN)输入位置上的逻辑“1”来激活。**SHL\_W** 指令用于将输入 IN 的 0 至 15 位逐位向左移动。16 到 31 位不受影响。输入 N 用于指定移位的位数。若 N 大于 16，此命令会在输出 OUT 位置上写入“0”，并将状态字中的 CC 0 位和 OV 位设置为“0”。将自右移入 N 个零，用以补上空出的位位置。可在输出 OUT 位置扫描移位指令的结果。如果 N 不等于 0，则 **SHL\_W** 会将 CC 0 位和 OV 位设为“0”。

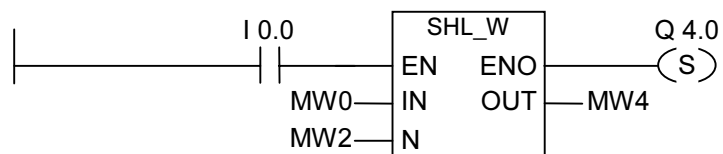
ENO 与 EN 具有相同的信号状态。



状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

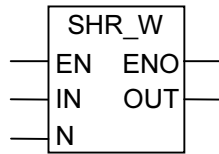
### 实例



SHL\_W 框由 I0.0 位置上的逻辑“1”激活。装载 MW0 并将其左移由 MW2 指定的位数。结果将被写入 MW4。置位 Q4.0。

### 11.1.5 SHR\_W 字右移

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	WORD	I、Q、M、L、D	要移位的值
N	WORD	I、Q、M、L、D	要移动的位数
OUT	WORD	I、Q、M、L、D	字移位指令的结果

描述

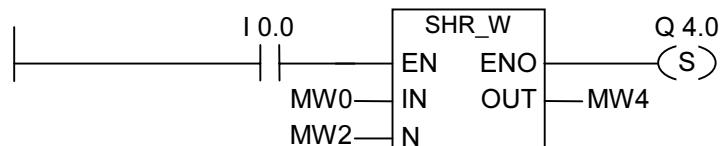
**SHR\_W** (字右移)指令通过使能(EN)输入位置上的逻辑“1”来激活。**SHR\_W** 指令用于将输入 IN 的 0 至 15 位逐位向右移动。16 到 31 位不受影响。输入 N 用于指定移位的位数。若 N 大于 16，此命令会在输出 OUT 位置上写入“0”，并将状态字中的 CC 0 位和 OV 位设置为“0”。将自左移入 N 个零，用以补上空出的位位置。可在输出 OUT 位置扫描移位指令的结果。如果 N 不等于 0，则 **SHR\_W** 会将 CC 0 位和 OV 位设为“0”。

ENO 与 EN 具有相同的信号状态。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

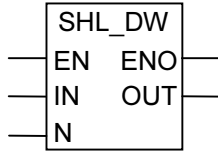
实例



**SHR\_W** 框由 I0.0 位置上的逻辑“1”激活。装载 MW0 并将其右移由 MW2 指定的位数。结果将被写入 MW4。置位 Q4.0。

### 11.1.6 SHL\_DW 双字左移

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DWORD	I、Q、M、L、D	要移位的值
N	WORD	I、Q、M、L、D	要移动的位数
OUT	DWORD	I、Q、M、L、D	双字移位指令的结果

描述

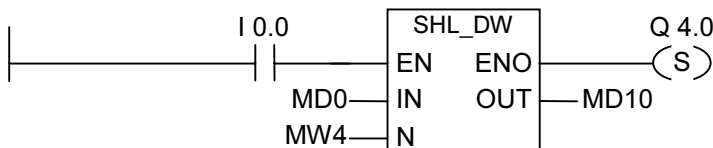
**SHL\_DW** (双字左移)指令通过使能(EN)输入位置上的逻辑“1”来激活。**SHL\_DW** 指令用于将输入 IN 的 0 至 31 位逐位向左移动。输入 N 用于指定移位的位数。若 N 大于 32，此命令会在输出 OUT 位置上写入“0”并将状态字中的 CC 0 位和 OV 位设置为“0”。将自右移入 N 个零，用以补上空出的位位置。可在输出 OUT 位置扫描双字移位指令的结果。如果 N 不等于 0，则 SHL\_DW 会将 CC 0 位和 OV 位设为“0”。

ENO 与 EN 具有相同的信号状态。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

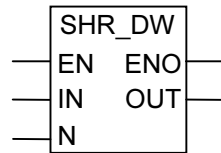
实例



SHL\_DW 框由 I0.0 位置上的逻辑“1”激活。装载 MD0 并将其左移由 MW4 指定的位数。结果将被写入 MD10。置位 Q4.0。

### 11.1.7 SHR\_DW 双字右移

符号

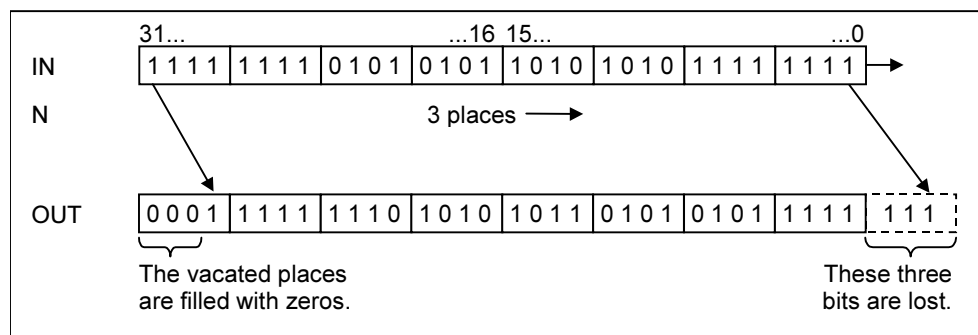


参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DWORD	I、Q、M、L、D	要移位的值
N	WORD	I、Q、M、L、D	要移动的位数
OUT	DWORD	I、Q、M、L、D	双字移位指令的结果

描述

**SHR\_DW** (双字右移)指令通过使能(EN)输入位置上的逻辑“1”来激活。**SHR\_DW**指令用于将输入 **IN** 的 0 至 31 位逐位向右移动。输入 **N** 用于指定移位的位数。若 **N** 大于 32，此命令会在输出 **OUT** 位置上写入“0”并将状态字中的 **CC 0** 位和 **OV** 位设置为“0”。将自左移入 **N** 个零，用以补上空出的位位置。可在输出 **OUT** 位置扫描双字移位指令的结果。如果 **N** 不等于 0，则 **SHR\_DW** 会将 **CC 0** 位和 **OV** 位设为“0”。

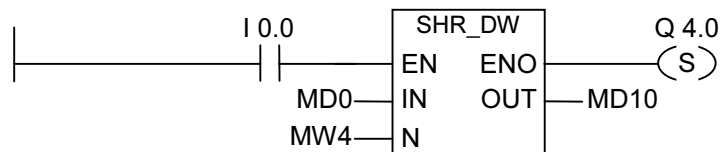
ENO 与 EN 具有相同的信号状态。



状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

### 实例



SHR\_DW 框由 I0.0 位置上的逻辑“1”激活。装载 MD0 并将其右移由 MW4 指定的位数。结果将被写入 MD10。置位 Q4.0。



## 11.2 循环移位指令

### 11.2.1 循环移位指令概述

#### 描述

可使用循环移位指令将输入 IN 的所有内容向左或向右逐位循环移位。移空的位将被移出输入 IN 的位的信号状态补上。

您为输入参数 N 提供的数值指定要循环移位的位数。

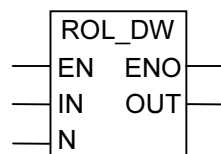
取决于指令的具体情况，循环移位也可以通过状态字的 CC 1 位进行。复位状态字的 CC 0 位为 0。

下列循环移位指令可用：

- ROL\_DW 双字循环左移
- ROR\_DW 双字循环右移

### 11.2.2 ROL\_DW 双字循环左移

#### 符号

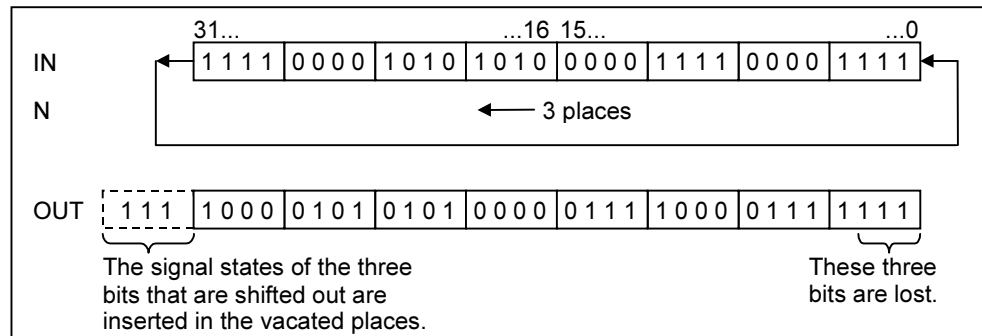


参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DWORD	I、Q、M、L、D	要循环移位的值
N	WORD	I、Q、M、L、D	要循环移位的位数
OUT	DWORD	I、Q、M、L、D	双字循环指令的结果

描述

**ROL\_DW** (双字循环左移)指令通过使能(EN)输入位置上的逻辑“1”来激活。  
**ROL\_DW** 指令用于将输入 IN 的全部内容逐位向左循环移位。输入 N 用于指定循环移位的位数。如果 N 大于 32，则双字 IN 将被循环移位((N-1)对 32 求模，所得的余数) +1 位。自右移入的位位置将被赋予向左循环移出的各个位的逻辑状态。可在输出 OUT 位置扫描双字循环指令的结果。如果 N 不等于 0，则 ROL\_DW 会将 CC 0 位和 OV 位设为“0”。

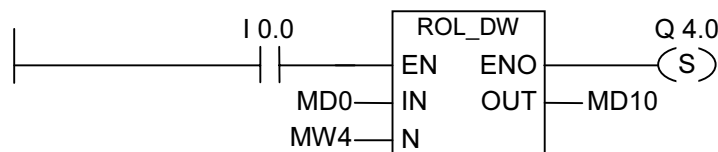
ENO 与 EN 具有相同的信号状态。



状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

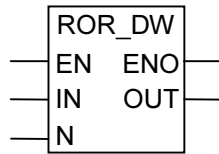
实例



ROL\_DW 框由 I0.0 位置上的逻辑“1”激活。装载 MD0 并将其向左循环移位由 MW4 指定的位数。结果将被写入 MD10。置位 Q4.0。

### 11.2.3 ROR\_DW 双字循环右移

符号

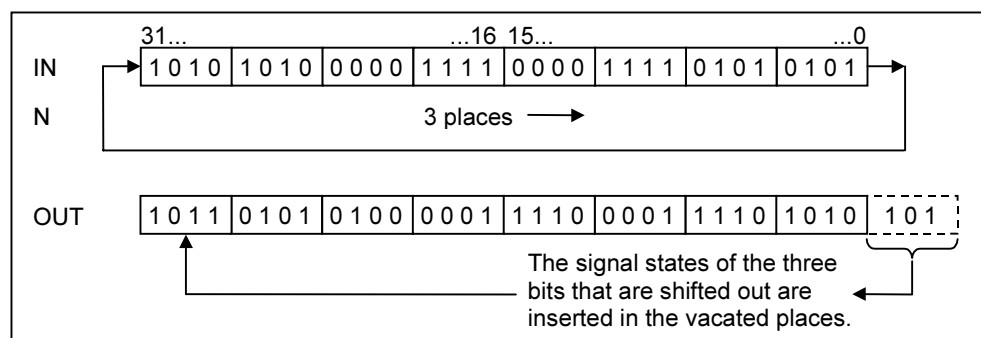


参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN	DWORD	I、Q、M、L、D	要循环移位的值
N	WORD	I、Q、M、L、D	要循环移位的位数
OUT	DWORD	I、Q、M、L、D	双字循环指令的结果

描述

**ROR\_DW** (双字循环右移)指令通过使能(EN)输入位置上的逻辑“1”来激活。**ROR\_DW** 指令用于将输入 IN 的全部内容逐位向右循环移位。输入 N 用于指定循环移位的位数。如果 N 大于 32，则双字 IN 将被循环移位 $((N-1)$ 对 32 求模，所得的余数) +1 位。自左移入的位位置将被赋予向右循环移出的各个位的逻辑状态。可在输出 OUT 位置扫描双字循环指令的结果。如果 N 不等于 0，则 **ROR\_DW** 会将 CC 0 位和 OV 位置为“0”。

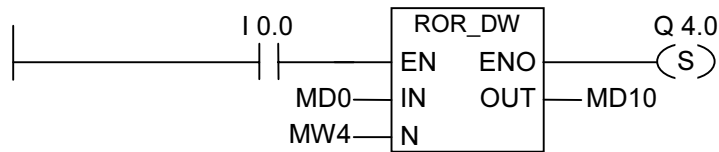
ENO 与 EN 具有相同的信号状态。



状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	x	x	x	x	-	x	x	x	1

实例



ROR\_DW 框由 I0.0 位置上的逻辑“1”激活。装载 MD0 并将其向左循环移位由 MW4 指定的位数。结果将被写入 MD10。置位 Q4.0。

## 12 状态位指令

### 12.1 状态位指令概述

#### 描述

状态位指令属于位逻辑指令，用于对状态字的位进行处理。各状态位指令分别对下列条件之一做出反应，其中每个条件以状态字的一个或多个位来表示：

- 二进制结果位(BR ---I I---)被置位(即信号状态为 1)。
- 数学运算函数发生溢出 (OV ---I I---)或存储溢出 (OS ---I I---)。
- 数学运算函数的结果是无序的 (UO ---I I---)。
- 数学运算函数的结果与 0 的关系有：  
== 0、<> 0、> 0、< 0、>= 0、<= 0。

当状态位指令以串联方式连接时，该指令将根据“与”真值表将其信号状态校验的结果与前一逻辑运算结果合并。当状态位指令以并联方式连接时，该指令将根据“或”真值表将其结果与前一 RLO 合并。

#### 状态字

状态字是 CPU 存储器中的一个寄存器，它包含可以在位逻辑指令和字逻辑指令的地址中引用的位。状态字的结构：

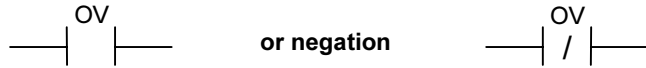
$2^{15}$ ...	...	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
		BR	CC1	CC0	OV	OS	OR	STA	RLO	/FC	

可以通过下列函数求状态字位的值

- 整数数学运算函数、
- 浮点数运算函数。

## 12.2 OV ---| |--- 异常位溢出

符号



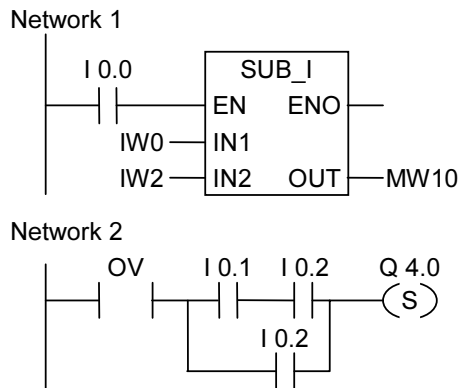
描述

**OV ---| |---** (异常位溢出)或 **OV ---| / |---** (异常位溢出取反)触点符号用于识别上次执行数学运算函数时的溢出。也就是说，函数执行后指令的结果超出了允许的正、负范围。串联使用时，扫描的结果将通过 AND 与 RLO 链接；并联使用时，扫描结果通过 OR 与 RLO 链接。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



I 0.0 的信号状态为“1”时将激活该框。如果数学运算函数“ $IW0 - IW2$ ”的结果超出了允许的整数范围，则置位 OV 位。

OV 的信号状态扫描为“1”。如果 OV 扫描的信号状态为“1”且程序段 2 的 RLO 为“1”，则置位 Q4.0。

**注意**

只有在有两个独立的程序段时，才需要 OV 扫描。否则，如果结果超出了允许的范围，则可以提取为“0”的数学运算函数的 ENO 输出。

## 12.3 OS ---| |--- 存储的异常位溢出

符号



描述

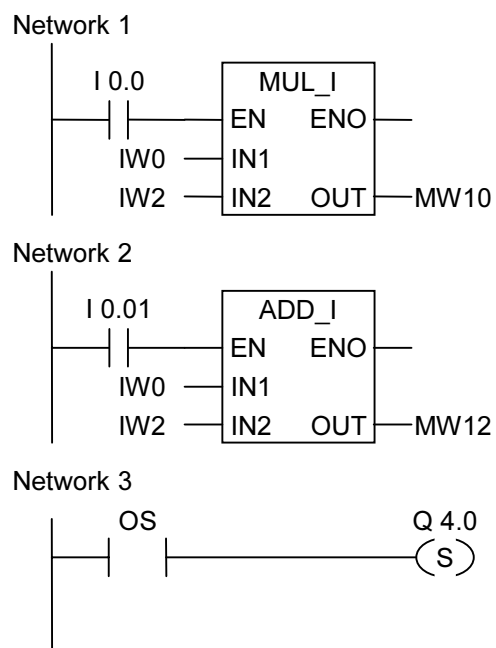
**OS ---| |---** (存储的异常位溢出)或 **OS ---| / |---** (存储的异常位溢出取反)触点符号用于识别和存储数学运算函数中的锁存溢出。如果指令的结果超出了允许的负或正范围，则置位状态字中的 OS 位。与需要在执行后续数学运算函数前重写的 OV 位不同，OS 位在溢出发生时存储。OS 位将保持置位状态，直至离开该块。

串联使用时，扫描的结果将通过 AND 与 RLO 链接；并联使用时，扫描结果通过 OR 与 RLO 链接。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



I0.0 的信号状态为“1”时将激活 MUL\_I 框。I0.1 的逻辑为“1”时将激活 ADD\_I 框。如果其中一个数学运算函数的结果超出了允许的整数范围，将把状态字中的 OS 位置位为“1”。如果 OS 扫描为逻辑“1”，则置位 Q4.0。

---

**注意**

只有在有两个独立的程序段时，才需要 OS 扫描。否则，将可以提取第一个数学运算函数的 ENO 输出，并将其与第二个(层叠排列)数学运算函数的 EN 输入连接。

---



## 12.4 UO ---| |--- 无序异常位

符号



描述

**UO ---| |---** (无序异常位)或 **UO ---| / |---** (无序异常位取反)触点符号用于识别含浮点数的数学运算函数是否无序(也就是说, 数学运算函数中的值是否有无效浮点数)。

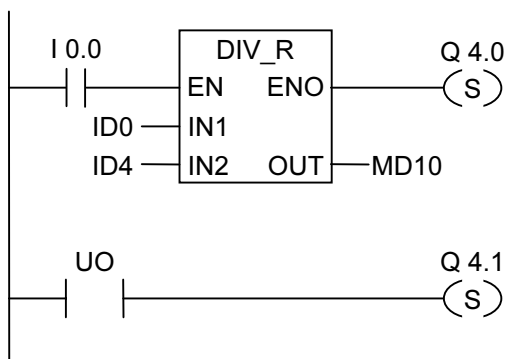
如果含浮点数(UO)的数学运算函数的结果无效, 则信号状态扫描为“1”。如果 **CC 1** 和 **CC 0** 中的逻辑运算显示“无效”, 信号状态扫描的结果将是“0”。

串联使用时, 扫描的结果将通过 **AND** 与 **RLO** 链接; 并联使用时, 扫描结果通过 **OR** 与 **RLO** 链接。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例

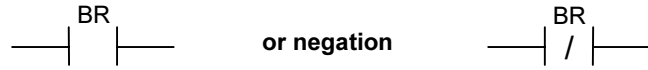


**I0.0** 的信号状态为“1”时将激活该框。如果 **ID0** 或 **ID4** 的值为无效浮点数, 则数学运算函数无效。如果 **EN** 的信号状态 = 1 (激活)且在处理函数 **DIV\_R** 时出错, 则 **ENO** 的信号状态 = 0。

执行函数 **DIV\_R** 时如果其中一个值不是有效的浮点数, 将置位输出 **Q4.1**。

## 12.5 BR ---| |--- 异常位二进制结果

符号



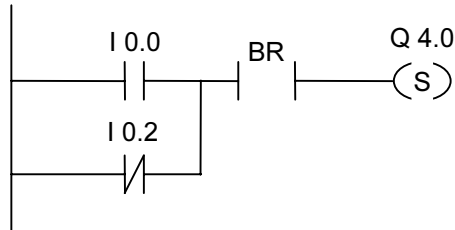
描述

**BR ---| |---** (异常位 BR 存储器)或 **BR ---| / |---** (异常位 BR 存储器取反)触点符号用于测试状态字中 BR 位的逻辑状态。串联使用时,扫描的结果将通过 AND 与 RLO 链接;并联使用时,扫描结果通过 OR 与 RLO 链接。BR 位用于字处理向位处理的转变。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



如果 I0.0 为“1”或 I0.2 为“0”，且除此 RLO 外 BR 位的逻辑状态为“1”，则置位 Q4.0。

## 12.6 ==0 ---| |--- 结果位等于 0

符号



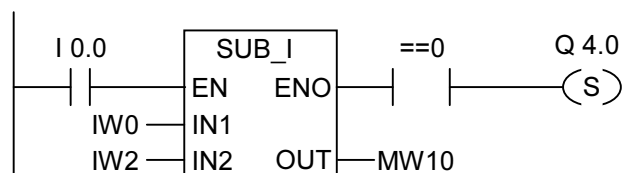
描述

**==0 ---| |---** (结果位等于 0)或 **==0 ---| / |---** (结果位取反后等于 0)触点符号用于识别数学运算函数的结果是否等于“0”。指令扫描状态字的条件代码位 **CC 1** 和 **CC 0**，以确定结果与“0”的关系。串联使用时，扫描的结果将通过 **AND** 与 **RLO** 链接；并联使用时，扫描结果通过 **OR** 与 **RLO** 链接。

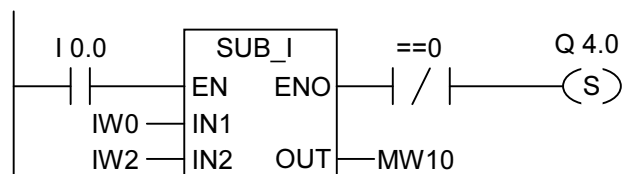
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



I0.0 的信号状态为“1”时将激活该框。如果 IW0 的值等于 IW2 的值，数学运算函数“ $IW0 - IW2$ ”的结果将等于“0”。如果函数得到正确执行且结果等于“0”，则置位 Q4.0。



如果函数得到正确执行且结果不等于“0”，则置位 Q4.0。

## 12.7 <>0 ---| |--- 结果位不等于 0

符号



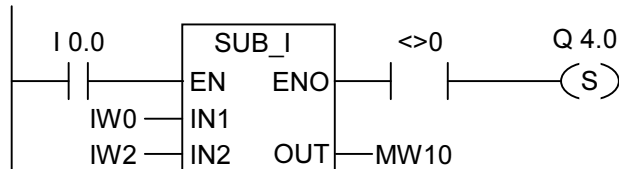
描述

<>0 ---| |--- (结果位不等于 0)或 <>0 ---| / |--- (结果位取反后不等于 0)触点符号用于识别数学运算函数的结果是否不等于“0”。指令扫描状态字的条件代码位 CC 1 和 CC 0，以确定结果与“0”的关系。串联使用时，扫描的结果将通过 AND 与 RLO 链接；并联使用时，扫描结果通过 OR 与 RLO 链接。

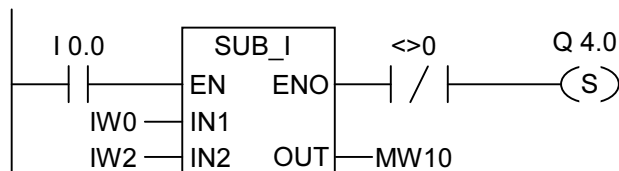
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



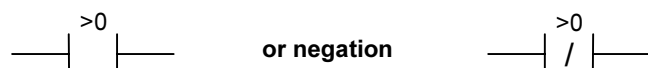
I0.0 的信号状态为“1”时将激活该框。如果 IW0 的值与 IW2 的值不同，数学运算函数“IW0 - IW2”的结果将不等于“0”。如果函数得到正确执行且结果**不等于**“0”，则置位 Q4.0。



如果函数得到正确执行且结果**等于**“0”，则置位 Q4.0。

## 12.8 >0 ---| |--- 结果位大于 0

符号



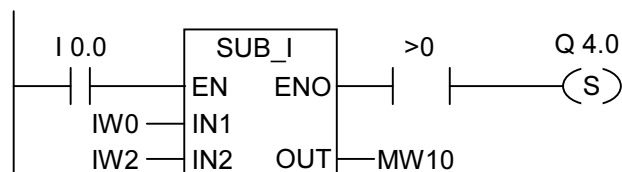
描述

>>0 ---| |--- (结果位大于 0) 或 >>0 ---| / |--- (结果位取反后大于 0) 触点符号用于识别数学运算函数的结果是否大于“0”。指令扫描状态字的条件代码位 CC 1 和 CC 0，以确定结果与“0”的关系。串联使用时，扫描的结果将通过 AND 与 RLO 链接；并联使用时，扫描结果通过 OR 与 RLO 链接。

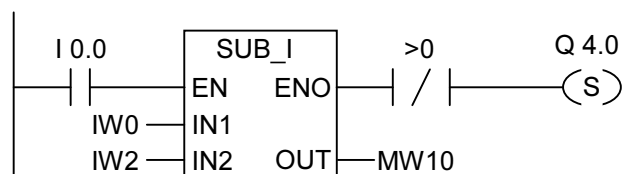
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



I0.0 的信号状态为“1”时将激活该框。如果 IW0 的值大于 IW2 的值，数学运算函数“IW0 - IW2”的结果将大于“0”。如果函数得到正确执行且结果大于“0”，则置位 Q4.0。



如果函数得到正确执行且结果不大于“0”，则置位 Q4.0。

## 12.9 <0 ---| |--- 结果位小于 0

符号



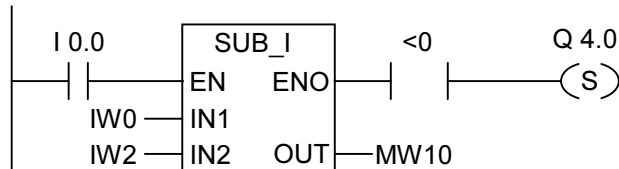
描述

<0 ---| |--- (结果位小于 0)或 <0 ---| / |--- (结果位取反后小于 0)触点符号用于识别数学运算函数的结果是否小于“0”。指令扫描状态字的条件代码位 CC 1 和 CC 0，以确定结果与“0”的关系。串联使用时，扫描的结果将通过 AND 与 RLO 链接；并联使用时，扫描结果通过 OR 与 RLO 链接。

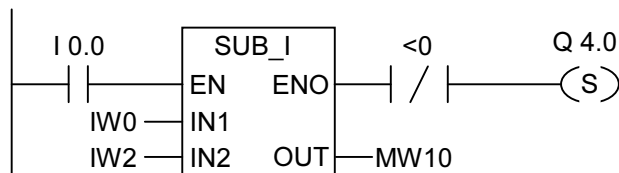
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



I0.0 的信号状态为“1”时将激活该框。如果 IW0 的值小于 IW2 的值，数学运算函数“IW0 - IW2”的结果将小于“0”。如果函数得到正确执行且结果小于“0”，则置位 Q4.0。



如果函数得到正确执行且结果不小于“0”，则置位 Q4.0。

## 12.10 $\geq 0$ ---| |--- 结果位大于等于 0

符号



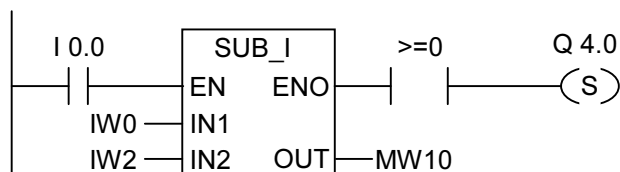
描述

$\geq 0$  ---| |--- (结果位大于等于 0)或  $\geq 0$  ---| / |--- (结果位取反后大于等于 0)触点符号用于识别数学运算函数的结果是否大于或等于“0”。指令扫描状态字的条件代码位 CC 1 和 CC 0，以确定结果与“0”的关系。串联使用时，扫描的结果将通过 AND 与 RLO 链接；并联使用时，扫描结果通过 OR 与 RLO 链接。

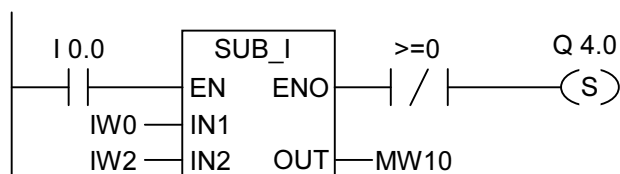
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



I0.0 的信号状态为“1”时将激活该框。如果 IW0 的值大于或等于 IW2 的值，数学运算函数“ $IW0 - IW2$ ”的结果将大于或等于“0”。如果函数得到正确执行且结果大于或等于“0”，则置位 Q4.0。



如果函数得到正确执行且结果不大于或等于“0”，则置位 Q4.0。

## 12.11 <=0 ---| |--- 结果位小于等于 0

符号



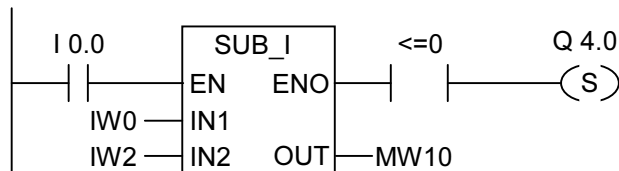
描述

<=0 ---| |--- (结果位小于等于 0)或 <=0 ---| / |--- (结果位取反后小于等于 0)触点符号用于识别数学运算函数的结果是否小于或等于“0”。指令扫描状态字的条件代码位 CC 1 和 CC 0，以确定结果与“0”的关系。串联使用时，扫描的结果将通过 AND 与 RLO 链接；并联使用时，扫描结果通过 OR 与 RLO 链接。

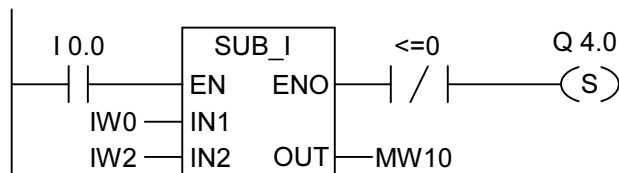
状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

实例



I0.0 的信号状态为“1”时将激活该框。如果 IW0 的值小于或等于 IW2 的值，数学运算函数“IW0 - IW2”的结果将小于或等于“0”。如果函数得到正确执行且结果小于或等于“0”，则置位 Q4.0。



如果函数得到正确执行且结果不小于或等于“0”，则置位 Q4.0。



## 13 定时器指令

### 13.1 定时器指令概述

#### 描述

有关设置和选择正确的定时信息，请参阅定时器在存储器中的位置与定时器组件描述。

以下定时器指令可用：

- S\_PULSE 脉冲 S5 定时器
- S\_PEXT 扩展脉冲 S5 定时器
- S\_ODT 接通延时 S5 定时器
- S\_ODTS 保持接通延时 S5 定时器
- S\_OFFDT 断开延时 S5 定时器
- ---( SP ) 脉冲定时器线圈
- ---( SE ) 扩展脉冲定时器线圈
- ---( SD ) 接通延时定时器线圈
- ---( SS ) 保持接通延时定时器线圈
- ---( SA ) 断开延时定时器线圈

## 13.2 定时器在存储器中的位置与定时器组件

### 存储器中的区域

CPU 存储器中有一个为定时器保留的区域。此存储区域为每个定时器的地址保留一个 16 位字。梯形图指令集支持 256 个定时器。请参阅 CPU 的技术信息以建立多个可用的定时器字。

以下功能可访问定时器存储区域：

- 定时器指令
- 利用时钟定时更新定时器字。在运行模式下，CPU 的这个功能可按照由时间基准指定的间隔将给定的时间值递减一个单位，直到该时间值等于零为止。

### 时间值

定时器字的 0 到 9 位包含二进制编码的时间值。此时间值指定多个单位。时间更新可按照由时间基准指定的间隔将时间值递减一个单位。递减会持续进行，直至时间值等于零为止。可以在累加器 1 的低字中以二进制、十六进制或二进制编码的十进制(BCD)格式装入时间值。

可以用以下任一格式预装入时间值：

- **W#16#wxyz**
  - 其中，w = 时间基准(即时间间隔或分辨率)
  - 此处 xyz = 以二进制编码的十进制格式表示的时间值
- **S5T#aH\_bM\_cS\_dMS**
  - 其中，H = 小时，M = 分钟，S = 秒，MS = 毫秒；  
a、b、c、d 由用户定义。
  - 自动选择时间基准，其值舍入为具有该时间基准的下一个较小的数字。

可以输入的最大时间值是 9,990s 或 2H\_46M\_30S。

S5TIME#4S = 4 秒

s5t#2h\_15m = 2 小时 15 分钟

S5T#1H\_12M\_18S = 1 小时 12 分钟 18 秒

## 时间基准

定时器字的第 12 和 13 位包含二进制编码的时间基准。时间基准定义时间值以一个单位递减的间隔。最小的时间基准是 10ms，最大为 10s。

时间基准	时间基准的二进制编码
10ms	00
100ms	01
1 s	10
10 s	11

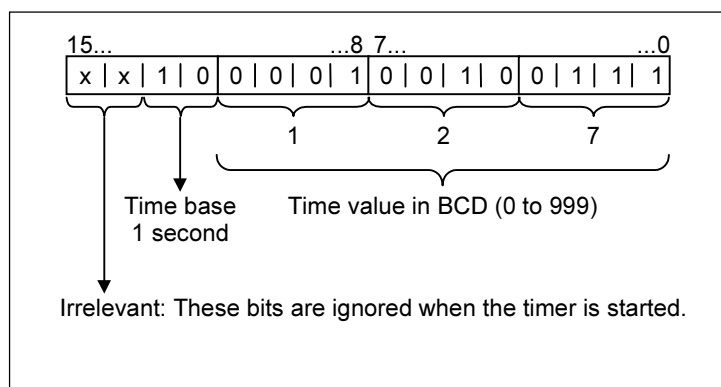
不接受超过 2 小时 46 分 30 秒的数值。对于范围限制(例如, 2h10ms)而言, 过高的分辨率将被截尾为有效分辨率。S5TIME 的通用格式对范围和分辨率有如下限制:

分辨率	范围
0.01s	10MS 到 9S_990MS
0.1s	100MS 到 1M_39S_900MS
1s	1S 到 16M_39S
10s	10S 到 2H_46M_30S

## 时间单元中的位组态

定时器启动时, 定时器单元的内容用作时间值。定时器单元的 0 到 11 位容纳二进制编码的十进制时间值(BCD 格式: 由四位组成的每一组都包含一个十进制值的二进制代码)。第 12 和 13 位存放二进制编码的时间基准。

下图显示装载了时间值 127, 时间基准 1 秒的定时器单元的内容:

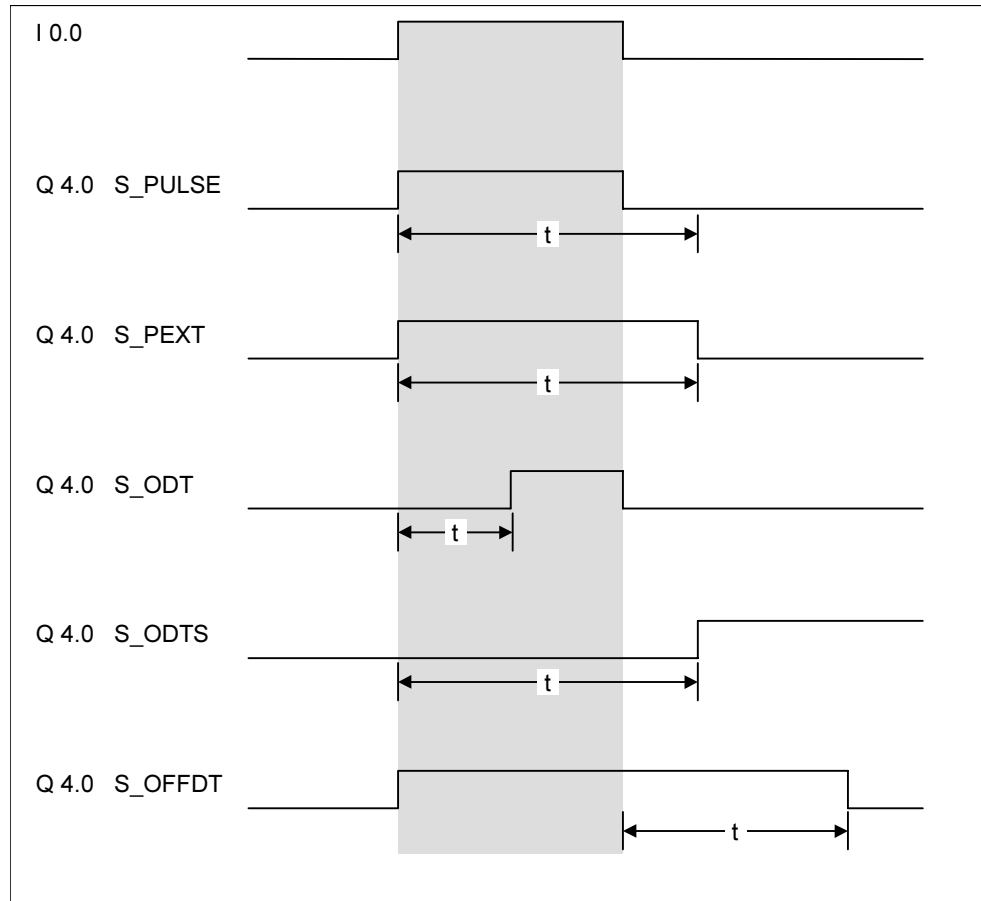


### 读取时间和时间基准

每个定时器逻辑框提供两种输出：**BI** 和 **BCD**，从中可指示一个字位置。**BI** 输出提供二进制格式的时间值。**BCD** 输出提供二进制编码的十进制(BCD)格式的时间基准和时间值。

### 选择正确的定时器

此概述旨在帮助您为定时作业选择正确的定时器。

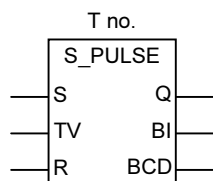


定时器	描述
<b>S_PULSE</b> 脉冲定时器	输出信号保持在 1 的最长时间与编程时间值 $t$ 相同。如果输入信号变为 0，则输出信号停留在 1 的时间会很短。
<b>S_PEXT</b> 扩展脉冲定时器	输出信号在编程时间长度内始终保持在 1，而与输入信号停留在 1 的时间长短无关。
<b>S_ODT</b> 接通延时定时器	仅在编程时间到期，且输入信号仍为 1 时，输出信号变为 1。
<b>S_ODTS</b> 带保持的接通延时定时器	输出信号仅在编程时间到期时才从 0 变为 1，而与输入信号停留在 1 的时间长短无关。
<b>S_OFFDT</b> 断开延时定时器	在输入信号变为 1 或在定时器运行时，输出信号变为 1。当输入信号从 1 变为 0 时启动计时器。

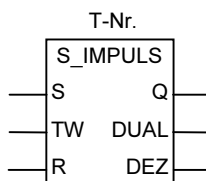
## 13.3 S\_PULSE 脉冲 S5 定时器

### 符号

#### English



#### German



参数英语	参数德语	数据类型	存储区	描述
T 编号	T-Nr.	TIMER	T	定时器标识号，范围取决于 CPU
S	S	BOOL	I、Q、M、L、D	使能输入
TV	TW	S5TIME	I、Q、M、L、D	预设时间值
R	R	BOOL	I、Q、M、L、D	复位输入
BI	DUAL	WORD	I、Q、M、L、D	剩余时间值，整型格式
BCD	DEZ	WORD	I、Q、M、L、D	剩余时间值，BCD 格式
Q	Q	BOOL	I、Q、M、L、D	定时器的状态

### 描述

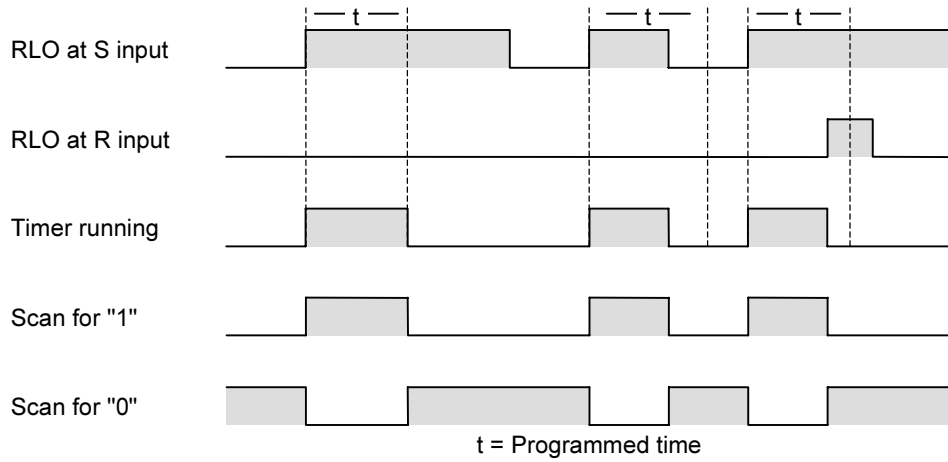
如果在启动(S)输入端有一个上升沿，**S\_PULSE**(脉冲 S5 定时器)将启动指定的定时器。信号变化始终是启用定时器的必要条件。定时器在输入端 S 的信号状态为“1”时运行，但最长周期是由输入端 TV 指定的时间值。只要定时器运行，输出端 Q 的信号状态就为“1”。如果在时间间隔结束前，S 输入端从“1”变为“0”，则定时器将停止。这种情况下，输出端 Q 的信号状态为“0”。

如果在定时器运行期间定时器复位(R)输入从“0”变为“1”时，则定时器将被复位。当前时间和时间基准也被设置为零。如果定时器不是正在运行，则定时器 R 输入端的逻辑“1”没有任何作用。

可在输出端 BI 和 BCD 扫描当前时间值。时间值在 BI 端是二进制编码，在 BCD 端是 BCD 编码。当前时间值为初始 TV 值减去定时器启动后经过的时间。

### 时序图

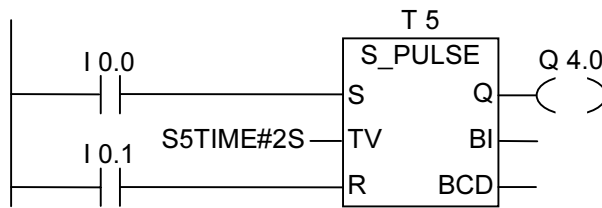
脉冲定时器特性曲线：



### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

### 实例



如果输入端 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿)，则定时器 T5 将启动。只要 I0.0 为“1”，定时器就将继续运行指定的两秒(2s)时间。如果定时器达到预定时间前，I0.0 的信号状态从“1”变为“0”，则定时器将停止。如果输入端 I0.1 的信号状态从“0”变为“1”，而定时器仍在运行，则时间复位。

只要定时器运行，输出端 Q4.0 就是逻辑“1”，如果定时器预设时间结束或复位，则输出端 Q4.0 变为“0”。

## 13.4 S\_PEXT 扩展脉冲 S5 定时器

### 符号



参数英语	参数德语	数据类型	存储区	描述
T 编号	T-Nr.	TIMER	T	定时器标识号，范围取决于 CPU
S	S	BOOL	I、Q、M、 L、D	使能输入
TV	TW	S5TIME	I、Q、M、 L、D	预设时间值
R	R	BOOL	I、Q、M、 L、D	复位输入
BI	DUAL	WORD	I、Q、M、 L、D	剩余时间值，整型格式
BCD	DEZ	WORD	I、Q、M、 L、D	剩余时间值，BCD 格式
Q	Q	BOOL	I、Q、M、 L、D	定时器的状态

### 描述

如果在启动(S)输入端有一个上升沿，**S\_PEXT**(扩展脉冲 S5 定时器)将启动指定的定时器。信号变化始终是启用定时器的必要条件。定时器以在输入端 TV 指定的预设时间间隔运行，即使在时间间隔结束前，S 输入端的信号状态变为“0”。只要定时器运行，输出端 Q 的信号状态就为“1”。如果在定时器运行期间输入端 S 的信号状态从“0”变为“1”，则将使用预设的时间值重新启动(“重新触发”)定时器。

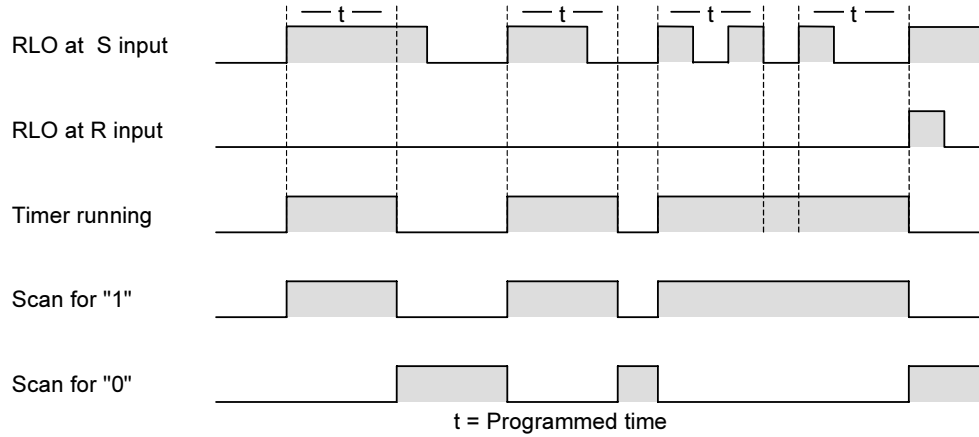
如果在定时器运行期间复位(R)输入从“0”变为“1”，则定时器复位。当前时间和时间基准被设置为零。

可在输出端 BI 和 BCD 扫描当前时间值。时间值在 BI 处为二进制编码，在 BCD 处为 BCD 编码。当前时间值为初始 TV 值减去定时器启动后经过的时间。

参见定时器在存储器中的位置与定时器组件。

### 时序图

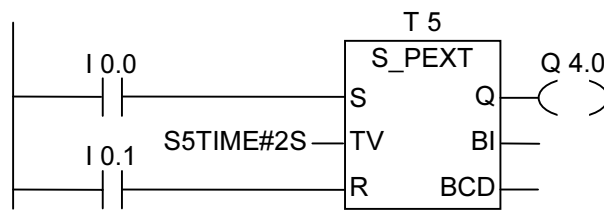
扩展脉冲定时器特征曲线:



### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

### 实例



如果输入端 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿), 则定时器 T5 将启动。定时器将继续运行指定的两秒(2 秒)时间, 而不会受到输入端 S 处下降沿的影响。如果在定时器达到预定时间前, I0.0 的信号状态从“0”变为“1”, 则定时器将被重新触发。只要定时器运行, 输出端 Q4.0 就为逻辑“1”。



## 13.5 S\_ODT 接通延时 S5 定时器

### 符号



参数英语	参数德语	数据类型	存储区	描述
T 编号	T-Nr.	TIMER	T	定时器标识号，范围取决于 CPU
S	S	BOOL	I、Q、M、 L、D	使能输入
TV	TW	S5TIME	I、Q、M、 L、D	预设时间值
R	R	BOOL	I、Q、M、 L、D	复位输入
BI	DUAL	WORD	I、Q、M、 L、D	剩余时间值，整型格式
BCD	DEZ	WORD	I、Q、M、 L、D	剩余时间值，BCD 格式
Q	Q	BOOL	I、Q、M、 L、D	定时器的状态

### 描述

如果在启动(S)输入端有一个上升沿，**S\_ODT**(接通延时 S5 定时器)将启动指定的定时器。信号变化始终是启用定时器的必要条件。只要输入端 S 的信号状态为正，定时器就以在输入端 TV 指定的时间间隔运行。定时器达到指定时间而没有出错，并且 S 输入端的信号状态仍为“1”时，输出端 Q 的信号状态为“1”。如果定时器运行期间输入端 S 的信号状态从“1”变为“0”，定时器将停止。这种情况下，输出端 Q 的信号状态为“0”。

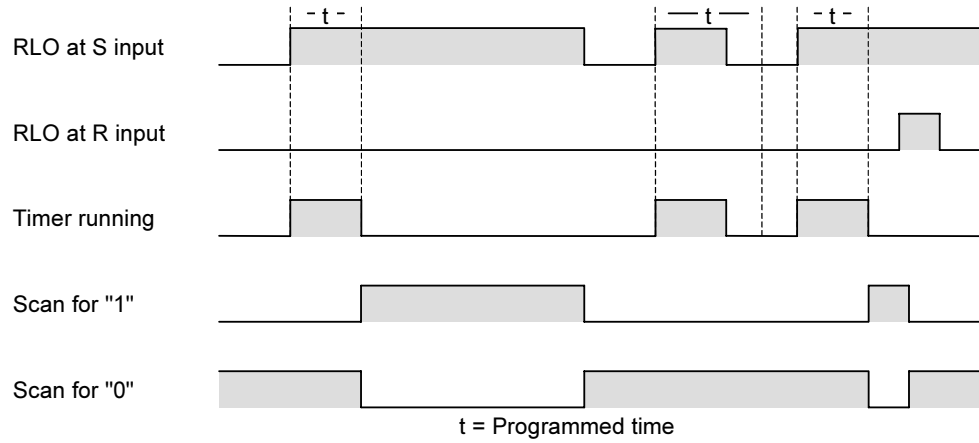
如果在定时器运行期间复位(R)输入从“0”变为“1”，则定时器复位。当前时间和时间基准被设置为零。然后，输出端 Q 的信号状态变为“0”。如果在定时器没有运行时 R 输入端有一个逻辑“1”，并且输入端 S 的 RLO 为“1”，则定时器也复位。

可在输出端 BI 和 BCD 扫描当前时间值。时间值在 BI 处为二进制编码，在 BCD 处为 BCD 编码。当前时间值为初始 TV 值减去定时器启动后经过的时间。

参见定时器在存储器中的位置与定时器组件。

### 时序图

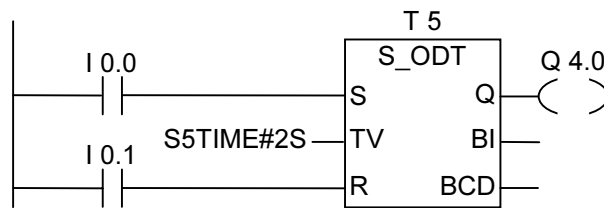
接通延时定时器特征曲线:



### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

### 实例



如果 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿), 则定时器 T5 将启动。如果指定的两秒时间结束并且输入端 I0.0 的信号状态仍为“1”, 则输出端 Q4.0 将为“1”。如果 I0.0 的信号状态从“1”变为“0”, 则定时器停止, 并且 Q4.0 将为“0” (如果 I0.1 的信号状态从“0”变为“1”, 则无论定时器是否运行, 时间都复位)。

## 13.6 S\_ODTS 保持接通延时 S5 定时器

### 符号



参数英语	参数德语	数据类型	存储区	描述
T 编号	T-Nr.	TIMER	T	定时器标识号，范围取决于 CPU
S	S	BOOL	I、Q、M、L、D	使能输入
TV	TW	S5TIME	I、Q、M、L、D	预设时间值
R	R	BOOL	I、Q、M、L、D	复位输入
BI	DUAL	WORD	I、Q、M、L、D	剩余时间值，整型格式
BCD	DEZ	WORD	I、Q、M、L、D	剩余时间值，BCD 格式
Q	Q	BOOL	I、Q、M、L、D	定时器的状态

### 描述

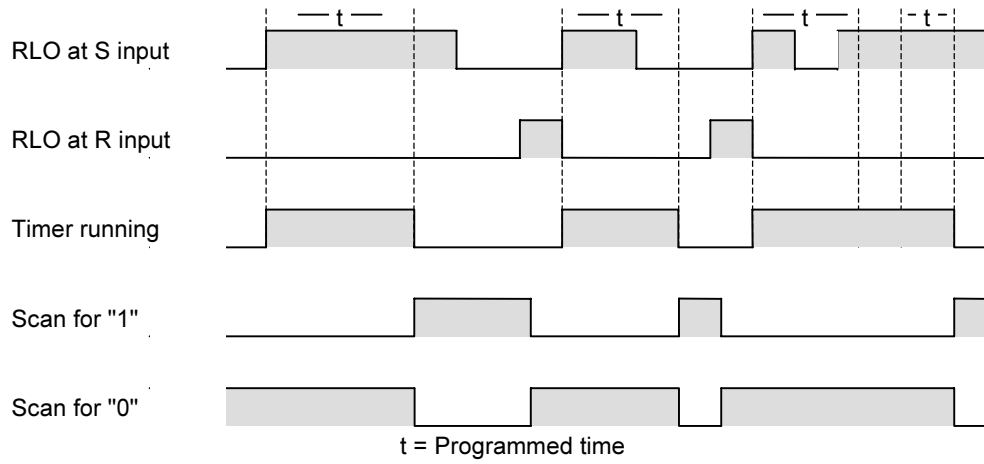
如果在启动(S)输入端有一个上升沿，**S\_ODTS**(保持接通延时 S5 定时器)将启动指定的定时器。信号变化始终是启用定时器的必要条件。定时器以在输入端 TV 指定的时间间隔运行，即使在时间间隔结束前，输入端 S 的信号状态变为“0”。定时器预定时间结束时，输出端 Q 的信号状态为“1”，而无论输入端 S 的信号状态如何。如果在定时器运行时输入端 S 的信号状态从“0”变为“1”，则定时器将以指定的时间重新启动(重新触发)。

如果复位(R)输入从“0”变为“1”，则无论 S 输入端的 RLO 如何，定时器都将复位。然后，输出端 Q 的信号状态变为“0”。

可在输出端 BI 和 BCD 扫描当前时间值。时间值在 BI 端是二进制编码，在 BCD 端是 BCD 编码。当前时间值为初始 TV 值减去定时器启动后经过的时间。

### 时序图

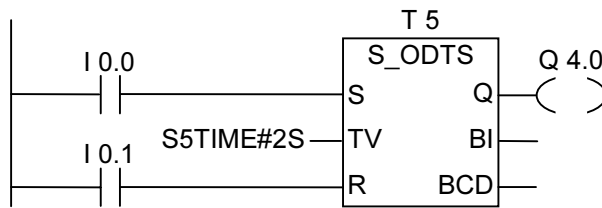
保持接通延时定时器特征曲线：



### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

### 实例



如果 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿)，则定时器 T5 将启动。无论 I0.0 的信号是否从“1”变为“0”，定时器都将运行。如果在定时器达到指定时间前，I0.0 的信号状态从“0”变为“1”，则定时器将重新触发。如果定时器达到指定时间，则输出端 Q4.0 将变为“1”。(如果输入端 I0.1 的信号状态从“0”变为“1”，则无论 S 处的 RLO 如何，时间都将复位。)

## 13.7 S\_OFFDT 断开延时 S5 定时器

### 符号



参数英语	参数德语	数据类型	存储区	描述
T 编号	T-Nr.	TIMER	T	定时器标识号，范围取决于 CPU
S	S	BOOL	I、Q、M、 L、D	使能输入
TV	TW	S5TIME	I、Q、M、 L、D	预设时间值
R	R	BOOL	I、Q、M、 L、D	复位输入
BI	DUAL	WORD	I、Q、M、 L、D	剩余时间值，整型格式
BCD	DEZ	WORD	I、Q、M、 L、D	剩余时间值，BCD 格式
Q	Q	BOOL	I、Q、M、 L、D	定时器的状态

### 描述

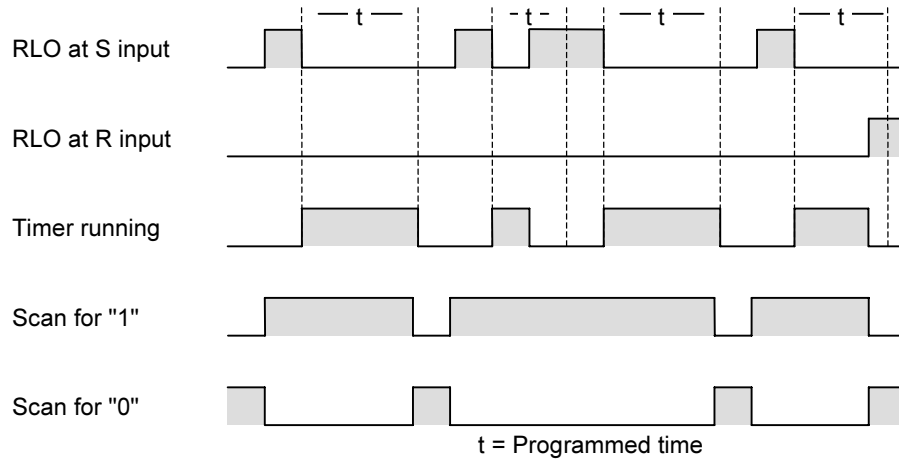
如果在启动(S)输入端有一个下降沿，**S\_OFFDT**(断开延时 S5 定时器)将启动指定的定时器。信号变化始终是启用定时器的必要条件。如果 S 输入端的信号状态为“1”，或定时器正在运行，则输出端 Q 的信号状态为“1”。如果在定时器运行期间输入端 S 的信号状态从“0”变为“1”时，定时器将复位。输入端 S 的信号状态再次从“1”变为“0”后，定时器才能重新启动。

如果在定时器运行期间复位(R)输入从“0”变为“1”时，定时器将复位。

可在输出端 BI 和 BCD 扫描当前时间值。时间值在 BI 端是二进制编码，在 BCD 端是 BCD 编码。当前时间值为初始 TV 值减去定时器启动后经过的时间。

### 时序图

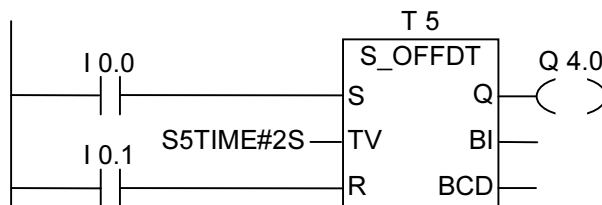
断开延时定时器特征曲线:



### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	x	x	x	1

### 实例



如果 I0.0 的信号状态从“1”变为“0”，则定时器启动。

I0.0 为“1”或定时器运行时，Q4.0 为“1”。(如果在定时器运行期间 I0.1 的信号状态从“0”变为“1”，则定时器复位)。

## 13.8 ---( SP ) 脉冲定时器线圈

### 符号

英语	德语
<T 编号.>	<T 编号>
---( SP )	---( SI )
<时间值>	<时间值>

参数	数据类型	存储区	描述
<T 编号>	TIMER	T	定时器标识号, 范围取决于 CPU
<时间值>	S5TIME	I、Q、M、L、D	预设时间值

### 描述

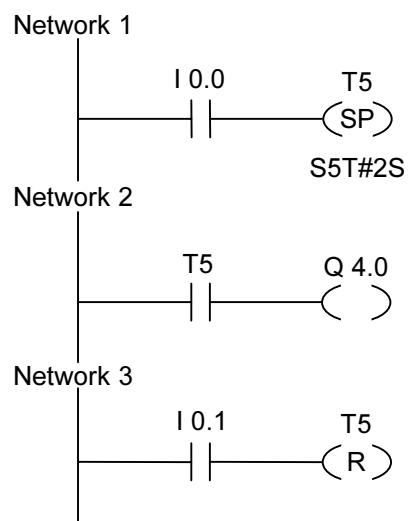
如果 RLO 状态有一个上升沿, ---( SP ) (脉冲定时器线圈)将以该<时间值>启动指定的定时器。只要 RLO 保持正值(“1”), 定时器就继续运行指定的时间间隔。只要定时器运行, 计数器的信号状态就为“1”。如果在达到时间值前, RLO 中的信号状态从“1”变为“0”, 则定时器将停止。这种情况下, 对于“1”的扫描始终产生结果“0”。

参见“存储器中定时器的位置和定时器的组件”和 S\_PULSE(脉冲 S5 定时器)。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	-	-	0

实例



如果输入端 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿)，则定时器 T5 启动。只要输入端 I0.0 的信号状态为“1”，定时器就继续运行指定的两秒时间。如果在指定的时间结束前输入端 I0.0 的信号状态从“1”变为“0”，则定时器停止。

只要定时器运行，输出端 Q4.0 的信号状态就为“1”。如果输入端 I0.1 的信号状态从“0”变为“1”，定时器 T5 将复位，定时器停止，并将时间值的剩余部分清为“0”。



## 13.9 ---( SE ) 扩展脉冲定时器线圈

### 符号

英语	德语
<T 编号>	<T 编号>
---( SE )	---( SV )
<时间值>	<时间值>

参数	数据类型	存储区	描述
<T 编号>	TIMER	T	定时器标识号, 范围取决于 CPU
<时间值>	S5TIME	I、Q、M、L、D	预设时间值

### 描述

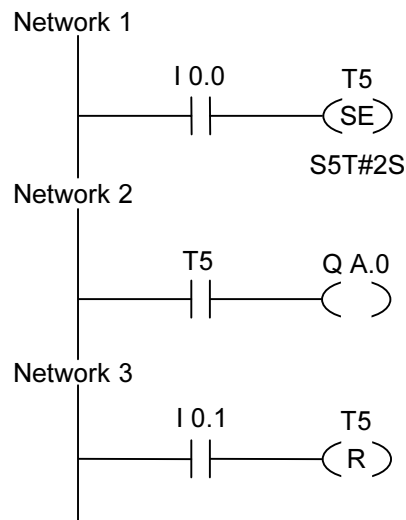
如果 RLO 状态有一个上升沿, ---( SE ) (扩展脉冲定时器线圈)将以指定的<时间值>启动指定的定时器。定时器继续运行指定的时间间隔, 即使定时器达到指定时间前 RLO 变为“0”。只要定时器运行, 计数器的信号状态就为“1”。如果在定时器运行期间 RLO 从“0”变为“1”, 则将以指定的时间值重新启动定时器(重新触发)。

参见“存储器中定时器的位置和定时器的组件”和 S\_PEXT(扩展脉冲 S5 定时器)。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	-	-	0

实例



如果输入端 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿)，则定时器 T5 启动。定时器继续运行，而无论 RLO 是否出现下降沿。如果在定时器达到指定时间前 I0.0 的信号状态从“0”变为“1”，则定时器重新触发。

只要定时器运行，输出端 Q4.0 的信号状态就为“1”。如果输入端 I0.1 的信号状态从“0”变为“1”，定时器 T5 将复位，定时器停止，并将时间值的剩余部分清为“0”。

## 13.10 ---( SD ) 接通延时定时器线圈

### 符号

英语	德语
<T 编号>	<T 编号>
---( SD )	---( SE )
<时间值>	<时间值>

参数	数据类型	存储区	描述
<T 编号>	TIMER	T	定时器标识号，范围取决于 CPU
<时间值>	S5TIME	I、Q、M、L、D	预设时间值

### 描述

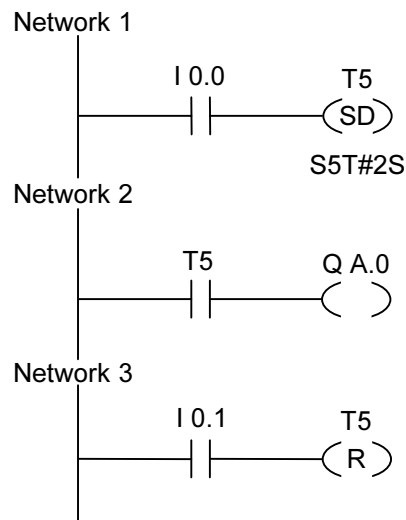
如果 RLO 状态有一个上升沿，---( SD ) (接通延时定时器线圈) 将以该<时间值>启动指定的定时器。如果达到该<时间值>而没有出错，且 RLO 仍为“1”，则定时器的信号状态为“1”。如果在定时器运行期间 RLO 从“1”变为“0”，则定时器复位。这种情况下，对于“1”的扫描始终产生结果“0”。

参见“存储器中定时器的位置和定时器的组件”和 S\_ODT(接通延时 S5 定时器)。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	-	-	0

实例



如果输入端 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿)，则定时器 T5 启动。如果指定时间结束而输入端 I0.0 的信号状态仍为“1”，则输出端 Q4.0 的信号状态将为“1”。

如果输入端 I0.0 的信号状态从“1”变为“0”，则定时器保持空闲，并且输出端 Q4.0 的信号状态将为“0”。如果输入端 I0.1 的信号状态从“0”变为“1”，定时器 T5 将复位，定时器停止，并将时间值的剩余部分清为“0”。

## 13.11 ---( SS ) 保持性接通延时定时器线圈

### 符号

英语	德语
<T 编号>	<T 编号>
---( SS )	---( SS )
<时间值>	<时间值>

参数	数据类型	存储区	描述
<T 编号>	TIMER	T	定时器标识号，范围取决于 CPU
<时间值>	S5TIME	I、Q、M、L、D	预设时间值

### 描述

如果 RLO 状态有一个上升沿，---( SS ) (保持接通延时定时器线圈)将启动指定的定时器。如果达到时间值，定时器的信号状态为“1”。只有明确进行复位，定时器才可能重新启动。只有复位才能将定时器的信号状态设为“0”。

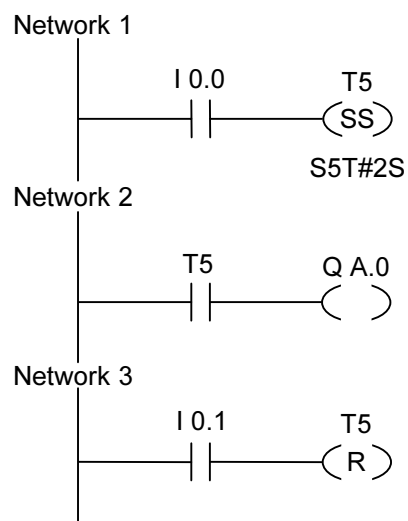
如果在定时器运行期间 RLO 从“0”变为“1”，则定时器以指定的时间值重新启动。

参见“存储器中定时器的位置和定时器的组件”和 S\_ODTS(保持接通延时 S5 定时器)。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	-	-	0

实例



如果输入端 I0.0 的信号状态从“0”变为“1” (RLO 中的上升沿)，则定时器 T5 启动。如果在定时器达到指定时间前输入端 I0.0 的信号状态从“0”变为“1”，则定时器将重新触发。如果定时器达到指定时间，则输出端 Q4.0 将变为“1”。输入端 I0.1 的信号状态“1”将复位定时器 T5，使定时器停止，并将时间值的剩余部分清为“0”。

## 13.12 ---( SF )断开延时定时器线圈

### 符号

英语	德语
<T 编号>	<T 编号>
---( SF )	---( SA )
<时间值>	<时间值>

参数	数据类型	存储区	描述
<T 编号>	TIMER	T	定时器标识号, 范围取决于 CPU
<时间值>	S5TIME	I、Q、M、L、D	预设时间值

### 描述

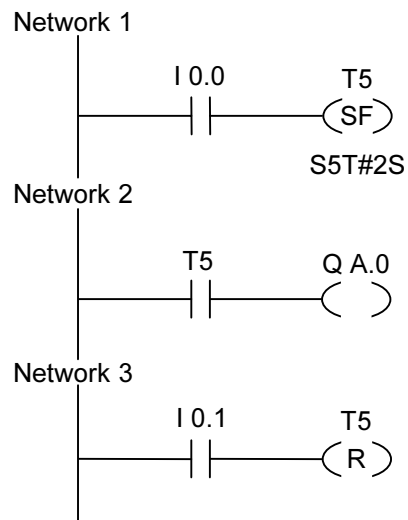
如果 RLO 状态有一个下降沿, ---( SF ) (断开延时定时器线圈)将启动指定的定时器。当 RLO 为“1”时或只要定时器在<时间值>时间间隔内运行, 定时器就为“1”。如果在定时器运行期间 RLO 从“0”变为“1”, 则定时器复位。只要 RLO 从“1”变为“0”, 定时器即会重新启动。

参见“存储器中定时器的位置和定时器的组件”和 S\_OFFDT(断开延时 S5 定时器)。

### 状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	-	-	0

实例



如果输入端 I0.0 的信号状态从“1”变为“0”，则定时器启动。

如果输入端 I0.0 为“1”或定时器正在运行，则输出端 Q4.0 的信号状态为“1”。  
 如果输入端 I0.1 的信号状态从“0”变为“1”，定时器 T5 将复位，定时器停止，  
 并将时间值的剩余部分清为“0”。



## 14 字逻辑指令

### 14.1 字逻辑指令概述

#### 描述

字逻辑指令按照布尔逻辑逐位比较字(16位)和双字(32位)对。

如果输出 OUT 的结果不等于 0，将把状态字的 CC 1 位设置为“1”。

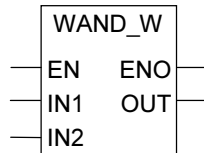
如果输出 OUT 的结果等于 0，将把状态字的 CC 1 位设置为“0”。

可以使用下列字逻辑指令：

- WAND\_W (字)单字与运算
- WOR\_W (字)单字或运算
- WXOR\_W (字)单字异或运算
  
- WAND\_DW (字)双字与运算
- WOR\_DW (字)双字或运算
- WXOR\_DW (字)双字异或运算

## 14.2 WAND\_W (字)单字与运算

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	WORD	I、Q、M、L、D	逻辑运算的第一个值
IN2	WORD	I、Q、M、L、D	逻辑运算的第二个值
OUT	WORD	I、Q、M、L、D	逻辑运算的结果字

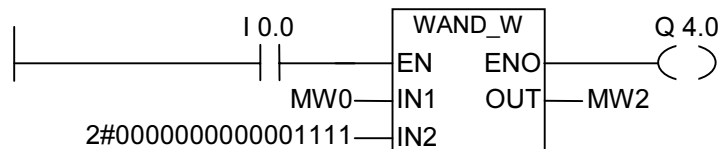
描述

使能(EN)输入的信号状态为“1”时将激活 **WAND\_W** (字与运算)，并逐位对 IN1 和 IN2 处的两个字值进行与运算。按纯位模式来解释这些值。可以在输出 OUT 扫描结果。ENO 与 EN 的逻辑状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	x	0	0	-	x	1	1	1

实例



如果 I0.0 为“1”，则执行指令。在 MW0 的位中，只有 0 至 3 位是相关的，其余位被 IN2 字位模式屏蔽：

MW0 = 01010101 01010101

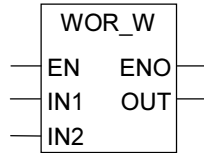
IN2 = 00000000 00001111

MW0 AND IN2 = MW2 = 00000000 00000101

如果执行了指令，则 Q4.0 为“1”。

## 14.3 WOR\_W (字)单字或运算

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	WORD	I、Q、M、L、D	逻辑运算的第一个值
IN2	WORD	I、Q、M、L、D	逻辑运算的第二个值
OUT	WORD	I、Q、M、L、D	逻辑运算的结果字

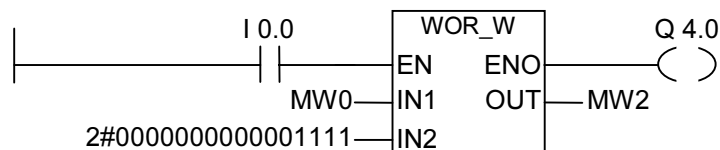
描述

使能(EN)输入的信号状态为“1”时将激活 **WOR\_W** (单字或运算), 并逐位对 **IN1** 和 **IN2** 处的两个字值进行或运算。按纯位模式来解释这些值。可以在输出 **OUT** 扫描结果。**ENO** 与 **EN** 的逻辑状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	x	0	0	-	x	1	1	1

实例



如果 I0.0 为“1”，则执行指令。将位 0 至 3 设置为“1”，不改变 MW0 的所有其它位。

MW0 = 01010101 01010101

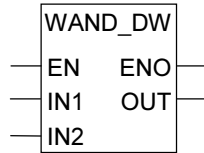
IN2 = 00000000 00001111

MW0 OR IN2=MW2 = 01010101 01011111

如果执行了指令，则 Q4.0 为“1”。

## 14.4 WAND\_DW (字)双字与运算

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DWORD	I、Q、M、L、D	逻辑运算的第一个值
IN2	DWORD	I、Q、M、L、D	逻辑运算的第二个值
OUT	DWORD	I、Q、M、L、D	逻辑运算的结果双字

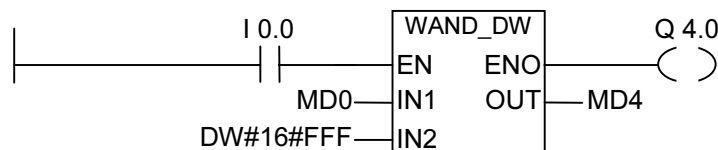
描述

使能(EN)输入的信号状态为“1”时将激活 **WAND\_DW** (双字与运算)，并逐位对 IN1 和 IN2 处的两个字值进行与运算。按纯位模式来解释这些值。可以在输出 OUT 扫描结果。ENO 与 EN 的逻辑状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	x	0	0	-	x	1	1	1

实例



如果 I0.0 为“1”，则执行指令。在 MD0 的位中，只有 0 和 11 位是相关的，其余位被 IN2 位模式屏蔽：

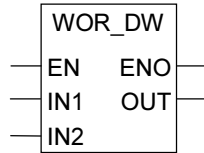
```

MD0           =      01010101 01010101 01010101 01010101
IN2           =      00000000 00000000 00001111 11111111
MD0 AND IN2 = MD4 =      00000000 00000000 00001010 01010101
    
```

如果执行了指令，则 Q4.0 为“1”。

## 14.5 WOR\_DW (字)双字或运算

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DWORD	I、Q、M、L、D	逻辑运算的第一个值
IN2	DWORD	I、Q、M、L、D	逻辑运算的第二个值
OUT	DWORD	I、Q、M、L、D	逻辑运算的结果双字

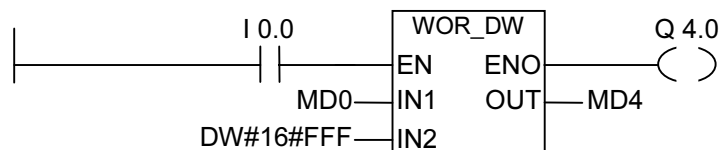
描述

使能(EN)输入的信号状态为“1”时将激活 **WOR\_DW** (双字或运算)，并逐位对 IN1 和 IN2 处的两个字值进行或运算。按纯位模式来解释这些值。可以在输出 OUT 扫描结果。ENO 与 EN 的逻辑状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	x	0	0	-	x	1	1	1

实例



如果 I0.0 为“1”，则执行指令。将位 0 至 11 设置为“1”，不改变 MD0 的其余位：

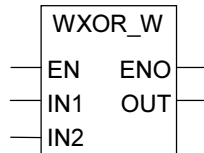
```

MD0          =      01010101 01010101 01010101 01010101
IN2          =      00000000 00000000 00001111 11111111
MD0 OR IN2 = MD4 =      01010101 01010101 01011111 11111111
  
```

如果执行了指令，则 Q4.0 为“1”。

## 14.6 WXOR\_W (字)单字异或运算

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	WORD	I、Q、M、L、D	逻辑运算的第一个值
IN2	WORD	I、Q、M、L、D	逻辑运算的第二个值
OUT	WORD	I、Q、M、L、D	逻辑运算的结果字

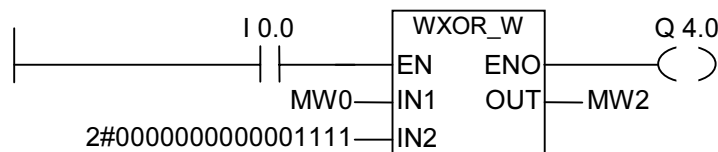
描述

使能(EN)输入的信号状态为“1”时将激活 **WXOR\_W** (单字异或运算), 并逐位对 IN1 和 IN2 处的两个字值进行异或运算。按纯位模式来解释这些值。可以在输出 OUT 扫描结果。ENO 与 EN 的逻辑状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	x	0	0	-	x	1	1	1

实例



如果 I0.0 为“1”，则执行指令：

MW0 = 01010101 01010101

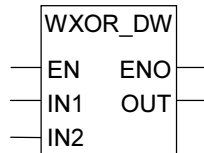
IN2 = 00000000 00001111

MW0 XOR IN2 = MW2 = 01010101 01011010

如果执行了指令，则 Q4.0 为“1”。

## 14.7 WXOR\_DW (字)双字异或运算

符号



参数	数据类型	存储区	描述
EN	BOOL	I、Q、M、L、D	启用输入
ENO	BOOL	I、Q、M、L、D	启用输出
IN1	DWORD	I、Q、M、L、D	逻辑运算的第一个值
IN2	DWORD	I、Q、M、L、D	逻辑运算的第二个值
OUT	DWORD	I、Q、M、L、D	逻辑运算的结果双字

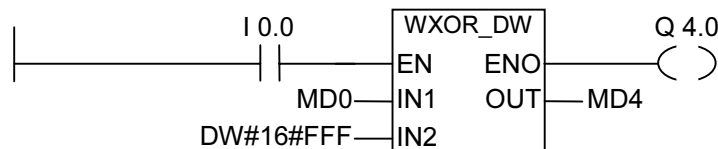
描述

使能(EN)输入的信号状态为“1”时将激活 **WXOR\_DW** (双字异或运算), 并逐位对 **IN1** 和 **IN2** 处的两个字值进行异或运算。按纯位模式来解释这些值。可以在输出 **OUT** 扫描结果。**ENO** 与 **EN** 的逻辑状态相同。

状态字

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	1	x	0	0	-	x	1	1	1

实例



如果 I0.0 为“1”，则执行指令：

MD0 = 01010101 01010101 01010101 01010101

IN2 = 00000000 00000000 00001111 11111111

MW2 = MD0 XOR IN2 = 01010101 01010101 01011010 10101010

如果执行了指令，则 Q4.0 为“1”。





# A 所有 LAD 指令总览

## A.1 按英语助记符(国际)排序的 LAD 指令

英语助记符	德语助记符	程序元素目录	描述
--- ---	--- ---	位逻辑指令	常开触点(地址)
---/ ---	---/ ---	位逻辑指令	常闭触点(地址)
---( )	---( )	位逻辑指令	输出线圈
---(#)--	---(#)--	位逻辑指令	中间输出
==0 --- ---	==0 --- ---	状态位	结果位等于 0
>0 --- ---	>0 --- ---	状态位	结果位大于 0
>=0 --- ---	>=0 --- ---	状态位	结果位大于等于 0
<=0 --- ---	<=0 --- ---	状态位	结果位小于等于 0
<0 --- ---	<0 --- ---	状态位	结果位小于 0
<>0 --- ---	<>0 --- ---	状态位	结果位不等于 0
ABS	ABS	浮点型指令	求浮点数的绝对值
ACOS	ACOS	浮点型指令	求反余弦值
ADD_DI	ADD_DI	整数数学运算指令	长整数加
ADD_I	ADD_I	整数数学运算指令	整数加
ADD_R	ADD_R	浮点型指令	实数加
ASIN	ASIN	浮点型指令	求反正弦值
ATAN	ATAN	浮点型指令	求反正切值
BCD_DI	BCD_DI	转换	BCD 码转换为长整型
BCD_I	BCD_I	转换	BCD 码转换为整型
BR --- ---	BIE --- ---	状态位	异常位二进制结果
----(CALL)	----(CALL)	程序控制	调用来自线圈的 FC SFC (不带参数)
CALL_FB	CALL_FB	程序控制	调用来自框的 FB
CALL_FC	CALL_FC	程序控制	调用来自框的 FC
CALL_SFB	CALL_SFB	程序控制	调用来自框的系统 FB
CALL_SFC	CALL_SFC	程序控制	调用来自框的系统 FC
----(CD)	----(ZR)	计数器	降值计数器线圈
CEIL	CEIL	转换	向上取整
CMP >=D	CMP >=D	比较	长整数比较 (==、<>、>、<、>=、<=)
CMP >=I	CMP >=I	比较	整数比较 (==、<>、>、<、>=、<=)
CMP >=R	CMP >=R	比较	实数比较 (==、<>、>、<、>=、<=)
COS	COS	浮点型指令	求余弦值

英语助记符	德语助记符	程序元素目录	描述
---(CU)	---( ZV )	计数器	升值计数器线圈
DI_BCD	DI_BCD	转换	长整型转换为 BCD 码
DI_R	DI_R	转换	长整型转换为浮点型
DIV_DI	DIV_DI	整数数学运算指令	长整数除
DIV_I	DIV_I	整数数学运算指令	整数除
DIV_R	DIV_R	浮点型指令	实数除
EXP	EXP	浮点型指令	求指数值
FLOOR	FLOOR	转换	向下取整
I_BCD	I_BCD	转换	整型转换为 BCD 码
I_DI	I_DI	转换	整型转换为长整型
INV_I	INV_I	转换	对整数求反码
INV_DI	INV_DI	转换	对长整数求反码
---(JMP)	---(JMP)	跳转	无条件跳转
---(JMP)	---(JMP)	跳转	条件跳转
---(JMPN)	---(JMPN)	跳转	若非则跳转
LABEL	LABEL	跳转	标号
LN	LN	浮点型指令	求自然对数
---(MCR>)	---(MCR>)	程序控制	主控制继电器关闭
---(MCR<)	---(MCR<)	程序控制	主控制继电器打开
---(MCRA)	---(MCRA)	程序控制	主控制继电器激活
---(MCRD)	---(MCRD)	程序控制	主控制继电器取消激活
MOD_DI	MOD_DI	整数数学运算指令	返回长整数余数
MOVE	MOVE	传送	分配值
MUL_DI	MUL_DI	整数数学运算指令	长整数乘法
MUL_I	MUL_I	整数数学运算指令	整数乘
MUL_R	MUL_R	浮点型指令	实数乘
---( N )---	---( N )---	位逻辑指令	RLO 负跳沿检测
NEG	NEG	位逻辑指令	地址下降沿检测
NEG_DI	NEG_DI	转换	对长整数求补码
NEG_I	NEG_I	转换	对整数求补码
NEG_R	NEG_R	转换	浮点数取反
---  NOT  ---	---  NOT  ---	位逻辑指令	能流取反
---(OPN)	---(OPN)	DB 调用	打开数据块: DB 或 DI
OS ---   ---	OS ---   ---	状态位	存储的异常位溢出
OV ---   ---	OV ---   ---	状态位	异常位溢出
---( P )---	---( P )---	位逻辑指令	RLO 上升沿检测
POS	POS	位逻辑指令	地址上升沿检测
---( R )	---( R )	位逻辑指令	重置线圈
---(RET)	---(RET)	程序控制	返回
ROL_DW	ROL_DW	移位/循环	双字循环左移
ROR_DW	ROR_DW	移位/循环	双字循环右移
ROUND	ROUND	转换	取整为长整型
RS	RS	位逻辑指令	置位优先型 RS 双稳态触发器
---( S )	---( S )	位逻辑指令	置位线圈
---( SAVE )	---( SAVE )	位逻辑指令	将 RLO 的状态保存到 BR
---( SC )	---( SZ )	计数器	设置计数器值

英语助记符	德语助记符	程序元素目录	描述
S_CD	Z_RUECK	计数器	减计数器
S_CU	Z_VORW	计数器	加计数器
S_CUD	ZAEHLER	计数器	双向计数器
---( SD )	---( SE )	定时器	接通延时定时器线圈
---( SE )	---( SV )	定时器	扩展脉冲定时器线圈
---( SF )	---( SA )	定时器	断开延时定时器线圈
SHL_DW	SHL_DW	移位/循环	双字左移
SHL_W	SHL_W	移位/循环	字左移
SHR_DI	SHR_DI	移位/循环	长整数右移
SHR_DW	SHR_DW	移位/循环	双字右移
SHR_I	SHR_I	移位/循环	整数右移
SHR_W	SHR_W	移位/循环	字右移
SIN	SIN	浮点型指令	求正弦值
S_ODT	S_EVERZ	定时器	接通延时 S5 定时器
S_ODTS	S_SEVERZ	定时器	保持接通延时 S5 定时器
S_OFFDT	S_AVERZ	定时器	断开延时 S5 定时器
---( SP )	---( SI )	定时器	脉冲定时器线圈
S_PEXT	S_VIMP	定时器	扩展脉冲 S5 定时器
S_PULSE	S_IMPULS	定时器	脉冲 S5 定时器
SQR	SQR	浮点型指令	求平方
SQRT	SQRT	浮点型指令	求平方根
SR	SR	位逻辑指令	复位优先型 SR 双稳态触发器
---( SS )	---( SS )	定时器	保持接通延时定时器线圈
SUB_DI	SUB_DI	整数数学运算指令	长整数减
SUB_I	SUB_I	整数数学运算指令	整数减
SUB_R	SUB_R	浮点型指令	实数减
TAN	TAN	浮点型指令	求正切值
TRUNC	TRUNC	转换	截取长整数部分
UO ---   ---	UO ---   ---	状态位	异常位无序
WAND_DW	WAND_DW	字逻辑指令	与运算双字
WAND_W	WAND_W	字逻辑指令	与运算字
WOR_DW	WOR_DW	字逻辑指令	或运算双字
WOR_W	WOR_W	字逻辑指令	或运算字
WXOR_DW	WXOR_DW	字逻辑指令	异或运算双字
WXOR_W	WXOR_W	字逻辑指令	异或运算字

## A.2 按德语助记符(SIMATIC)排序的 LAD 指令

德语助记符	英语助记符	程序元素目录	描述
---   ---	---   ---	位逻辑指令	常开触点(地址)
--- / ---	--- / ---	位逻辑指令	常闭触点(地址)
---( )	---( )	位逻辑指令	输出线圈
---(#)--	---(#)--	位逻辑指令	中间输出
==0 ---   ---	==0 ---   ---	状态位	结果位等于 0
>0 ---   ---	>0 ---   ---	状态位	结果位大于 0
>=0 ---   ---	>=0 ---   ---	状态位	结果位大于等于 0
<=0 ---   ---	<=0 ---   ---	状态位	结果位小于等于 0
<0 ---   ---	<0 ---   ---	状态位	结果位小于 0
<>0 ---   ---	<>0 ---   ---	状态位	结果位不等于 0
ABS	ABS	浮点型指令	求浮点数的绝对值
ACOS	ACOS	浮点型指令	求反余弦值
ADD_DI	ADD_DI	整数数学运算指令	长整数加
ADD_I	ADD_I	整数数学运算指令	整数加
ADD_R	ADD_R	浮点型指令	实数加
ASIN	ASIN	浮点型指令	求反正弦值
ATAN	ATAN	浮点型指令	求反正切值
BCD_DI	BCD_DI	转换	BCD 码转换为长整型
BCD_I	BCD_I	转换	BCD 码转换为整型
BIE ---   ---	BR ---   ---	状态位	异常位二进制结果
----(CALL)	----(CALL)	程序控制	调用来自线圈的 FC SFC (不带参数)
CALL_FB	CALL_FB	程序控制	调用来自框的 FB
CALL_FC	CALL_FC	程序控制	调用来自框的 FC
CALL_SFB	CALL_SFB	程序控制	调用来自框的系统 FB
CALL_SFC	CALL_SFC	程序控制	调用来自框的系统 FC
CEIL	CEIL	转换	向上取整
CMP >=D	CMP >=D	比较	长整数比较 (==、<>、>、<、>=、<=)
CMP >=I	CMP >=I	比较	整数比较 (==、<>、>、<、>=、<=)
CMP >=R	CMP >=R	比较	实数比较 (==、<>、>、<、>=、<=)
COS	COS	浮点型指令	求余弦值
DI_BCD	DI_BCD	转换	长整型转换为 BCD 码

德语助记符	英语助记符	程序元素目录	描述
DI_R	DI_R	转换	长整型转换为浮点型
DIV_DI	DIV_DI	整数数学运算指令	长整数除
DIV_I	DIV_I	整数数学运算指令	整数除
DIV_R	DIV_R	浮点型指令	实数除
EXP	EXP	浮点型指令	求指数值
FLOOR	FLOOR	转换	向下取整
I_BCD	I_BCD	转换	整型转换为 BCD 码
I_DI	I_DI	转换	整型转换为长整型
INV_I	INV_I	转换	对整数求反码
INV_DI	INV_DI	转换	对长整数求反码
---(JMP)	---(JMP)	跳转	条件跳转
---(JMP)	---(JMP)	跳转	无条件跳转
---(JMPN)	---(JMPN)	跳转	若非则跳转
LABEL	LABEL	跳转	标号
LN	LN	浮点型指令	求自然对数
---(MCR>)	---(MCR>)	程序控制	主控制继电器关闭
---(MCR<)	---(MCR<)	程序控制	主控制继电器打开
---(MCRA)	---(MCRA)	程序控制	主控制继电器激活
---(MCRD)	---(MCRD)	程序控制	主控制继电器取消激活
MOD_DI	MOD_DI	整数数学运算指令	返回长整数余数
MOVE	MOVE	传送	分配值
MUL_DI	MUL_DI	整数数学运算指令	长整数乘法
MUL_I	MUL_I	整数数学运算指令	整数乘
MUL_R	MUL_R	浮点型指令	实数乘
---( N )---	---( N )---	位逻辑指令	RLO 负跳沿检测
NEG	NEG	位逻辑指令	地址下降沿检测
NEG_DI	NEG_DI	转换	对长整数求补码
NEG_I	NEG_I	转换	对整数求补码
NEG_R	NEG_R	转换	浮点数取反
---  NOT  ---	---  NOT  ---	位逻辑指令	能流取反
---(OPN)	---(OPN)	DB 调用	打开数据块: DB 或 DI
OS --- ---	OS --- ---	状态位	存储的异常位溢出
OV --- ---	OV --- ---	状态位	异常位溢出
---( P )---	---( P )---	位逻辑指令	RLO 上升沿检测

德语助记符	英语助记符	程序元素目录	描述
POS	POS	位逻辑指令	地址上升沿检测
---( R )	---( R )	位逻辑指令	重置线圈
---(RET)	---(RET)	程序控制	返回
ROL_DW	ROL_DW	移位/循环	双字循环左移
ROR_DW	ROR_DW	移位/循环	双字循环右移
ROUND	ROUND	转换	取整为长整型
RS	RS	位逻辑指令	置位优先型 RS 双稳态触发器
---( S )	---( S )	位逻辑指令	置位线圈
---( SA )	---( SF )	定时器	断开延时定时器线圈
---( SAVE )	---( SAVE )	位逻辑指令	将 RLO 的状态保存到 BR
S_AVERZ	S_OFFDT	定时器	断开延时 S5 定时器
---( SE )	---( SD )	定时器	接通延时定时器线圈
S_EVERZ	S_ODT	定时器	接通延时 S5 定时器
SHL_DW	SHL_DW	移位/循环	双字左移
SHL_W	SHL_W	移位/循环	字左移
SHR_DI	SHR_DI	移位/循环	长整数右移
SHR_DW	SHR_DW	移位/循环	双字右移
SHR_I	SHR_I	移位/循环	整数右移
SHR_W	SHR_W	移位/循环	字右移
---( SI )	---( SP )	定时器	脉冲定时器线圈
S_IMPULS	S_PULSE	定时器	脉冲 S5 定时器
SIN	SIN	浮点型指令	求正弦值
SQR	SQR	浮点型指令	求平方
SQRT	SQRT	浮点型指令	求平方根
SR	SR	位逻辑指令	复位优先型 SR 双稳态触发器
---( SS )	---( SS )	定时器	保持接通延时定时器线圈
S_SEVERZ	S_ODTS	定时器	保持接通延时 S5 定时器
SUB_DI	SUB_DI	整数数学运算指令	长整数减
SUB_I	SUB_I	整数数学运算指令	整数减
SUB_R	SUB_R	浮点型指令	实数减
---( SV )	---( SE )	定时器	扩展脉冲定时器线圈
S_VIMP	S_PEXT	定时器	扩展脉冲 S5 定时器
---( SZ )	---( SC )	计数器	设置计数器值
TAN	TAN	浮点型指令	求正切值
TRUNC	TRUNC	转换	截取长整数部分
UO ---   ---	UO ---   ---	状态位	异常位无序

德语助记符	英语助记符	程序元素目录	描述
WAND_DW	WAND_DW	字逻辑指令	与运算双字
WAND_W	WAND_W	字逻辑指令	与运算字
WOR_DW	WOR_DW	字逻辑指令	或运算双字
WOR_W	WOR_W	字逻辑指令	或运算字
WXOR_DW	WXOR_DW	字逻辑指令	异或运算双字
WXOR_W	WXOR_W	字逻辑指令	异或运算字
ZAEHLER	S_CUD	计数器	双向计数器
----(ZR)	----(CD)	计数器	降值计数器线圈
Z_RUECK	S_CD	计数器	减计数器
---(ZV)	----(CU)	计数器	升值计数器线圈
Z_VORW	S_CU	计数器	加计数器





## B 编程实例

### B.1 编程实例概述

#### 实际应用

本手册中描述的每个梯形图指令都会触发一个特定操作。将这些指令组合到一个程序中时，便可完成多种自动化任务。本章提供梯形图指令实际应用的以下实例：

- 控制传送带 - 使用位逻辑指令
- 检测传送带的移动方向 - 使用位逻辑指令
- 生成时钟脉冲 - 使用定时器指令
- 跟踪存储空间 - 使用计数器和比较指令
- 使用整数数学运算指令解决问题
- 设置加热烘炉的时间长度

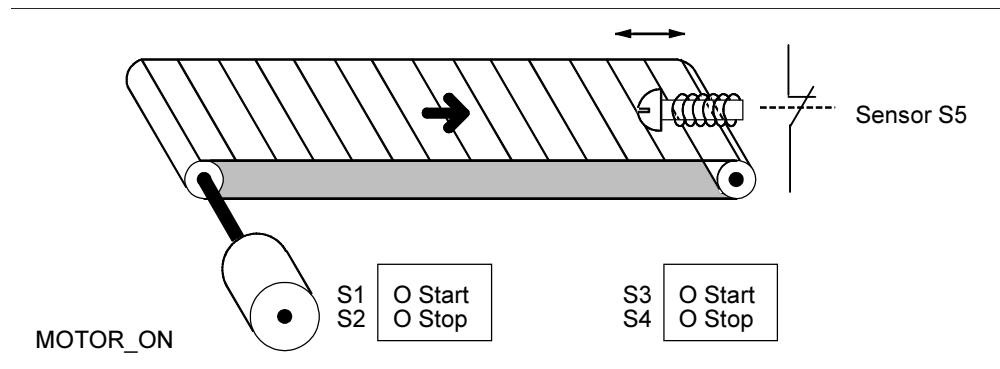
#### 使用的指令

助记符	程序元素目录	描述
WAND_W	字逻辑指令	(字)与运算
WOR_W	字逻辑指令	(字)或运算
---( CD )	计数器	降值计数器线圈
---( CU )	计数器	升值计数器线圈
---( R )	位逻辑指令	重置线圈
---( S )	位逻辑指令	置位线圈
---( P )	位逻辑指令	RLO 上升沿检测
ADD_I	浮点指令	整数加
DIV_I	浮点指令	整数除
MUL_I	浮点指令	整数乘
CMP <=I, CMP >=I	比较	比较整数
- -    - -	位逻辑指令	常开触点
- -   /   - -	位逻辑指令	常闭触点
- - ( )	位逻辑指令	输出线圈
---(JMPN)	跳转	若非则跳转
---(RET)	程序控制	返回
MOVE	传送	分配值
---( SE )	定时器	扩展脉冲定时器线圈

## B.2 实例：位逻辑指令

### 实例 1：控制传送带

下图显示可用电动方式激活的传送带。在传送带的开始位置有两个按钮开关：用于启动的 S1 和用于停止的 S2。在传送带末端也有两个按钮开关：用于启动的 S3 和用于停止的 S4。可从任何一端启动或停止传送带。此外，当传送带上的部件到达终点时，传感器 S5 将停止传送带。



### 绝对地址和符号编程

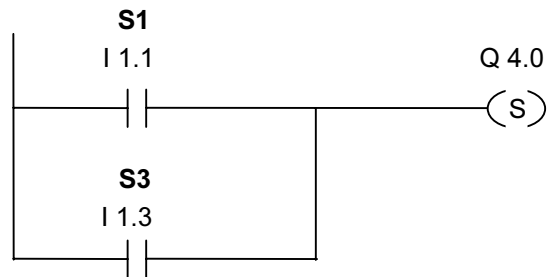
您可编写程序使用**绝对地址**或代表传送带系统各种组件的**符号**来控制传送带。

需要制定一个符号表，以建立所选择的符号与绝对地址的联系(参见 STEP 7 在线帮助)。

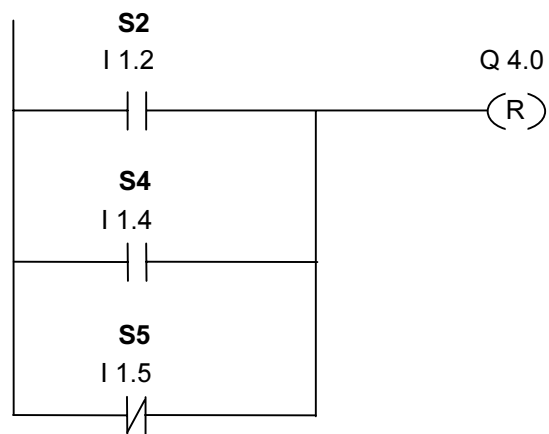
系统组件	绝对地址	符号	符号表
按钮启动开关	I 1.1	S1	I 1.1 S1
按钮停止开关	I 1.2	S2	I 1.2 S2
按钮启动开关	I 1.3	S3	I 1.3 S3
按钮停止开关	I 1.4	S4	I 1.4 S4
传感器	I 1.5	S5	I 1.5 S5
电机	Q 4.0	MOTOR_ON	Q 4.0 MOTOR_ON

## 控制传送带的梯形图程序

程序段 1: 按下任一启动开关打开电机。

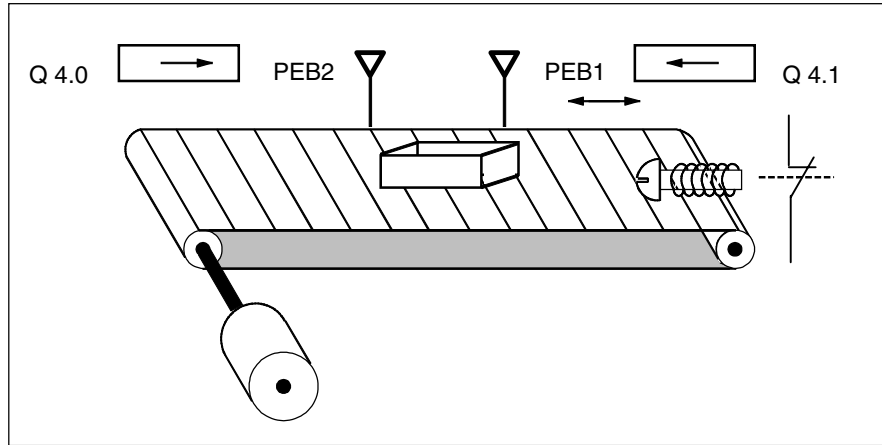


程序段 2: 按下任一停止开关或打开传送带尾部的常闭触点以关闭电机。



## 实例 2: 检测传送带方向

下图显示配备两个光电屏障(PEB1 和 PEB2)的传送带, 这两个光电屏障专用于检测包裹在传送带上移动的方向。每个光电屏障的功能类似常开触点。



## 绝对地址和符号编程

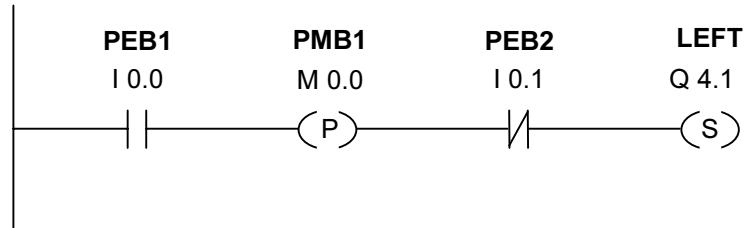
您可编写程序以使用**绝对地址**或代表传送带系统各种组件的**符号**来激活传送带系统的方向显示。

需要制定一个符号表, 以建立所选择的符号与绝对地址的联系(参见 STEP 7 在线帮助)。

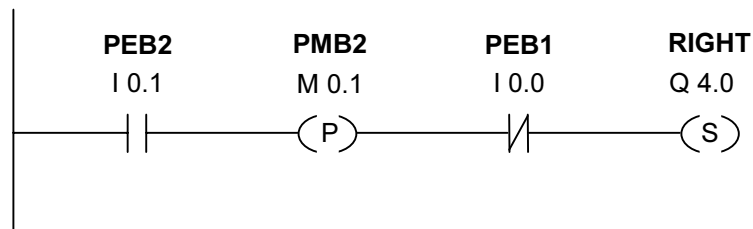
系统组件	绝对地址	符号	符号表
光电屏障 1	I 0.0	PEB1	I 0.0 PEB1
光电屏障 2	I 0.1	PEB2	I 0.1 PEB2
显示向右移动	Q 4.0	RIGHT	Q 4.0 RIGHT
显示向左移动	Q 4.1	LEFT	Q 4.1 LEFT
脉冲存储器位 1	M 0.0	PMB1	M 0.0 PMB1
脉冲存储器位 2	M 0.1	PMB2	M 0.1 PMB2

### 用于检测传送带方向的梯形图程序

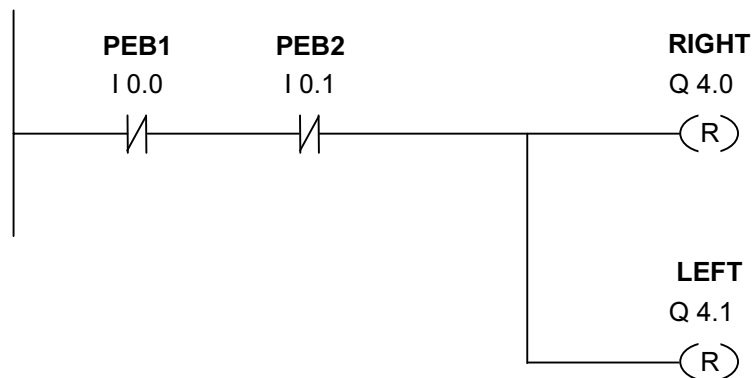
程序段 1: 如果输入 I 0.0 处信号状态从 0 过渡到 1(上升沿), 与此同时, 输入 I 0.1 处信号状态为 0, 则传送带上的包裹向左移动。



程序段 2: 如果输入 I 0.1 处信号状态从 0 过渡到 1(上升沿), 与此同时, 输入 I 0.0 处信号状态为 0, 则传送带上的包裹向右移动。如果光电屏障之一被中断, 则表明屏障之间有包裹。



程序段 3: 如果两个光电屏障都未中断, 则表明屏障之间没有包裹。方向指针关闭。



## B.3 实例：定时器指令

### 时钟脉冲发生器

当需要生成定期重复的信号时，可使用时钟脉冲发生器或闪烁继电器。时钟脉冲发生器在控制指示灯闪烁的信号系统中很常见。

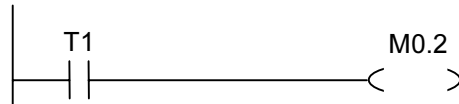
当使用 S7-300 时，您可用特殊组织块中的时间处理功能来执行时钟脉冲发生器功能。但下列梯形图程序中显示的实例说明的是使用定时器功能产生时钟脉冲。实例程序显示如何通过使用定时器实现任意的时钟脉冲发生器。

### 产生时钟脉冲(脉冲占空比 1:1)的梯形图程序

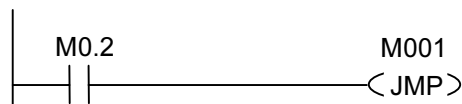
程序段 1：如果定时器 T1 的信号状态为 0，将时间值 250 毫秒载入 T1，并将 T1 作为扩展脉冲定时器启动。



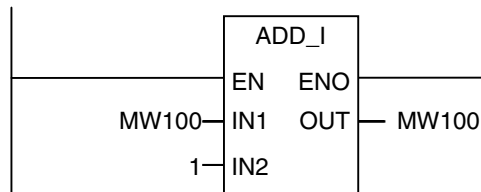
程序段 2：该定时器的状态临时保存在一个辅助存储器符号中。



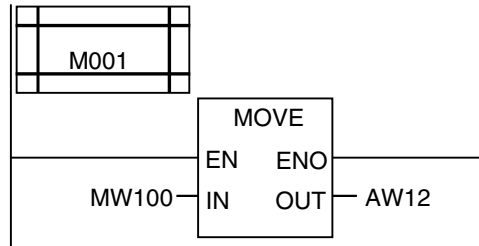
程序段 3：如果定时器 T1 的信号状态为 1，则跳转至跳转标签 M001。



程序段 4：定时器 T1 超时后，存储器字 100 增加 1。

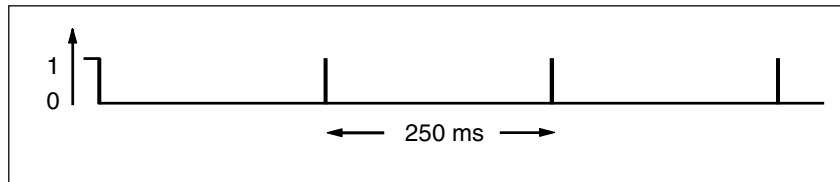


程序段 5: **MOVE** 指令允许在输出 Q12.0 到 Q13.7 输出不同的时钟频率。



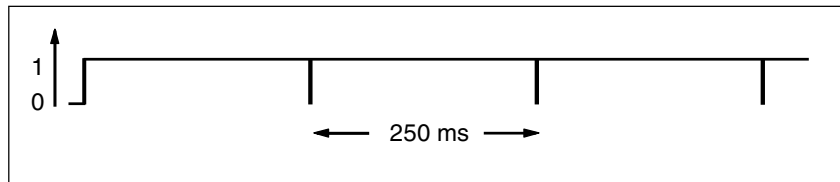
### 信号检查

定时器 T1 的信号检查为 opener M0.2 生成以下逻辑运算(RLO)结果。



一旦定时时间到，就会重新启动定时器。因此，由  $\text{---|/|---}$  M0.2 进行的信号检查只简单产生信号状态 1。

RLO 取反(反向):



每隔 250 毫秒 RLO 位为 0。忽略跳转且存储器字 MW100 的内容增加 1。

## 实现特定频率

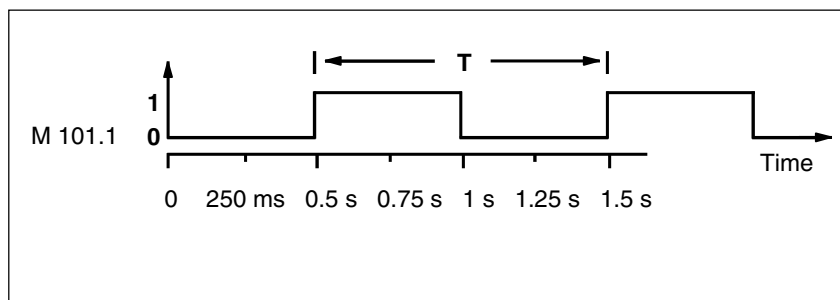
通过存储器字节 MB101 和 MB100 的单个位，可以实现下列频率：

MB101/MB100 的位	频率(赫兹)	持续时间
M 101.0	2.0	0.5 秒 (250 毫秒开/ 250 毫秒关)
M 101.1	1.0	1 秒 (0.5 秒开/ 0.5 秒关)
M 101.2	0.5	2 秒 (1 秒开/ 1 秒关)
M 101.3	0.25	4 秒 (2 秒开/ 2 秒关)
M 101.4	0.125	8 秒 (4 秒开/ 4 秒关)
M 101.5	0.0625	16 秒 (8 秒开/ 8 秒关)
M 101.6	0.03125	32 秒 (16 秒开/ 16 秒关)
M 101.7	0.015625	64 秒 (32 秒开/ 32 秒关)
M 100.0	0.0078125	128 秒 (64 秒开/ 64 秒关)
M 100.1	0.0039062	256 秒 (128 秒开/ 128 秒关)
M 100.2	0.0019531	512 秒 (256 秒开/ 256 秒关)
M 100.3	0.0009765	1024 秒 (512 秒开/ 512 秒关)
M 100.4	0.0004882	2048 秒 (1024 秒开/ 1024 秒关)
M 100.5	0.0002441	4096 秒 (2048 秒开/ 2048 秒关)
M 100.6	0.000122	8192 秒 (4096 秒开/ 4096 秒关)
M 100.7	0.000061	16384 秒 (8192 秒开/ 8192 秒关)

## 存储器 MB 101 的位信号状态

扫描周期	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位	第 0 位	时间值 (单位: 毫秒)
0	0	0	0	0	0	0	<b>0</b>	0	250
1	0	0	0	0	0	0	<b>0</b>	1	250
2	0	0	0	0	0	0	1	0	250
3	0	0	0	0	0	0	1	1	250
4	0	0	0	0	0	1	<b>0</b>	0	250
5	0	0	0	0	0	1	<b>0</b>	1	250
6	0	0	0	0	0	1	1	0	250
7	0	0	0	0	0	1	1	1	250
8	0	0	0	0	1	0	<b>0</b>	0	250
9	0	0	0	0	1	0	<b>0</b>	1	250
10	0	0	0	0	1	0	1	0	250
11	0	0	0	0	1	0	1	1	250
12	0	0	0	0	1	1	<b>0</b>	0	250

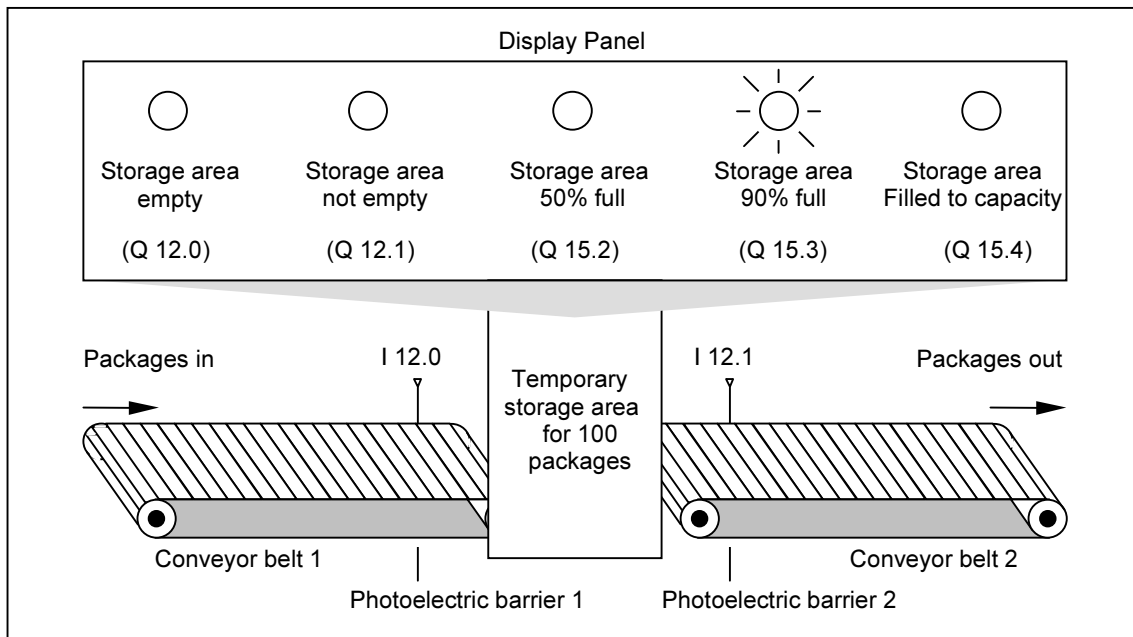


**MB 101 (M 101.1)第 1 位的信号状态**频率 =  $1/T = 1/1\text{ s} = 1$  赫兹

## B.4 实例：计数器和比较指令

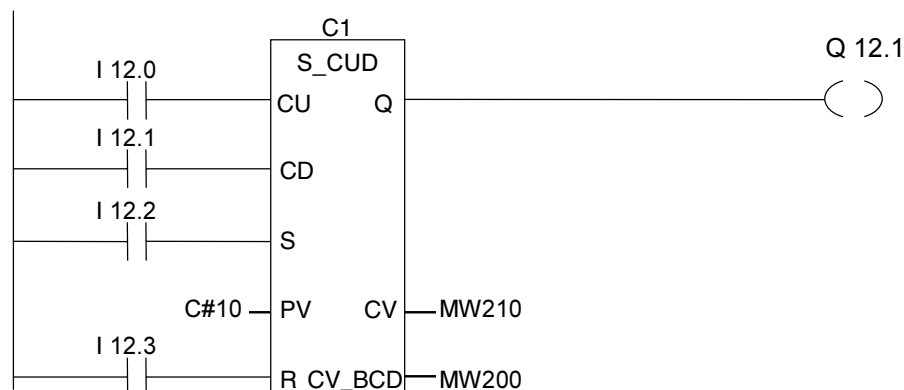
### 带计数器和比较器的存储区域

下图显示了具有两个传送带且在传送带之间有临时存储区域的系统。传送带 1 将包裹传送到存储区域。存储区域附近的传送带 1 末端的光电屏障确定向存储区域传送的包裹数量。传送带 2 会将包裹从临时存储区域传输到装载码头，而卡车在此将包裹发送给客户。存储区域附近传送带 2 末端的光电屏障确定离开存储区域而转向装载码头的包裹数量。带五个指示灯的显示面板将指示临时存储区域的填充量。



### 激活显示面板上的指示灯的梯形图程序

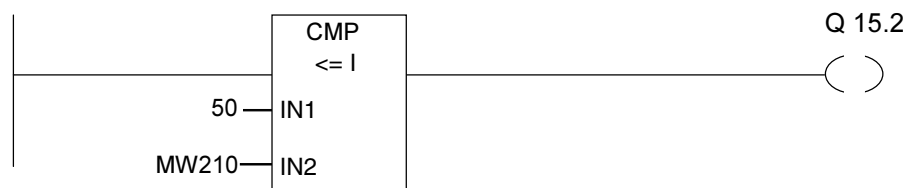
程序段 1: 计数器 C1 对输入 CU 处每次从“0”到“1”的信号改变都进行正计数, 而对输入 CD 处每次从“0”到“1”的信号改变都进行倒计数。对于输入 S 处从“0”到“1”的信号改变, 计数器值被设置为值 PV。输入 R 处从“0”到“1”的信号改变将计数器值复位为“0”。MW200 包含 C1 的当前计数器值。Q12.1 指示“存储区域非空”。



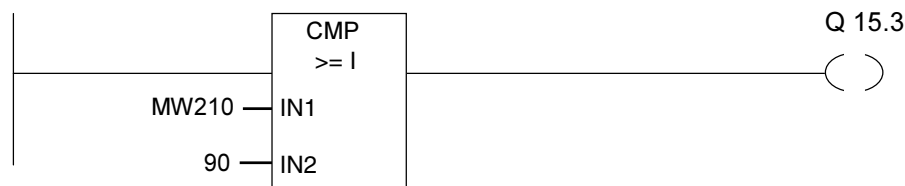
程序段 2: Q12.0 表明“存储区域为空”。



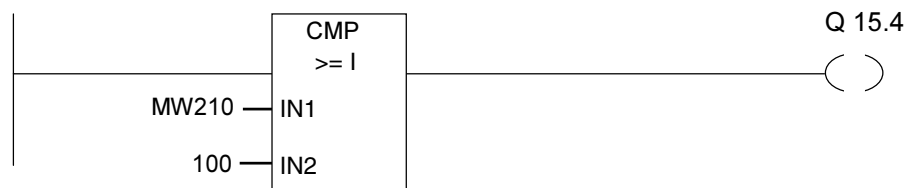
程序段 3: 如果 50 小于等于计数器值(换句话说, 如果当前计数器值大于等于 50), 则表示“存储区域 50%满”的指示灯变亮。



程序段 4: 程序段 4: 如果计数器值大于或等于 90, 则表示“存储区域 90%满”的指示灯变亮。



程序段 5: 如果计数器值大于或等于 100, 则表示“存储区域满”的指示灯变亮。



## B.5 实例：整型数学运算指令

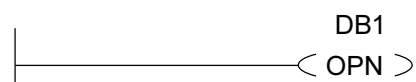
### 解决数学问题

实例程序显示了如何使用三个整数数学运算指令来产生与下列方程式相同的结果：

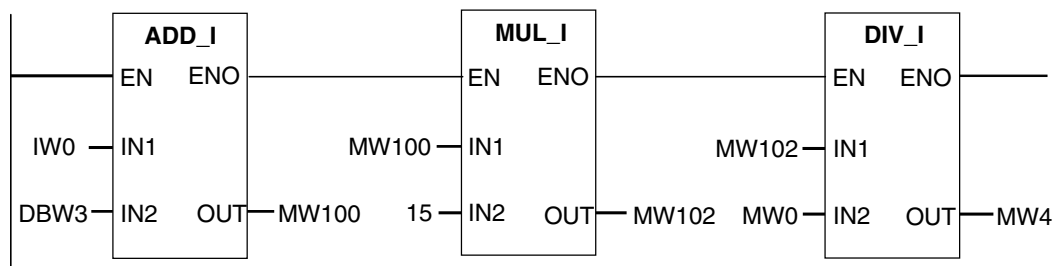
$$MW4 = ((IW0 + DBW3) \times 15) / MW0$$

### 梯形图程序

程序段 1：打开数据块 DB1。



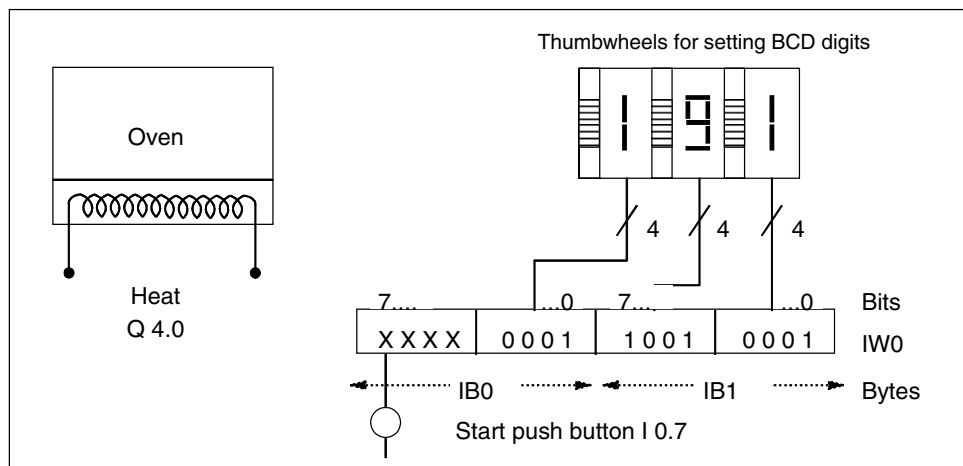
程序段 2：输入字 IW0 加到共享数据字 DBW3(必须定义和打开数据块)，总和被载入存储器字 MW100。然后，MW100 乘以 15，结果存储到存储器字 MW102 中。MW102 除以 MW0，结果存储到 MW4 中。



## B.6 实例：字逻辑指令

### 加热烘炉

烘炉操作员通过按启动按钮来启动烘炉加热。操作员可用图中所示的码盘开关来设置加热的时间。操作员设置的值以二进制编码的十进制(BCD)格式显示，单位为秒。



系统组件	绝对地址
启动按钮	I 0.7
个位码盘	I 1.0 到 I 1.3
十位指轮开关	I 1.4 到 I 1.7
百位码盘	I 0.0 到 I 0.3
加热启动	Q 4.0

## 梯形图程序

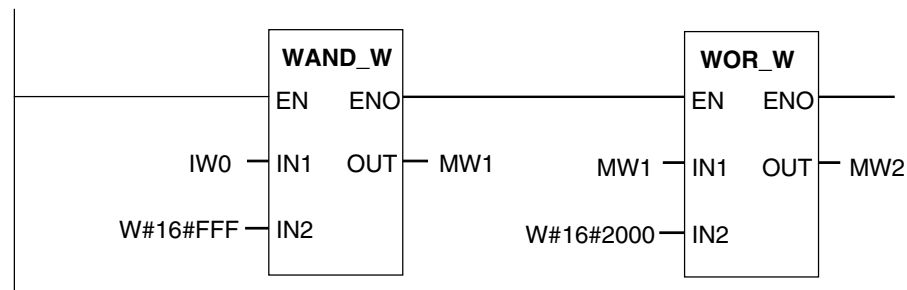
程序段 1: 如果定时器正在运行, 则打开加热器。



程序段 2: 如果定时器正在运行, 返回指令结束此处的处理。



程序段 3: 屏蔽输入位 I 0.4 到 I 0.7 (即, 将它们复位为 0)。指轮开关输入的这些位未被使用。16 位指轮开关输入根据(字)与运算指令与 W#16#0FFF 组合。结果载入存储器字 MW1 中。为了设置时间基准的秒数, 预设值根据(字)或运算指令与 W#16#2000 组合, 将位 13 设置为 1, 并将位 12 复位为 0。



程序段 4: 如果按下启动按钮, 则将定时器 T1 作为扩展脉冲定时器启动, 并作为预设值存储器字 MW2 装载(来自于上述逻辑)。







## C 使用梯形图

### C.1 EN/ENO 机制

FBD/LAD 框的启用(EN)和启用输出(ENO)通过 BR 位来获取。

如果连接了 EN 和 ENO，则以下规则适用：

#### **ENO = EN AND NOT (框错误)**

当没有发生错误(框错误 = 0)时， $ENO = EN$ 。

EN/ENO 机制用于：

- 数学运算指令、
- 传送和转换指令、
- 移位和循环移位指令、
- 块调用。

该机制不用于：

- 比较、
- 计数器、
- 定时器。

在框的实际指令周围，为 EN/ENO 机制生成附加的 STL 指令，这些指令依赖于现有的在此之前和之后的逻辑运算。使用一个加法器实例，显示下列四种可能的情况：

1. 加法器连接了 EN 和 ENO
2. 加法器连接了 EN 但未连接 ENO
3. 加法器未连接 EN 但连接了 ENO
4. 加法器未连接 EN 和 ENO

## 创建块的注意事项

如果要编程在 **FBD** 或 **LAD** 中调用的块，那么必须确保退出块时，置位 **BR** 位。第四个实例显示这种结果并不会自动出现。不能将 **BR** 作为存储位，因为 **EN/ENO** 机制不断改写 **BR**。作为代替，可使用一个临时变量来保存发生的所有错误。用 **0** 初始化该变量。在块中任何一个您认为指令不成功即表示整个块出错的地方，借助 **EN/ENO** 机制来设置该变量。一个 **NOT** 和一个 **SET** 线圈足以完成这项工作。在块结束处，编程下列程序段：

```
end:   AN error
      SAVE
```

确保在任何情况下都处理本程序段，这表示禁止在块内使用 **BEC**，并禁止跳过本程序段。

### C.1.1 连接了 EN 和 ENO 的加法器

当加法器连接了 EN 和 ENO 时，触发下列 STL 指令：

```

1  A   I   0.0           //EN 连接
2  JNB _001             //在 RLO = 0 时，将 RLO 移位到 BR 中并跳转
3  L   in1              //框参数
4  L   in2              //框参数
5  +I                   //实际加法
6  T   out              //框参数
7  AN   OV              //错误识别
8  SAVE                 //在 BR 中保存错误
9  CLR                  //第一次校验
10 _001: A   BR         //将 BR 移位到 RLO 中
11 =   Q   4.0

```

在第 1 行后，RLO 包含在此之前的逻辑运算的结果。JNB 指令将 RLO 复制到 BR 位，并设置第一个校验位。

- 当 RLO = 0 时，程序跳转到第 10 行，继续执行 A BR。不执行加法指令。在第 10 行，重新将 BR 复制到 RLO 中，然后给输出赋值 0。
- 当 RLO = 1 时，程序不跳转，表示执行加法指令。在第 7 行中，程序判断是否在执行加法指令期间发生了错误，然后在第 8 行存储到 BR 中。第 9 行设置第一个校验位。现在，在第 10 行中将 BR 位复制回 RLO，因此，输出显示是否成功执行了加法指令。  
第 10 行和第 11 行不改变 BR 位，因此，也显示是否成功执行了加法指令。

### C.1.2 连接 EN 但未连接 ENO 的加法器

当加法器连接了 EN 但未连接 ENO 时，触发下列 STL 指令：

```
1 A      I      0.0      //EN 连接
2 JNB   _001          //在 RLO = 0 时，将 RLO 移位到 BR 中并跳转
3 L     in1          //框参数
4 L     in2          //框参数
5 +I                    //实际加法
6 T     out          //框参数
7 _001: NOP  0
```

在第 1 行后，RLO 包含在此之前的逻辑运算的结果。JNB 指令将 RLO 复制到 BR 位，并设置第一个校验位。

- 当 RLO = 0 时，程序跳转到第 7 行，不执行加法指令。RLO 和 BR 都为 0。
- 当 RLO 为 1 时，程序不跳转，表示执行加法指令。程序不判断在执行加法指令期间是否发生错误。RLO 和 BR 都为 1。

### C.1.3 连接 EN 但未连接 ENO 的加法器

当加法器没有连接 EN，但连接了 ENO 时，触发下列 STL 指令：

```
1 L    in1           //框参数
2 L    in2           //框参数
3 +I                    //实际加法
4 T    out           //框参数
5 AN   OV            //错误识别
6  SAVE                //在 BR 中保存错误
7 CLR                    //第一次校验
8  A    BR            //将 BR 移位到 RLO 中
9  =   Q    4.0
```

在每种情况下都执行加法指令。在第 5 行中，程序判断在执行加法指令期间是否发生了错误，然后在第 6 行存储到 BR 中。第 7 行设置第一个校验位。现在，在第 8 行中将 BR 位复制回 RLO，因此，输出显示是否成功执行了加法指令。

第 8 行和第 9 行不改变 BR 位，因此，也显示是否成功执行了加法指令。

#### C.1.4 没有连接 EN 和 ENO 的加法器

当加法器没有连接 EN 和 ENO 时，触发下列 STL 指令：

```
1 L    in1          //框参数
2 L    in2          //框参数
3 +I                    //实际加法
4 T    out          //框参数
5 NOP 0
```

执行加法指令。RLO 和 BR 位保持不变。

## C.2 参数传送

块的参数作为数值传送。对于功能块，在被调用块中使用实例数据块中的实际参数值副本。对于功能，实际值副本位于本地数据堆栈内。不复制指针。调用前，将 INPUT 数值复制到实例 DB 或 L 堆栈中。调用后，将 OUTPUT 数值复制回变量中。在被调用块中，只能使用副本。为此要求使用的 STL 指令位于调用块中，用户不可见。

---

### 注意

如果存储位、输入、输出或外围设备 I/O 作为功能的实际地址使用，那么它们以与其它地址不同的方式进行处理。在此，直接执行更新，而不是通过 L 堆栈执行更新。

### 例外：

如果相应的形式参数为 BOOL 数据类型的一个输入参数，那么通过 L 堆栈更新当前参数。

---

### 当心

在编程调用块时，请确保写入声明为 OUTPUT 的参数。否则，输出值为随机值！对于功能块，该值为由最后一个调用指定的实例 DB 中的数值；对于功能，该值为位于 L 堆栈的数值。

注意下列各点：

- 如有可能，初始化所有 OUTPUT 参数。
  - 尽量不要使用任何置位和复位指令。这些指令取决于 RLO。当 RLO 的数值为 0 时，将保持随机值。
  - 如果在块内跳转，那么确保没有跳过写入 OUTPUT 参数的位置。不要忘记 BEC 和 MCR 指令的影响。
-





# 索引

字母	
--( )	1-6
--( CD )	4-12
--( CU )	4-10
--( N )--	1-18
--( P )--	1-19
--( R )	1-10
--( S )	1-12
--( SA )	13-23
--( SAVE )	1-20
--( SC )	4-9
--( SD )	13-19
--( SE )	13-17、13-19
--( SF )	13-23
--( SI )	13-15
--( SP )	13-15
--( SS )	13-21
--( SV )	13-17
--( SZ )	4-9
--( ZR )	4-12
--( ZV )	4-10
--(#)--	1-8
--(Call)	10-2
--(JMP)--	6-2、6-3
--(JMPN)	6-4
--(MCR<)	10-14
--(MCR>)	10-16、10-17
--(MCRA)	10-18
--(MCRD)	10-19
--(OPN)	5-1
--(RET)	10-20
(字)单字或运算	14-3
(字)单字异或运算	14-6
(字)单字与运算	14-2
(字)双字或运算	14-5
(字)双字异或运算	14-7
(字)双字与运算	14-4
--   --	12-1
--   --	1-2
--  /  --	1-3、12-1
-- NOT --	1-5
<=0 --   --	12-12
<=0 --  /  --	12-12
<>0 --   --	12-8
<>0 --  /  --	12-8
<0 --   --	12-10
<0 --  /  --	12-10
=0 --   --	12-7
=0 --  /  --	12-7
>0 --   --	12-11
>0 --  /  --	12-11
>0 --   --	12-9
>0 --  /  --	12-9
ABS	8-8
ACOS	8-17
ADD_DI	7-7
ADD_I	7-3
ADD_R	8-4
ASIN	8-16
ATAN	8-18
BCD_DI	3-5
BCD_I	3-2
BCD 码转换为长整型	3-5
BCD 码转换为整型	3-2
BR --   --	12-6
BR --  /  --	12-6
CALL_FB	10-4
CALL_FC	10-6
CALL_SFB	10-8
CALL_SFC	10-10
CEIL	3-15
CMP ? D	2-4
CMP ? I	2-2
CMP ? R	2-6
COS	8-14
DI_BCD	3-6
DI_REAL	3-7
DIV_DI	7-10
DIV_I	7-6
DIV_R	8-7
EN/ENO 机制	C-1、C-2
EXP	8-11
FLOOR	3-16
I_BCD	3-3
I_DINT	3-4
INV_DI	3-9
INV_I	3-8
LABEL	6-5
LN	8-12
MOD_DI	7-11
MOVE	9-1、9-2
MUL_DI	7-9
MUL_I	7-5
MUL_R	8-6
NEG	1-21

NEG_DI .....	3-11
NEG_I .....	3-10
NEG_R .....	3-12
OS ---   --- .....	12-3
OS ---  / --- .....	12-3
OV ---   --- .....	12-2
OV ---  / --- .....	12-2
POS .....	1-22
RLO 负跳沿检测 .....	1-18
RLO 上升沿检测 .....	1-19
ROL_DW .....	11-12
ROR_DW .....	11-13、11-14
ROUND .....	3-13
RS .....	1-14
S_AVERZ .....	13-13
S_CD .....	4-7
S_CU .....	4-5
S_CUD .....	4-3
S_EVERZ .....	13-9
S_IMPULS .....	13-5
S_ODT .....	13-9
S_ODTS .....	13-11
S_OFFDT .....	13-13
S_PEXT .....	13-7
S_PULSE .....	13-5
S_SEVERZ .....	13-11
S_VIMP .....	13-7
SHL_DW .....	11-8
SHL_W .....	11-5、11-6
SHR_DI .....	11-4
SHR_DW .....	11-9、11-10
SHR_I .....	11-2、11-3
SHR_W .....	11-7
SIN .....	8-13
SQR .....	8-9
SQRT .....	8-10
SR .....	1-16
SUB_DI .....	7-8
SUB_I .....	7-4
SUB_R .....	8-5
TAN .....	8-15
TRUNC .....	3-14
UO ---   --- .....	12-5
UO ---  / --- .....	12-5
WAND_DW .....	14-4
WAND_W .....	14-2
WOR_DW .....	14-5
WOR_W .....	14-3
WXOR_DW .....	14-7
WXOR_W .....	14-6
XOR .....	1-4
Z_RUECK .....	4-7
Z_VORW .....	4-5

**A**

按德语助记符(SIMATIC)排序的

LAD 指令 .....A-4

按英语助记符(国际)排序的

LAD 指令 .....A-1

**B**

保持接通延时 S5 定时器 ..... 13-11

保持接通延时定时器线圈 ..... 13-21

比较指令概述 ..... 2-1

编程实例概述 ..... B-1

标号 ..... 6-5

**C**

参数传送 ..... C-7

长整数乘法 ..... 7-9

长整数除 ..... 7-10

长整数加 ..... 7-7

长整数减 ..... 7-8

长整数右移 ..... 11-4

长整型转换为 BCD 码 ..... 3-6

长整型转换为浮点型 ..... 3-7

常闭触点(地址) ..... 1-3

常开触点(地址) ..... 1-2

程序控制指令概述 ..... 10-1

从库中调用块 ..... 10-13

存储的异常位溢出 ..... 12-3

存储的异常位溢出取反 ..... 12-3

**D**

打开数据块

DB 或 DI ..... 5-1

地址上升沿检测 ..... 1-22

地址下降沿检测 ..... 1-21

调用多重实例 ..... 10-12

调用来自框的 FB ..... 10-4

调用来自框的 FC ..... 10-6

调用来自框的系统 FB ..... 10-8

调用来自框的系统 FC ..... 10-10

调用来自线圈的 FC SFC (不带参数) ..... 10-2

定时器在存储器中的位置与

定时器组件 ..... 13-2

定时器指令概述 ..... 13-1

断开延时 S5 定时器 ..... 13-13

断开延时定时器线圈 ..... 13-23

对长整数求补码 ..... 3-11

对长整数求反码 ..... 3-9

对整数求补码 ..... 3-10

对整数求反码 ..... 3-8

- F**
- 返回 ..... 10-20
  - 返回长整数余数 ..... 7-11
  - 分配值 ..... 9-1
  - 浮点数取反 ..... 3-12
  - 浮点运算指令 ..... 8-2
  - 浮点运算指令概述 ..... 8-1
  - 复位优先型 SR 双稳态触发器 ..... 1-16
- J**
- 计数器指令概述 ..... 4-1
  - 加计数器 ..... 4-5
  - 加法器连接了 EN 但未连接 ENO ..... C-4
  - 加法器连接了 EN 和 ENO ..... C-3
  - 加法器未连接 EN 但连接了 ENO ..... C-5
  - 加法器未连接 EN 和 ENO ..... C-6
  - 减计数器 ..... 4-7
  - 将 RLO 的状态保存到 BR ..... 1-20
  - 降值计数器线圈 ..... 4-12
  - 接通延时 S5 定时器 ..... 13-9
  - 接通延时定时器线圈 ..... 13-19
  - 结果位不等于 0 ..... 12-8
  - 结果位大于 0 ..... 12-9
  - 结果位大于等于 0 ..... 12-11
  - 结果位等于 0 ..... 12-7
  - 结果位取反后不等于 0 ..... 12-8
  - 结果位取反后大于 0 ..... 12-9
  - 结果位取反后大于等于 0 ..... 12-11
  - 结果位取反后等于 0 ..... 12-7
  - 结果位取反后小于 0 ..... 12-10
  - 结果位取反后小于等于 0 ..... 12-12
  - 结果位小于 0 ..... 12-10
  - 结果位小于等于 0 ..... 12-12
  - 截取长整数部分 ..... 3-14
- K**
- 扩展脉冲 S5 定时器 ..... 13-7
  - 扩展脉冲定时器线圈 ..... 13-17
- L**
- 立即读取 ..... 1-23
  - 立即写入 ..... 1-24、1-25
  - 逻辑 ..... 1-4
  - 逻辑控制指令概述 ..... 6-1
- M**
- 脉冲 S5 定时器 ..... 13-5
  - 脉冲定时器线圈 ..... 13-15
- N**
- 能流取反 ..... 1-5
- Q**
- 求反余弦值 ..... 8-17
  - 求反正切值 ..... 8-18
  - 求反正弦值 ..... 8-16
  - 求浮点数的绝对值 ..... 8-8
  - 求平方 ..... 8-9
  - 求平方根 ..... 8-10
  - 求余弦值 ..... 8-14
  - 求正切值 ..... 8-15
  - 求正弦值 ..... 8-13
  - 求指数值 ..... 8-11
  - 求自然对数 ..... 8-12
  - 取反无序异常位 ..... 12-5
  - 取整为长整型 ..... 3-13
- R**
- 若非则跳转 ..... 6-4
- S**
- 设置计数器值 ..... 4-9
  - 升值计数器线圈 ..... 4-10
  - 实际应用 ..... B-1
  - 实例
    - 定时器指令 ..... B-6
    - 计数器和比较指令 ..... B-10
    - 位逻辑指令 ..... B-2
    - 整型数学运算指令 ..... B-13
    - 字逻辑指令 ..... B-14
  - 实数乘 ..... 8-6
  - 实数除 ..... 8-7
  - 实数加 ..... 8-3
  - 实数减 ..... 8-5
  - 使用 MCR 功能的重要注意事项 ..... 10-13
  - 使用整数算术指令时得出
    - 状态字的位数值 ..... 7-2
  - 输出线圈 ..... 1-6
  - 双向计数器 ..... 4-3
  - 双字循环右移 ..... 11-13
  - 双字循环左移 ..... 11-11
  - 双字右移 ..... 11-9
  - 双字左移 ..... 11-8
- T**
- 条件跳转 ..... 6-3
  - 跳转指令 ..... 6-5

- W**
- 位逻辑指令概述 ..... 1-1
- 无条件跳转 ..... 6-2
- X**
- 向上取整 ..... 3-15
- 向下取整 ..... 3-16
- 循环移位指令概述 ..... 11-11
- Y**
- 移位指令概述 ..... 11-1
- 异常位二进制结果 ..... 12-6
- 异常位二进制结果取反 ..... 12-6
- 异常位无序 ..... 12-5
- 异常位溢出 ..... 12-2
- 异常位溢出取反 ..... 12-2
- Z**
- 在线帮助 ..... v
- 整数乘 ..... 7-5
- 整数除 ..... 7-6
- 整数加 ..... 7-3
- 整数减 ..... 7-4
- 整数算术指令概述 ..... 7-1
- 整数右移 ..... 11-2
- 整型转换为 BCD 码 ..... 3-3
- 整型转换为长整型 ..... 3-4
- 置位线圈 ..... 1-12
- 置位优先型 RS 双稳态触发器 ..... 1-14
- 中间输出 ..... 1-8
- 重置线圈 ..... 1-10
- 助记符
- 德语(SIMATIC) ..... A-4
- 英语(国际) ..... A-1
- 主控制继电器打开 ..... 10-14
- 主控制继电器关闭 ..... 10-16
- 主控制继电器激活 ..... 10-18
- 主控制继电器取消激活 ..... 10-19
- 转换指令概述 ..... 3-1
- 状态字中位的判断 ..... 8-2
- 字逻辑指令概述 ..... 14-1
- 字右移 ..... 11-7
- 字左移 ..... 11-5