SIEMENS

SIMATIC

用于 S7-300 和 S7-400 编程的语句表 (STL)

参考手册

10/2010
A5E03290620-01

法律资讯

警告提示系统

为了您的人身安全以及避免财产损失，必须注意本手册中的提示。人身安全的提示用一个警告三角表示，仅与财产损失有关的提示不带警告三角。警告提示根据危险等级由高到低如下表示。

⚠ 危险

表示如果不采取相应的小心措施，**将会**导致死亡或者严重的人身伤害。

⚠ 警告

表示如果不采取相应的小心措施，**可能**导致死亡或者严重的人身伤害。

⚠ 小心

带有警告三角，表示如果不采取相应的小心措施，可能导致轻微的人身伤害。

小心

不带警告三角，表示如果不采取相应的小心措施，可能导致财产损失。

注意

表示如果不注意相应的提示，可能会出现不希望的结果或状态。

当出现多个危险等级的情况下，每次总是使用最高等级的警告提示。如果在某个警告提示中带有警告可能导致人身伤害的警告三角，则可能在该警告提示中另外还附带有可能导致财产损失的警告。

合格的专业人员

仅允许安装和驱动与本文件相关的附属设备或系统。设备或系统的调试和运行仅允许由**合格的专业人员**进行。本文件安全技术提示中的合格专业人员是指根据安全技术标准具有从事进行设备、系统和电路的运行，接地和标识资格的人员。

按规定使用 Siemens 产品

注意下列各项：

⚠ 警告

Siemens 产品只允许用于目录和相关技术文件中规定的使用情况。如果要使用其他公司的产品和组件，必须得到 Siemens 推荐和允许。正确的运输、储存、组装、装配、安装、调试、操作和维护是产品安全、正常运行的前提。必须保证允许的环境条件。必须注意相关文件中的提示。

商标

所有带有标记符号 ® 的都Siemens AG的注册商标。标签中的其他符号可能是一些其他商标，这是出于保护所有者权利的目地由第三方使用而特别标示的。

责任免除

我们已对印刷品中所述内容与硬件和软件的一致性作过检查。然而不排除存在偏差的可能性，因此我们不保证印刷品中所述内容与硬件和软件完全一致。印刷品中的数据都按规定经过检测，必要的修正值包含在下一版本中。

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E03290620-01
℗ 09/2010

前言

用途

本手册是您以语句表编程语言 STL 创建用户程序的指南。

本手册还包含了对 STL 语言元素的语法和函数进行描述的参考部分。

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。

除此之外，还需要具有计算机应用能力和其它类似于 PC (例如，编程设备) 的、使用 MS Windows XP、MS Windows Server 2003 或 MS Windows 7 版操作系统的工作设备的知识。

手册应用范围

本手册适用于 STEP 7 编程软件包 5.5 版本。

符合的标准

STL 符合国际电工技术委员会 IEC 1131-3 标准所定义的“指令表”语言，但在操作方面有很大的不同。欲知更多资料，请参见 STEP 7 文件 NORM_TBL.RTF 中的标准表。

要求

如要有效地使用该语句表手册，应当熟悉 STEP 7 在线帮助所提供的 S7 编程理论。该语言包也使用 STEP 7 标准软件，因此应当熟悉该软件的操作，并阅读所附文档。

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 • 使用 STEP 7，使用入门手册 • 使用 STEP 7 进行编程 • 配置硬件和通讯连接，STEP 7 版本 • 从 S5 到 S7，变频器手册	提供给技术人员的基础信息，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。	6ES7810-4CA10-8BW0
STEP 7 参考书目 • 用于 S7-300/400 的梯形图 (LAD) /功能块图 (FBD) /语句表 (STL) 手册 • S7-300/400 的标准函数及系统函数 第 1 卷和第 2 卷	提供了参考信息，并描述了编程语言 LAD、FBD、STL、标准函数以及系统函数，扩充了 STEP 7 基础信息的范围。	6ES7810-4CA10-8BW1

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 标准软件中的一部分。
STL/LAD/FBD 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 标准软件中的一部分。

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。可以通过通过菜单命令**帮助 > 上下文相关的帮助**，或按下 F1 键或通过使用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 中的常规帮助。
- 可以通过按“词汇表”按钮，调用所有 STEP7 应用程序的词汇表。

本手册是“语句表帮助”的摘录。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单个 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 90026) 的培训总部，以获取详细信息。

网址： http://www.sitrain.com

技术支持

您可访问“技术支持”来了解所有的工业自动化和驱动技术产品

- 通过网站请求支持
  http://www.siemens.com/automation/support-request

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。
- 您当地的关于工业自动化和驱动技术的销售代表。
- 关于现场服务、维修、备件和咨询的信息。

目录

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。

布尔位逻辑适用于下列基本指令：

- A 与运算
- AN 与非运算
- O 或运算
- ON 或非运算
- X 异或运算
- XN 同或运算
- O 先与运算后或运算

可使用下列指令执行嵌套表达式：

- A(与运算嵌套开始
- AN(与非运算嵌套开始
- O(或运算嵌套开始
- ON(或非运算嵌套开始
- X(异或运算嵌套开始
- XN(同或运算嵌套开始
-) 嵌套结束

可使用下列指令之一终止布尔位逻辑串：

- = 赋值
- R 复位
- S 置位

可使用下列指令之一更改逻辑运算的结果 (RLO) ：

- NOT　对 RLO 取反
- SET　置位 RLO (=1)
- CLR　清零 RLO (=0)
- SAVE　将 RLO 保存到 BR 寄存器

对上升沿或下降沿转换做出反应的其它指令：

- FN　下降沿
- FP　上升沿

1.2 A 与运算

格式

A <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D、T、C

描述

A 检查寻址位的状态是否为“1”，并将测试结果与 RLO 进行与运算。

状态字位检查：

AND 指令还可通过下列地址直接检查状态字：==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

状态字

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

实例

STL Program		Relay Logic	
		Power rail	
A	**I 1.0**	I 1.0 signal state 1	NO contact
A	**I 1.1**	I 1.1 signal state 1	NC contact
=	**Q 4.0**	Q 4.0 signal state 1	Coil
Displays closed switch			

1.3 AN 与非运算

格式

N <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D、T、C

描述

AN 检查寻址位的状态是否为“0”，并将测试结果与 RLO 进行与运算。

状态字位检查：

AND NOT 指令还可通过下列地址直接检查状态字：==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

状态字

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

实例

STL Program		Relay Logic	
		Power rail	
A	**I 1.0**	I 1.0 Signal state 0	NO contact
AN	**I 1.1**	I 1.1 Signal state 1	NC contact
=	**Q 4.0**	Q 4.0 Signal state 0	Coil

1.4 O 或运算

格式

O <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D、T、C

描述

O 检查寻址位的状态是否为“1”，并将测试结果与 RLO 进行或运算。

状态字位检查：

OR 指令还可通过下列地址直接检查状态字：==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

状态字

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

实例

STL Program		Relay Logic
		Power rail
O	**I 1.0**	I 1.0 Signal state 1 No contact I 1.1 Signal state 0 No contact
O	**I 1.1**	
=	**Q 4.0**	Q 4.0 Signal state 1 Coil
		Displays closed switch

1.5 ON 或非运算

格式

ON <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D、T、C

描述

ON 检查寻址位的状态是否为“0”，并将测试结果与 RLO 进行或运算。

状态字位检查：

OR NOT 指令还可通过下列地址直接检查状态字：==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

状态字

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

实例

STL Program		Relay Logic
		Power rail
O	**I 1.0**	I 1.0 Signal state 0 NO contact
ON	**I 1.1**	I 1.1 Signal state 1 NC contact
=	**Q 4.0**	Q 4.0 Signal state 1 Coil

1.6 X 异或运算

格式

X <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D、T、C

描述

X 检查寻址位的状态是否为“1”，并将测试结果与 RLO 进行异或运算。

也可以重复使用 **Exclusive OR** 函数。这样，如果有奇数个被检查地址状态为“1”，则逻辑运算的最终结果为“1”。

状态字位检查：

EXCLUSIVE OR 指令还可通过使用下列地址直接检查状态字：==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

状态字

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

实例

Statement List Program		Relay Logic
		Power rail
X	**I 1.0**	Contact I 1.0
X	**I 1.1**	Contact I 1.1
=	**Q 4.0**	Q 4.0 Coil

1.7 XN 同或运算

格式

XN <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D、T、C

描述

XN 检查寻址位的状态是否为“0”，并将测试结果与 RLO 进行异或运算。

状态字位检查：

EXCLUSIVE OR NOT 指令还可通过使用下列地址直接检查状态字：==0、<>0、>0、<0、>=0、<=0、OV、OS、UO、BR。

状态字

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

实例

Statement List Program		Relay Logic
		Power rail
X	**I 1.0**	Contact I 1.0
XN	**I 1.1**	Contact I 1.1
=	**Q 4.0**	Q 4.0 Coil

1.8 O 先与运算后或运算

格式

O

描述

O 函数根据下列规则对 AND 函数执行逻辑 OR 指令：先与运算后或运算

状态字

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

实例

Statement List Program		Relay Logic
		Power rail
A	**I 0.0**	I 0.0
A	**M 10.0**	I 0.2
		M 10.1
O		
A	**I 0.2**	M 10.0
A	**M 0.3**	M 0.3
O	**M 10.1**	
=	**Q 4.0**	Q 4.0 Coil

1.9 A(与运算嵌套开始

格式

A(

描述

A((与运算嵌套开始) 将 RLO 和 OR 位及一个函数代码保存到嵌套的堆栈中。最多可有七个嵌套堆栈条目。

状态字

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

实例

Statement List Program		Relay Logic
		Power rail
A(		
O	**I 0.0**	I 0.0
O	**M 10.0**	M 10.0
)		
A(		
O	**I 0.2**	I 0.2
O	**M 10.3**	M10.3
)		
A	**M 10.1**	M 10.1
=	**Q 4.0**	Q 4.0 Coil

1.10 AN(与非运算嵌套开始

格式

AN(

描述

AN((与非运算嵌套打开) 将 RLO 和 OR 位及一个函数代码保存到嵌套的堆栈中。最多可有七个嵌套堆栈条目。

状态字

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

1.11 O(或运算嵌套开始

格式

O(

描述

O((或运算嵌套打开) 将 RLO 和 OR 位及一个函数代码保存到嵌套的堆栈中。最多可有七个嵌套堆栈条目。

状态字

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

1.12 ON(　或非运算嵌套开始

格式

ON(

描述

ON((OR NOT 嵌套打开) 将 RLO 和 OR 位及一个函数代码保存到嵌套的堆栈中。最多可有七个嵌套堆栈条目。

状态字

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

1.13 X(　异或运算嵌套开始

格式

X(

描述

X((异或运算嵌套打开) 将 RLO 和 OR 位及一个函数代码保存到嵌套堆栈中。最多可有七个嵌套堆栈条目。

状态字

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

1.14 XN(同或运算嵌套开始

格式

XN(

描述

XN((同或运算嵌套打开) 将 RLO 和 OR 位及一个函数代码保存到嵌套堆栈中。最多可有七个嵌套堆栈条目。

状态字

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

1.15) 嵌套结束

格式

)

描述

) (嵌套结束) 从嵌套堆栈中删除条目，恢复 OR 位，根据函数代码将包含在堆栈条目中的 RLO 与当前 RLO 互连，并将结果分配给 RLO。如果函数代码为“AND”或“AND NOT”，则 OR 位也包括在内。

打开括号组的语句：

- U(与运算嵌套开始
- UN(与非运算嵌套开始
- O(或运算嵌套开始
- ON(或非运算嵌套开始
- X(异或运算嵌套开始
- XN(同或运算嵌套开始

状态字

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

实例

Statement List Program		Relay Logic
		Power rail
A(		
O	**I 0.0**	I 0.0
O	**M 10.0**	M 10.0
)		
A(		
O	**I 0.2**	I 0.2
O	**M 10.3**	M10.3
)		
A	**M 10.1**	M 10.1
=	**Q 4.0**	Q 4.0 Coil

1.16 = 赋值

格式

<位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D

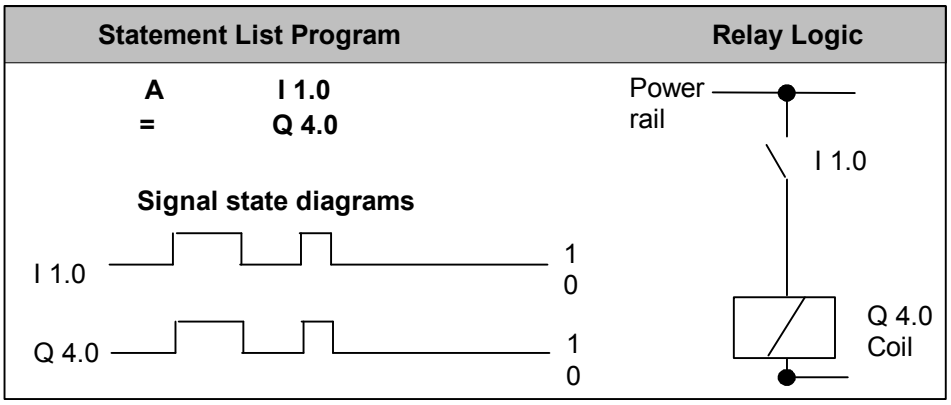
描述

= <位>如果 MCR = 1，则将 RLO 写入打开的主控继电器的寻址位。如果 MCR = 0，则将值 0 而不是 RLO 写入寻址位。

状态字

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

实例



1.17 R 复位

格式

R <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D

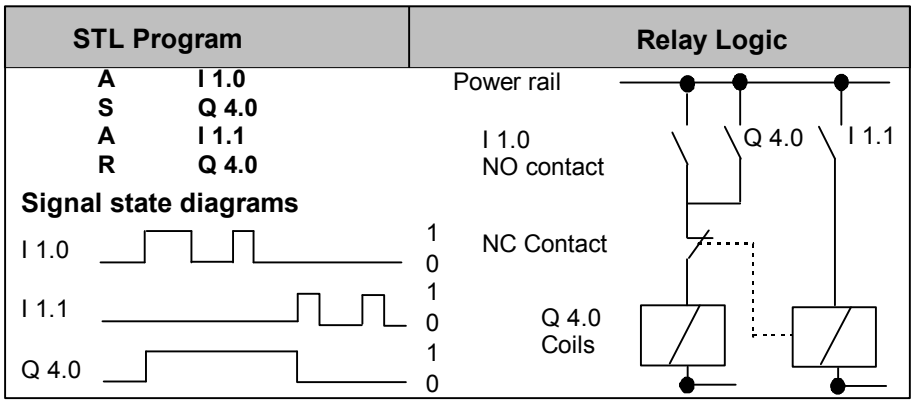
描述

R (将位进行复位) 如果 RLO = 1 且主控继电器 MCR = 1，则在寻址位中放入“0”。
如果 MCR = 0，则寻址位不变。

状态字

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

实例



1.18 S 置位

格式

S <位>

地址	数据类型	存储区域
<位>	BOOL	I、Q、M、L、D

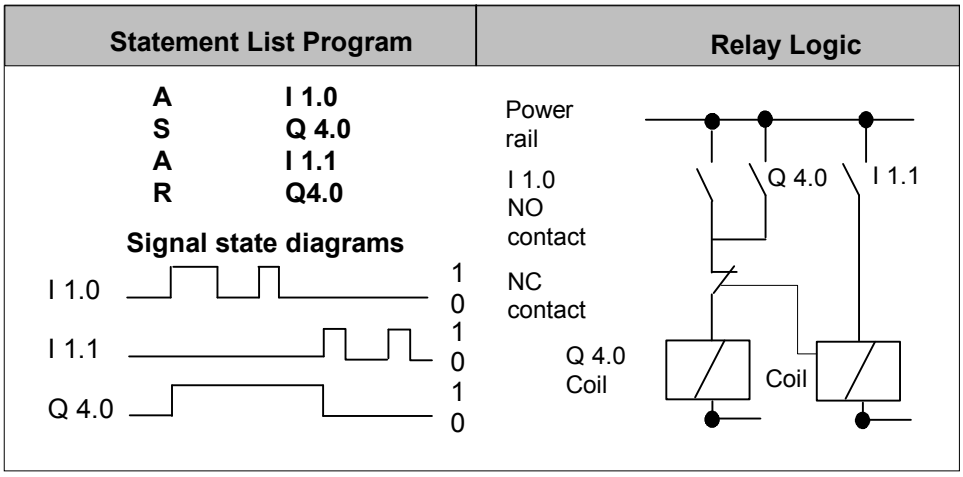
指令描述

S (将位进行置位) 如果 RLO = 1 且打开的主控继电器 MCR = 1，则在寻址位中放入“1”。
如果 MCR = 0，则寻址位不变。

状态字

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

实例



1.19 NOT 对 RLO 取反

格式

NOT

描述

NOT 对 RLO 取反。

状态字

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

1.20 SET 置位 RLO (=1)

格式

SET

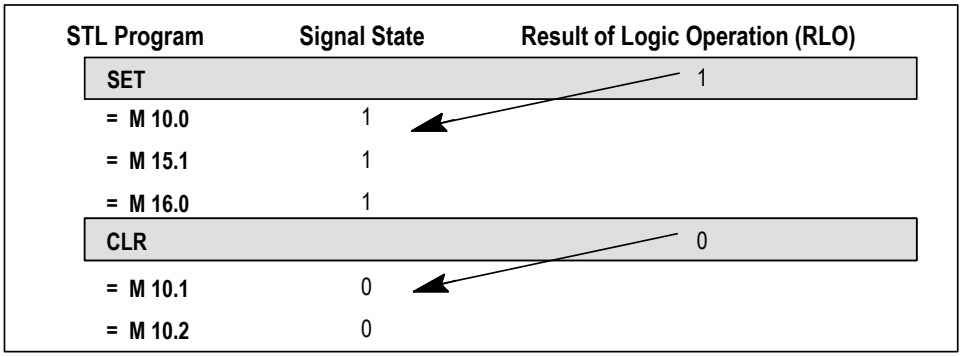
描述

SET 将 RLO 置位到信号状态“1”。

状态字

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

实例


STL Program
Signal State
Result of Logic Operation (RLO)
SET
1
= M 10.0
1
= M 15.1
1
= M 16.0
1
CLR
0
= M 10.1
0
= M 10.2
0

1.21 CLR 清零 RLO (=0)

格式

CLR

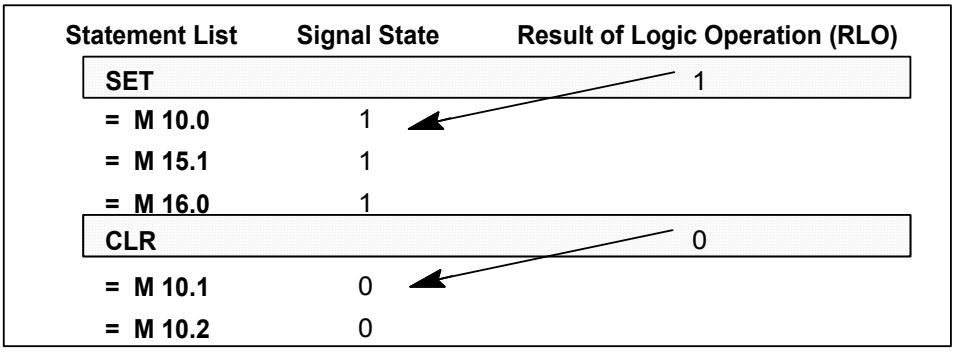
描述

CLR 将 RLO 设置到信号状态“0”。

状态字

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

实例



1.22 SAVE 将 RLO 保存到 BR 寄存器

格式

SAVE

指令描述

SAVE 将 RLO 保存到 BR 位中。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。

建议不要在同一个块或二级块中对 BR 位使用 **SAVE** 并执行后续查询，因为 BR 位可能会被这两个操作之间的大量指令改变。退出块之前使用 **SAVE** 指令的意义在于，此操作将 ENO 输出 (= BR 位) 设置为 RLO 位的值，从而可以将该块的出错处理添加至此。

状态字

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

1.23 FN 下降沿

格式

FN <位>

地址	数据类型	存储区域	描述
<位>	BOOL	I、Q、M、L、D	沿标记，存储 RLO 的上一信号状态。

描述

FN <位> (RLO 下降沿) 检测 RLO 从“1”跳转到“0”时的下降沿，并以 RLO = 1 指示此情况。

在每个程序扫描周期期间，都会将 RLO 位的信号状态与上一周期获取的状态进行比较，以判断状态是否改变。上一 RLO 状态必须存储在沿标记地址 (**<位>**) 中才能进行比较。如果当前状态与上一 RLO“1”状态 (检测到下降沿) 不同，则执行此指令之后 RLO 位将为“1”。

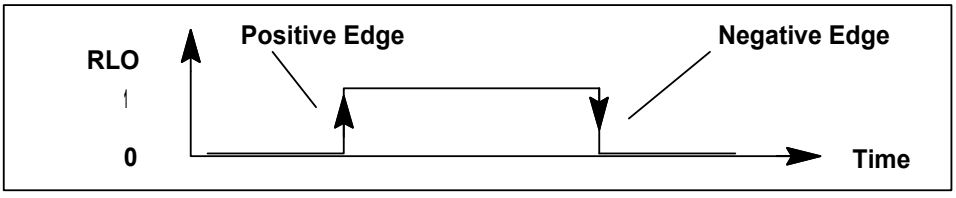
注意

由于块的本地数据只在块运行时有效，因此如果要监视的位位于过程映像中，则此指令没有意义。

状态字

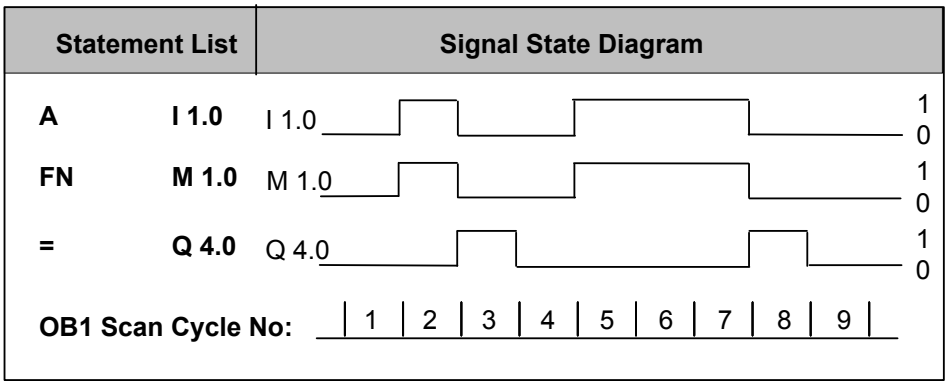
	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写：	-	-	-	-	-	0	x	x	1

定义



实例

如果可编程逻辑控制器在触点 I 1.0 检测到下降沿，则会在 Q 4.0 处励磁线圈一个 OB1 扫描周期。



1.24 FP 上升沿

格式

FP <位>

地址	数据类型	存储区域	描述
<位>	BOOL	I、Q、M、L、D	沿标记，存储 RLO 的上一信号状态。

描述

FP <位> (RLO 上升沿) 检测 RLO 从“0”跳转到“1”时的上升沿，并以 RLO = 1 指示此情况。

在每个程序扫描周期期间，都会将 RLO 位的信号状态与上一周期获取的状态进行比较，以判断状态是否改变。上一 RLO 状态必须存储在沿标记地址 (**<位>**) 中才能进行比较。如果当前状态与上一 RLO“0”状态 (检测到上升沿) 不同，则执行此指令之后 RLO 位将为“1”。

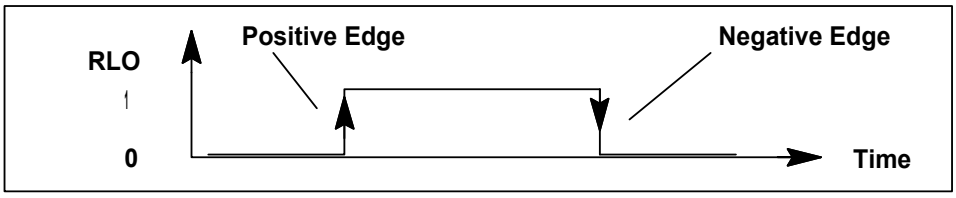
注意

由于块的本地数据只在块运行时有效，因此如果要监视的位位于过程映像中，则此指令没有意义。

状态字

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

定义



实例

如果可编程逻辑控制器在触点 I 1.0 检测到上升沿，则会在 Q 4.0 处励磁线圈一个 OB1 扫描周期。

Statement List		Signal State Diagram
A	I 1.0	I 1.0 1 0
FP	M 1.0	M 1.0 1 0
=	Q 4.0	Q 4.0 1 0
OB1 Scan Cycle No:		1 2 3 4 5 6 7 8 9

2 比较指令

2.1 比较指令概述

描述

根据您选择的比较类型来比较 ACCU1 和 ACCU2：

==　ACCU1 等于 ACCU2
<>　ACCU1 不等于 ACCU2
>　ACCU1 大于 ACCU2
<　ACCU1 小于 ACCU2
>=　ACCU1 大于等于 ACCU2
<=　ACCU1 小于等于 ACCU2

如果比较结果为 true，则此函数的 RLO 为“1”。状态字的位 CC 1 和 CC 0 指示关系“小于”、“等于”或“大于”。

有执行以下功能的比较指令：

- ? I　比较整数 (16 位)
- ? D　比较长整数 (32 位)
- ? R　比较浮点数 (32 位)

2.2 ? I 比较整数 (16 位)

格式

==I、<>I、>I、<I、>=I、<=I

指令描述

比较整数 (16 位) 指令将 ACCU 2-L 的内容与 ACCU 1-L 的内容进行比较。ACCU 2-L 和 ACCU 1-L 的内容被解释为 16 位整数。其比较结果由 RLO 和相关状态字位的设置来表示。RLO = 1 表示比较结果为 true；RLO = 0 表示比较结果为 false。状态字的位 CC 1 和 CC 0 指示关系“小于”、“等于”或“大于”。

状态字

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

RLO 的值

执行的比较指令	RLO 结果 (如果 ACCU 2 > ACCU 1	RLO 结果 (如果 ACCU 2 = ACCU 1	RLO 结果 (如果 ACCU 2 < ACCU 1
==I	0	1	0
<>I	1	0	1
>I	1	0	0
<I	0	0	1
>=I	1	1	0
<=I	0	1	1

实例

```
STL             解释
L    MW10       //装载 MW10 的内容 (16 位整数)。
L    IW24       //装载 IW24 的内容 (16 位整数)。
>I              //比较 ACCU 2-L (MW10) 是否大于 (>)  ACCU 1-L (IW24)。
=    M 2.0      //RLO = 1 (如果 MW10 > IW24)。
```

2.3 ? D 比较长整数 (32 位)

格式

==D、<>D、>D、<D、>=D、<=D

指令描述

比较长整数 (32 位) 指令将 ACCU 2 的内容与 ACCU 1 的内容进行比较。ACCU 2 和 ACCU 1 的内容被解释为 32 位整数。其比较结果由 RLO 和相关状态字位的设置来表示。RLO = 1 表示比较结果为 true；RLO = 0 表示比较结果为 false。状态字的位 CC 1 和 CC 0 指示关系“小于”、“等于”或“大于”。

状态字

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

RLO 的值

执行的比较指令	RLO 结果 (如果 ACCU 2 > ACCU 1)	RLO 结果 (如果 ACCU 2 = ACCU 1)	RLO 结果 (如果 ACCU 2 < ACCU 1)
==D	0	1	0
<>D	1	0	1
>D	1	0	0
<D	0	0	1
>=D	1	1	0
<=D	0	1	1

实例

```
STL             解释
L    MD10       //装载 MD10 的内容 (长整型，32 位)。
L    ID24       //装载 ID24 的内容 (长整型，32 位)。
>D              //比较 ACCU 2 (MD10) 是否大于 (>)  ACCU 1 (ID24)。
=    M 2.0      //RLO = 1 (如果 MD10 > ID24)
```

2.4 ? R 比较浮点数 (32 位)

格式

==R、<>R、>R、<R、>=R、<=R

指令描述

比较浮点数 (32 位，IEEE 754) 指令将 ACCU 2 的内容与 ACCU 1 的内容进行比较。ACCU 1 和 ACCU 2 的内容被解释为浮点数 (32 位，IEEE754)。其比较结果由 RLO 和相关状态字位的设置来表示。RLO = 1 表示比较结果为 true；RLO = 0 表示比较结果为 false。状态字的位 CC 1 和 CC 0 指示关系“小于”、“等于”或“大于”。

状态字

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

RLO 的值

执行的比较指令	RLO 结果 (如果 ACCU 2 > ACCU 1)	RLO 结果 (如果 ACCU 2 = ACCU 1)	RLO 结果 (如果 ACCU 2 < ACCU 1)
==R	0	1	0
<>R	1	0	1
>R	1	0	0
<R	0	0	1
>=R	1	1	0
<=R	0	1	1

实例

```
STL                 解释
L     MD10          //装载 MD10 的内容 (浮点数)。
L     1.359E+02     //装载常数 1.359E+02。
>R                  //比较 ACCU 2 (MD10) 是否大于 (>) ACCU 1 (1.359-E+02)。
=     M 2.0         //RLO = 1 (如果 MD10 > 1.359E+02)。
```

3 转换指令

3.1 转换指令概述

描述

可使用下列指令将二进制编码的十进制数和整数转换为其它类型的数字：

- BTI 将 BCD 码转换为整型 (16 位)
- ITB 将整型 (16 位) 转换为 BCD 码
- BTD 将 BCD 码转换为整型 (32 位)
- ITD 将整型 (16 位) 转换为长整型 (32 位)
- DTB 将长整型 (32 位) 转换为 BCD 码
- DTR 将长整型 (32 位) 转换为浮点数 (32 位 IEEE 754)

可使用下列指令计算整数的补 (反) 码，或将浮点数的符号取反：

- INVI 对整数 (16 位) 求反码
- INVD 对长整数 (32 位) 求反码
- NEGI 对整数 (16 位) 求补码
- NEGD 对长整数 (32 位) 求补码
- NEGR 浮点数 (32 位，IEEE 754) 取反

可使用下列“改变累加器 1 中的位顺序”指令将累加器 1 的低字字节或整个累加器中的字节的顺序反转。

- CAW 改变 ACCU 1-L (16 位) 中的字节顺序
- CAD 改变 ACCU 1 (32 位) 中的字节顺序

可使用下列任何指令将累加器 1 中的 32 位 IEEE 浮点数转换为 32 位整型 (长整型)。各个指令的取整方法有所不同：

- RND 取整
- TRUNC 截断
- RND+ 取整为高位长整数
- RND- 取整为低位长整数

3.2 BTI 将 BCD 码转换为整型 (16 位)

格式

BTI

描述

BTI (3 位 BCD 数从十进制到二进制的转换) 将 ACCU 1-L 的内容解释为三位二进制编码的十进制数 (BCD 码)，并将其转换为 16 位整型。结果存储在累加器 1 的低字中。累加器 1 的高字和累加器 2 保持不变。

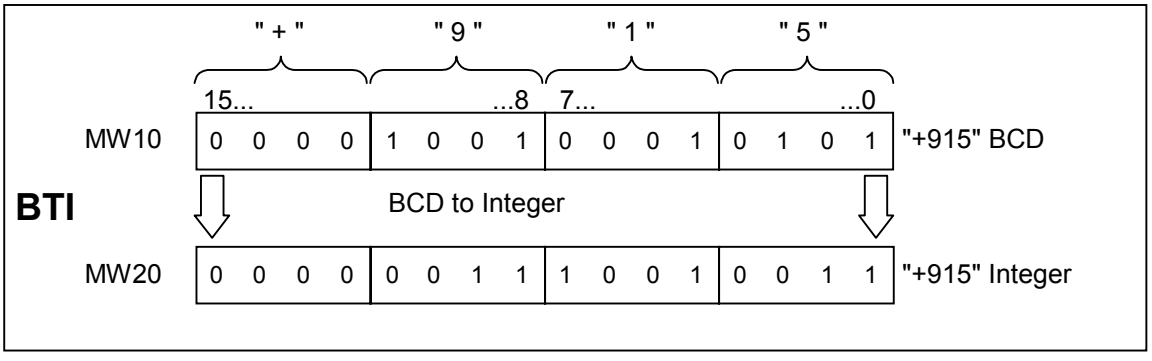
ACCU 1-L 中的 BCD 数字：BCD 数字的允许值范围从“-999”至“+999”。位 0 到位 11 解释为数值，位 15 解释为 BCD 数字的符号 (0 = 正，1= 负)。位 12 至位 14 在转换中不使用。如果 BCD 数字的十进制 (四位) 数字处于 10 至 15 的无效范围，则在转换期间会出现 BCDF 错误。通常，CPU 会转入 STOP 模式。但是，通过对 OB121 编程可设计另一种出错响应，用以处理该同步编程错误。

状态字

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

实例

```
STL             解释
L     MW10      //将 BCD 数字载入 ACCU 1-L。
BTI             //从 BCD 码转换为整型；将结果存储在 ACCU 1-L 中。
T     MW20      //将结果 (整数) 传送到 MW20。
```



3.3 ITB 将整型 (16 位) 转换为 BCD 码

格式

ITB

描述

ITB (16 位整数从二进制到十进制的转换) 将 ACCU 1-L 的内容解释为 16 位整数，并将其转换为三位二进制编码的十进制数 (BCD 码)。结果存储在累加器 1 的低字中。位 0 到位 11 包含 BCD 数字的值。位 12 至位 15 用来表示 BCD 数字的符号状态 (0000 = 正，1111= 负)。累加器 1 的高字和累加器 2 保持不变。

BCD 数字的范围为“-999”至“+999”。如果超出允许范围，则状态位 OV 和 OS 被置位为 1。

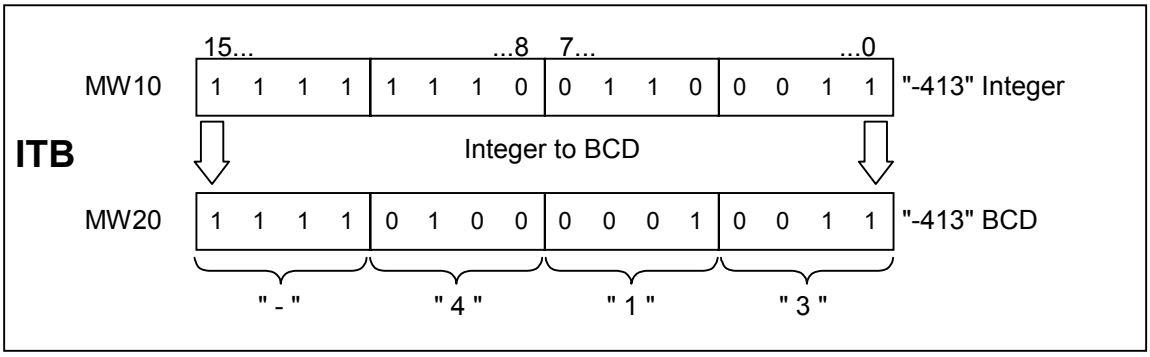
执行指令时既不考虑也不会影响 RLO。

状态字

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

实例

```
STL          解释
L     MW10   //将整数载入 ACCU 1-L。
ITB          //从整型转换为 BCD (16 位)；将结果存储在 ACCU 1-L 中。
T     MW20   //将结果 (BCD 数字) 传送到 MW20。
```



3.4 BTD 将 BCD 码转换为整型 (32 位)

格式

BTD

描述

BTD (7 位 BCD 数字从十进制到二进制的转换) 将 ACCU 1 的内容解释为 7 位二进制编码的十进制数 (BCD)，并将其转换为 32 位长整型。结果存储在累加器 1 中。累加器 2 保持不变。

ACCU 1 中的 BCD 数字：BCD 数字的允许值范围从“-9,999,999”至“+9,999,999”。位 0 到位 27 解释为数值，位 31 解释为 BCD 数字的符号 (0 = 正，1= 负)。位 28 至位 30 在转换中不使用。

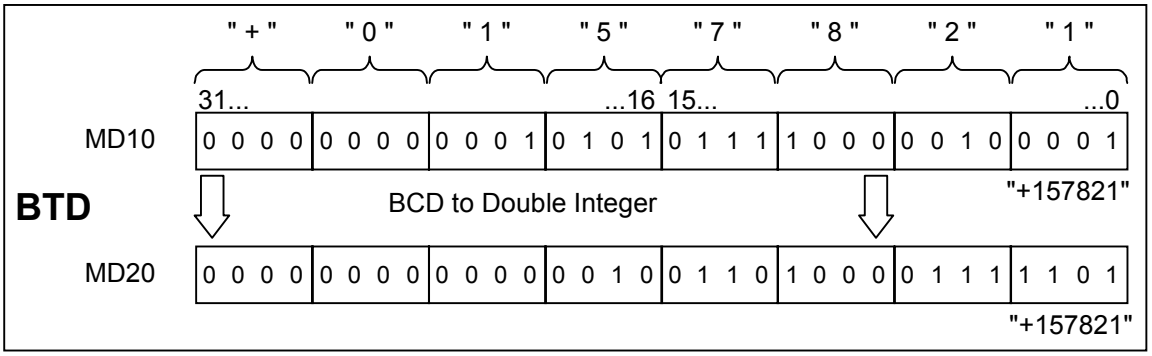
如果任何十进制数 (BCD 编码的四位组) 处于 10 至 15 的无效范围，则在转换期间会出现 BCDF 错误。通常，CPU 会转入 STOP 模式。但是，通过对 OB121 编程可设计另一种出错响应，用以处理该同步编程错误。

状态字

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

实例

```
STL             解释
L     MD10      //将 BCD 数字载入 ACCU 1。
BTD             //从 BCD 码转换为整型；将结果存储在 ACCU 1 中。
T     MD20      //将结果 (长整数) 传送到 MD20。
```



3.5 ITD 将整型 (16 位) 转换为长整型 (32 位)

格式

ITD

描述

ITD (16 位整数转换为 32 位整数) 将 ACCU 1-L 的内容解释为 16 位整数并将其转换为 32 位长整数。结果存储在累加器 1 中。累加器 2 保持不变。

状态字

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

实例

```
STL             解释
L     MW12      //将整数载入 ACCU 1。
ITD             //从整型 (16 位) 转换为长整型 (32 位)；将结果存储在 ACCU 1 中。
T     MD20      //将结果 (长整型) 传送到 MD20。
```

实例：MW12 = “-10” (整型，16 位)

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . .16	15 . . .	..	. .	. . .0
执行 **ITD** 之前	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
执行 **ITD** 之后	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 或 1，这些位不用于转换)							

3.6 DTB 将长整型 (32 位) 转换为 BCD 码

格式

DTB

描述

DTB (32 位整型数的二进制到十进制转换) 将 ACCU 1 中的内容解析为 32 位长整型数，并将其转换为七位二进制编码的十进制数 (BCD)。结果保存在累加器 1 中。位 0 到位 27 包含 BCD 数字的值。位 28 至位 31 用来表示 BCD 数字的符号状态 (0000 = 正，1111 = 负)。累加器 2 保持不变。

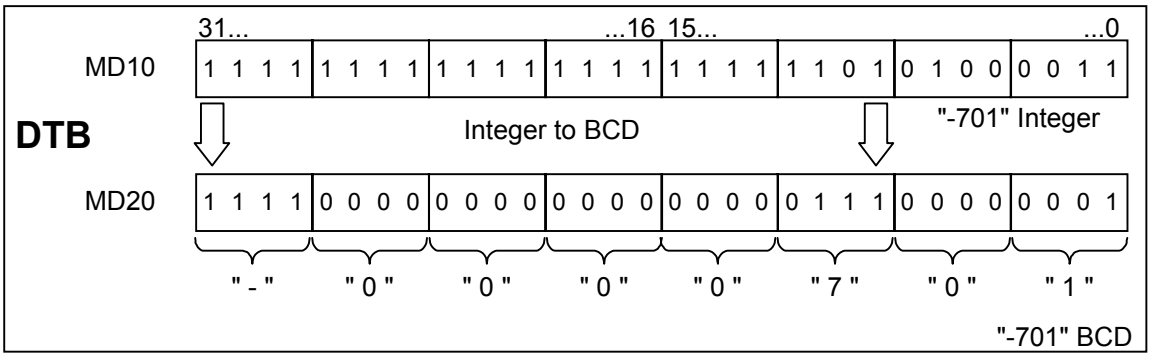
BCD 数的范围为“-9,999,999”至“+9,999,999”。如果超出允许范围，则状态位 OV 和 OS 被置位为 1。

状态字

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

实例

```
STL          解释
L     MD10   //将 32 位整数载入 ACCU 1。
DTB          //从整型 (32 位) 转换为 BCD 码，将结果存储在 ACCU 1 中。
T     MD20   //将结果 (BCD 数字) 传送到 MD20。
```



3.7 DTR 将长整型 (32 位) 转换为浮点数 (32 位 IEEE 754)

格式

DTR

描述

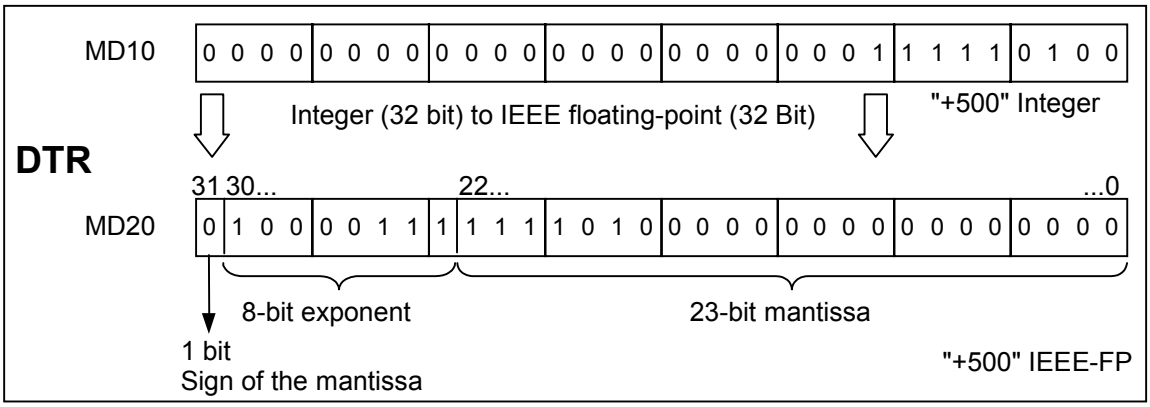
DTR (32 位整数转换为 32 位 IEEE 浮点数) 将 ACCU 1 的内容解释为 32 位长整型，并将其转换为 32 位 IEEE 浮点数。如必要，该指令会对结果取整。 (32 位整数比 32 位浮点数精度更高)。结果存储在累加器 1 中。

状态字

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

实例

```
STL          解释
L     MD10   //将 32 位整数载入 ACCU 1。
DTR          //从长整型转换为浮点型 (32 位 IEEE FP)；将结果存储在 ACCU 1 中。
T     MD20   //将结果 (BCD 数字) 传送到 MD20。
```



3.8 INVI 对整数 (16 位) 求反码

格式

INVI

描述

INVI (对整数求反码) 在 ACCU 1-L 中形成 16 位数值的二进制反码。二进制反码是通过将各个位的值取反形成的，即，用“0”替换“1”，用“1”替换“0”。结果存储在累加器 1 的低字中。

状态字

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

实例

```
STL          解释
L     IW8    //将值载入 ACCU 1-L。
INVI         //形成 16 位二进制反码。
T     MW10   //将结果传送到 MW10。
```

目录	ACCU1-L			
位	15 . . .	. .	. .	. . .0
执行 **IVNI** 之前	0110	0011	1010	1110
执行 **INVI** 之后	1001	1100	0101	0001

3.9 INVD 对长整数 (32 位) 求反码

格式

INVD

描述

INVD (二进制反码双精度整数) 在 ACCU 1 中形成 32 位数值的二进制反码。二进制反码是通过将各个位的值取反形成的，即，用“0”替换“1”，用“1”替换“0”。结果存储在累加器 1 中。

状态字

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

实例

```
STL             解释
L       ID8     //将值载入 ACCU 1 中。
INVD            //形成二进制反码 (32 位)。
T       MD10    //将结果传送到 MD10。
```

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **IVND** 之前	0110	1111	1000	1100	0110	0011	1010	1110
执行 **INVD** 之后	1001	0000	0111	0011	1001	1100	0101	0001

3.10 NEGI 对整数 (16 位) 求补码

格式

NEGI

描述

INVI (二进制补码整数) 在 ACCU 1-L 中形成 16 位数值的二进制补码。二进制补码是通过将各个位的值取反形成的，即，用“0”替换“1”，用“1”替换“0”；然后加“1”。结果存储在累加器 1 的低字中。补码指令等效于乘以“-1”。状态位 CC 1、CC 0、OS 和 OV 则根据函数运算的结果来设置。

状态字

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

状态字生成	CC 1	CC 0	OV	OS
结果 = 0	0	0	0	-
-32768 <= 结果 <= -1	0	1	0	-
32767 >= 结果 >= 1	1	0	0	-
结果 = 2768	0	1	1	1

实例

```
STL             解释
L     IW8       //将值载入 ACCU 1-L。
NEGI            //形成 16 位二进制补码。
T     MW10      //将结果传送到 MW10。
```

目录	ACCU1-L			
位	15 . . .	. .	. .	. . . 0
执行 **NEGI** 之前	0101	1101	0011	1000
执行 **NEGI** 之后	1010	0010	1100	1000

3.11 NEGD 对长整数 (32 位) 求补码

格式

NEGD

描述

NEGD (二进制补码双精度整数) 在 ACCU 1 中形成 32 位数值的二进制补码。二进制补码是通过将各个位的值取反形成的，即，用“0”替换“1”，用“1”替换“0”；然后加“1”。结果存储在累加器 1 中。补码指令等效于乘以“-1”。执行该指令时不涉及 RLO，也不会影响 RLO。状态位 CC 1、CC 0、OS 和 OV 则根据函数运算的结果来设置。

状态字

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

状态字生成	CC 1	CC 0	OV	OS
结果 = 0	0	0	0	-
-2.147.483.647 <= 结果 <= -1	0	1	0	-
2.147.483.647 >= 结果 >= 1	1	0	0	-
结果 = -2 147 483 648	0	1	1	1

实例

```
STL             解释
L     ID8       //将值载入 ACCU 1 中。
NEGD            //生成二进制补码 (32 位)。
T     MD10      //将结果传送到 MD10。
```

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **NEGD** 之前	0101	1111	0110	0100	0101	1101	0011	1000
执行 **NEGD** 之后	1010	0000	1001	1011	1010	0010	1100	1000
	(X = 0 或 1，这些位不用于转换)							

3.12 NEGR 浮点数 (32 位，IEEE 754) 取反

格式

NEGR

指令描述

NEGR (32 位 IEEE 浮点数取反) 对 ACCU 1 中的浮点数 (32 位，IEEE 754) 取反。该指令将反转 ACCU 1 中位 31 (尾数的符号) 的状态。结果存储在累加器 1 中。

状态字

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

实例

```
STL             解释
L       ID8     //将值载入 ACCU 1 (实例：ID 8 = 1.5E+02)。
NEGR            //浮点数 (32 位，IEEE 754) 取反；结果存储在 ACCU 1 中。
T       MD10    //将结果传送到 MD10 (实例：结果 = −1.5E+02)。
```

3.13 CAW 改变 ACCU 1-L (16 位) 中的字节顺序

格式

CAW

描述

CAW 反转 ACCU 1-L 中的字节顺序。结果存储在累加器 1 的低字中。累加器 1 的高字和累加器 2 保持不变。

状态字

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

实例

```
STL             解释
L     MW10      //将 MW10 的值载入 ACCU 1。
CAW             //反转 ACCU 1-L 中的字节顺序。
T     MW20      //将结果传送到 MW20。
```

目录	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
执行 CAW 之前	值 A	值 B	值 C	值 D
执行 CAW 之后	值 A	值 B	值 D	值 C

3.14 CAD 改变 ACCU 1 (32 位) 中的字节顺序

格式

CAD

描述

CAD 反转 ACCU 1 中的字节顺序。结果存储在累加器 1 中。累加器 2 保持不变。

状态字

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

实例

```
STL             解释
L     MD10      //将 MD10 的值载入 ACCU 1。
CAD             //反转 ACCU 1 中的字节顺序。
T     MD20      //将结果传送到 MD20。
```

目录	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
执行 CAD 之前	值 A	值 B	值 C	值 D
执行 CAD 之后	值 D	值 C	值 B	值 A

3.15 RND 取整

格式

RND

描述

RND (32 位 IEEE 浮点数转换为 32 位整型) 将 ACCU 1 的内容解释为 32 位 IEEE 浮点数 (32 位，IEEE 754)。该指令将 32 位 IEEE 浮点数转换位 32 位整型 (长整型)，并将结果取整为最接近的整数。如果所转换数字的小数部分介于偶数和奇数结果之间，则该指令选择偶数结果。如果超出允许范围，则状态位 OV 和 OS 被置位为 1。结果存储在累加器 1 中。

出现错误 (使用了不能表示为 32 位整数的 NaN 或浮点数) 时不执行转换并显示溢出。

状态字

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

实例

```
STL          解释
L    MD10    //将浮点数载入 ACCU 1-L。
RND          //将浮点数 (32 位，IEEE 754) 转换为整型 (32 位) 并对结果进行舍入。
T    MD20    //将结果 (长整数) 传送到 MD20。
```

转换前的值		转换后的值
MD10 = “100.5”	=> RND =>	MD20 = “+100”
MD10 = “-100.5”	=> RND =>	MD20 = “-100”

3.16 TRUNC 截断

格式

TRUNC

描述

TRUNC (32 位 IEEE 浮点数转换为 32 位整型) 将 ACCU 1 的内容解释为 32 位 IEEE 浮点数。该指令将 32 位 IEEE 浮点数转换为 32 位整型 (长整型)。运算结果为所转换浮点数的整数部分 (IEEE 取整模式“取整到零”)。如果超出允许范围，则状态位 OV 和 OS 被置位为 1。结果存储在累加器 1 中。

出现错误 (使用了不能表示为 32 位整数的 NaN 或浮点数) 时不执行转换并显示溢出。

状态字

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

实例

```
STL             解释
L      MD10     //将浮点数载入 ACCU 1-L。
TRUN            //将浮点数 (32 位，IEEE 754) 转换为整型 (32 位)，
C               //并对结果取整。将结果存储在 ACCU 1 中。
T      MD20     //将结果 (长整数) 传送到 MD20。
```

转换前的值		转换后的值
MD10 = “100.5”	=> TRUNC =>	MD20 = “+100”
MD10 = “-100.5”	=> TRUNC =>	MD20 = “-100”

3.17 RND+ 取整为高位长整数

格式

RND+

描述

RND+ (32 位 IEEE 浮点数转换为 32 位整型) 将 ACCU 1 的内容解释为 32 位 IEEE 浮点数。该指令将 32 位 IEEE 浮点数转换为 32 位整型 (长整型)，并将结果取整为大于或等于所转换浮点数的最小整数 (IEEE 取整模式“取整为正无穷”)。如果数字超出允许范围，则状态位 OV 和 OS 被置位到 1。结果存储在累加器 1 中。

出现错误 (使用了不能表示为 32 位整数的 NaN 或浮点数) 时不执行转换并显示溢出。

状态字

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

实例

```
STL             解释
L     MD10      //将浮点数 (32 位，IEEE 754) 载入 ACCU 1-L 中。
RND+            //将浮点数 (32 位，IEEE 754) 转换为整型 (32 位)，并对结果取整。将输出存储在 ACCU 1 中。
T     MD20      //将结果 (长整数) 传送到 MD20。
```

转换前的值		转换后的值
MD10 = “100.5”	=> RND+ =>	MD20 = “+101”
MD10 = “-100.5”	=> RND+ =>	MD20 = “-100”

3.18 RND-　取整为低位长整数

格式

RND -

描述

RND- (32 位 IEEE 浮点数转换为 32 位整型) 将 ACCU 1 的内容解释为 32 位 IEEE 浮点数。该指令将 32 位 IEEE 浮点数转换为 32 位整型 (长整型)，并将结果取整为小于或等于所转换浮点数的最大整数 (IEEE 取整模式“取整为负无穷”)。如果超出允许范围，则状态位 OV 和 OS 被置位为 1。结果存储在累加器 1 中。

出现错误 (使用了不能表示为 32 位整数的 NaN 或浮点数) 时不执行转换并显示溢出。

状态字

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

实例

```
STL          解释
L     MD10   //将浮点数载入 ACCU 1-L。
RND-         //将浮点数 (32 位，IEEE 754) 转换为整型 (32 位)，并对结果取整。将结果存储在 ACCU 1 中。
T     MD20   //将结果 (长整数) 传送到 MD20。
```

转换前的值		转换后的值
MD10 = “100.5”	=> RND- =>	MD20 = “+100”
MD10 = “-100.5”	=> RND- =>	MD20 = “-100”

4 计数器指令

4.1 计数器指令概述

描述

计数器是 STEP 7 编程语言中用于计数的函数元素。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保留一个 16 位字。语句表指令集支持 256 个计数器。要获得有关 CPU 中有多少可用计数器的信息，请参考 CPU 的技术数据。

计数器指令是仅有的可以访问存储区的函数。

可使用下列计数器指令在此范围内改变计数值：

- FR 启用计数器 (释放)
- L 将当前计数器值载入 ACCU 1
- LC 将当前计数器值作为 BCD 码载入 ACCU 1
- R 将计数器复位
- S 设置计数器预设值
- CU 升值计数器
- CD 降值计数器

4.2 FR 启用计数器 (释放)

格式

FR <计数器>

地址	数据类型	存储区域	描述
<计数器>	COUNTER	C	计数器,其范围取决于CPU。

描述

当 RLO 从“0”跳转到“1”时，FR <计数器>会将边缘检测标记清零，该标记用于设置和选择寻址计数器的升值或降值计数。设置计数器或进行正常计数时不需启用计数器。也就是说，即使设置计数器的预设值、升值计数器或降值计数器的常数 RLO 为 1，在启用之后也不会再次执行这些指令。

状态字

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

实例

```
STL             解释
A    I 2.0      //检查输入 I 2.0 的信号状态。
FR   C3         //当 RLO 从 0 跳转到 1 时，启用计数器 C3。
```

4.3 L 将当前计数器值载入 ACCU 1

格式

L <计数器>

地址	数据类型	存储区域	描述
<计数器>	COUNTER	C	计数器，其范围取决于CPU。

描述

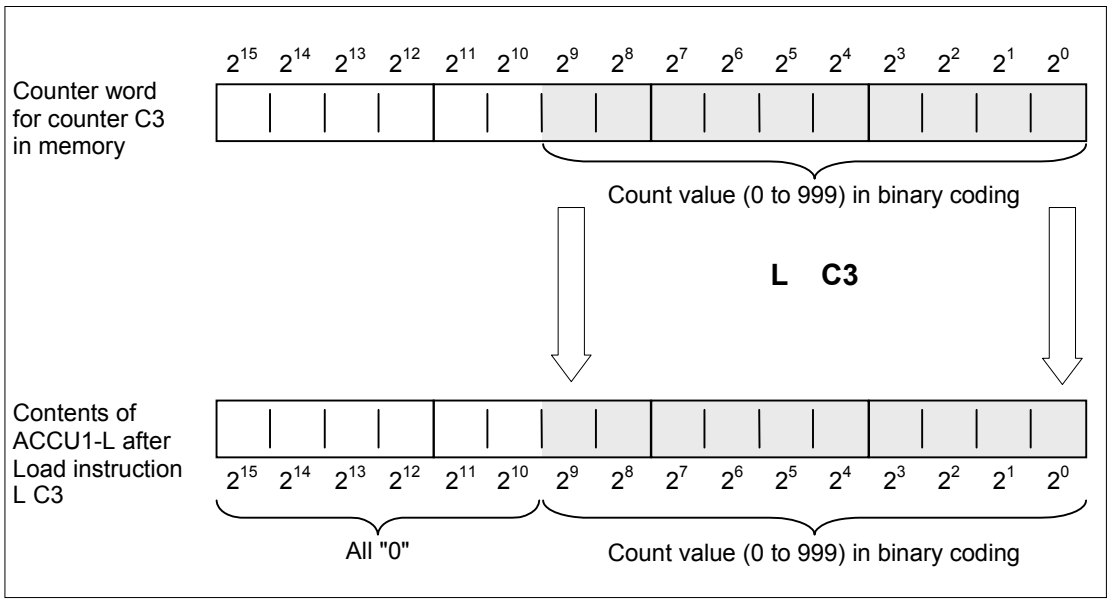
ACCU 1 的内容保存到 ACCU 2 中后，L <计数器> 将寻址计数器的当前计数值作为整型值载入 ACCU 1-L。

状态字

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

实例

```
STL          解释
L    C3      //以二进制格式将计数器 C3 的计数值载入 ACCU 1-L。
```



4.4 LC 将当前计数器值作为 BCD 码载入 ACCU 1

格式

LC <计数器>

地址	数据类型	存储区域	描述
<计数器>	COUNTER	C	计数器，其范围取决于CPU。

描述

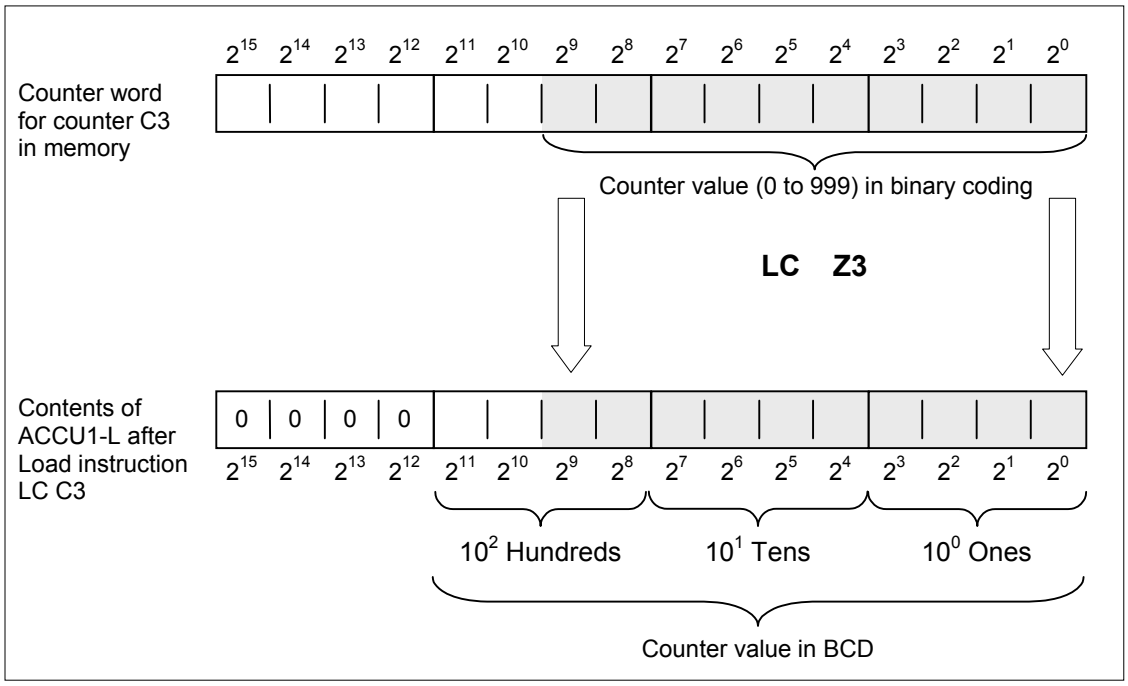
ACCU 1 的旧内容保存到 ACCU 2 中后，LC <计数器>将寻址计数器的计数值作为 BCD 码载入 ACCU 1。

状态字

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

实例

STL		解释
LC	C3	//以二进制编码的十进制格式将计数器 C3 的计数值载入 ACCU 1-L。



4.5 R 将计数器复位

格式

R <计数器>

地址	数据类型	存储区域	描述
<计数器>	COUNTER	C	要预置的计数器，其范围取决于 CPU。

描述

如果 RLO=1，R <计数器> 会将“0”载入寻址计数器中。

状态字

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

实例

```
STL             解释
A    I 2.3      //检查输入 I 2.3 的信号状态。
R    C3         //如果 RLO 从 0 跳转到 1，则将计数器 C3 复位到 0。
```

4.6 S 设置计数器预设值

格式

S <计数器>

地址	数据类型	存储区域	描述
<计数器>	COUNTER	C	要预置的计数器，其范围取决于 CPU。

描述

当 RLO 从“0”跳转到“1”时，S <计数器> 将 ACCU 1-L 的计数值载入寻址计数器。ACCU 1 中的计数值必须是介于“0”和“999”之间的 BCD 码形式的数值。

状态字

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

实例

```
STL             解释
A     I 2.3     //检查输入 I 2.3 的信号状态。
L     C#3       //将计数值 3 载入 ACCU 1-L。
S     C1        //如果 RLO 从 0 跳转到 1，则设置计数器 C1，以进行计数。
```

4.7 CU 升值计数器

格式

CU <计数器>

地址	数据类型	存储区域	描述
<计数器>	COUNTER	C	计数器，其范围取决于 CPU。

描述

当 RLO 从“0”跳转到“1”，并且计数小于“999”时，CU <计数器> 寻址计数器的计数值增 1。
当计数值达到上限“999”时，升值过程停止。RLO 的附加跳转无效，而且溢出的 OV 位不被置位。

状态字

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

实例

```
STL             解释
A    I 2.1      //如果在输入 I 2.1 有上升沿改变。
CU   C3         //当 RL0 从 0 跳转到 1 时，计数器 C3 的计数值增 1。
```

4.8 CD 降值计数器

格式

CD <计数器>

地址	数据类型	存储区域	描述
<计数器>	COUNTER	C	计数器，其范围取决于CPU。

描述

当 RLO 从“0”跳转到“1”，并且计数大于 0 时，CD <计数器>将寻址计数器的计数值减 1。当计数达到下限“0”时，降值过程停止。由于计数器不用负值计数，所以 RLO 的附加跳转无效。

状态字

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

实例

```
STL             解释
L     C#14      //计数器预设值。
A     I 0.1     //检测到 I 0.1 的上升沿后，预置计数器的值。
S     C1        //如果启用预置，则载入计数器 1 预设值。
A     I 0.0     //每当 I 0.0 有上升沿时，降值计数一次。
CD    C1        //当 RLO 根据输入 I 0.0 的状态从 0 跳转至 1 时，将计数器 C1 减 1。
AN    C1        //使用 C1 位进行零检测。
=     Q 0.0     //如果计数器 1 的值为零，则 Q 0.0 = 1。
```

5 数据块指令

5.1 数据块指令概述

描述

可以使用“打开数据块” (OPN) 指令将数据块作为共享数据块或实例数据块打开。程序本身可同时容纳一个打开的共享数据块和一个打开的实例数据块。

提供下列“数据块”指令：

- OPN 打开数据块
- CDB 交换共享数据块和实例 DB
- L DBLG 在 ACCU 1 中装载共享数据块的长度
- L DBNO 在 ACCU 1 中装载共享数据块的编号
- L DILG 在 ACCU 1 中装载实例 DB 的长度
- L DINO 在 ACCU 1 中装载实例 DB 的编号

5.2 OPN 打开数据块

格式

OPN <数据块>

地址	数据块类型	源地址
<数据块>	DB、DI	1 至 65535

指令描述

OPN <数据块> 将数据块作为共享数据块或实例数据块打开。可以同时打开一个共享数据块和一个实例数据块。

状态字

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

实例

```
STL             解释
OPN   DB10      //将数据块 DB10 作为共享数据块打开。
L     DBW35     //将已打开数据块的数据字 35 装载到 ACCU 1-L 中。
T     MW22      //将 ACCU 1-L 的内容传送到 MW22 中。
OPN   DI20      //将数据块 DB20 作为实例数据块打开。
L     DIB12     //将已打开实例数据块的数据字节 12 装载到 ACCU 1-L 中。
T     DBB37     //将 ACCU 1-L 的内容传送到已打开的共享数据块的数据字节 37。
```

5.3 CDB 交换共享数据块和实例 DB

格式

CDB

指令描述

CDB 用于交换共享数据块和实例数据块。该指令的作用是交换数据块寄存器。共享数据块变为实例数据块，或相反。

状态字

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

5.4 L DBLG 在 ACCU 1 中装载共享数据块的长度

格式

L DBLG

指令描述

L DBLG (装载共享数据块的长度) 会在 ACCU 1 的内容保存到 ACCU 2 中后，将共享数据块的长度装载到 ACCU 1 中。

状态字

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

实例

```
STL             解释
OPN   DB10      //将数据块 DB10 作为共享数据块打开。
L     DBLG      //装载共享数据块的长度 (DB10 的长度)。
L     MD10      //如果数据块足够长，则该长度作为用于比较的值。
<D
JC    ERRO      //如果长度小于 MD10 中的值，则跳转至 ERRO 跳转标签。
```

5.5 L DBNO 在 ACCU 1 中装载共享数据块的编号

格式

L DBNO

指令描述

L DBNO (装载共享数据块的编号) 会在 ACCU 1 的内容保存到 ACCU 2 中后，将打开的共享数据块的编号装载到 ACCU 1-L 中。

状态字

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

5.6 L DILG 在 ACCU 1 中装载实例 DB 的长度

格式

L DILG

指令描述

L DILG (装载实例数据块的长度) 会在 ACCU 1 的内容保存到 ACCU 2 中后，将实例数据块的长度装载到 ACCU 1-L 中。

状态字

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

实例

```
STL             解释
OPN   D120      //将数据块 DB20 作为实例数据块打开。
L     DILG      //装载实例数据块的长度 (DB20 的长度)。
L     MW10      //如果数据块足够长，则该长度作为用于比较的值。
<I
JC              //如果长度小于 MW10 中的值，则跳转到 ERRO 跳转标签。
```

5.7 L DINO 在 ACCU 1 中装载实例 DB 的编号

格式

L DINO

指令描述

L DINO (装载实例数据块的编号) 会在 ACCU 1 的内容保存到 ACCU 2 中后，将打开的实例数据块的编号装载到 ACCU 1 中。

状态字

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

6 逻辑控制指令

6.1 逻辑控制指令概述

描述

可使用跳转指令来控制逻辑流，允许程序中断其线性流，在一个不同点处继续进行扫描。可使用 LOOP 指令来多次调用一个程序段。

跳转或循环指令的地址为一个标签。跳转标签最多四个字符，且第一个字符必须为字母。跳转标签后必须带有一个冒号“:”，且在行中必须位于程序语句之前。

注意

请注意，对于 S7-300 CPU 程序而言，存在跳转指令时，跳转目标始终 (不用于 318– 2) 组成布尔逻辑字符串的**开头**。禁止在逻辑字符串中包含跳转目标。

可使用下列跳转指令来无条件中断正常的程序流：

- JU 无条件跳转
- JL 跳转到标签

下列跳转指令可根据上一个指令语句生成的逻辑运算结果 (RLO) 来中断程序的逻辑流。

- JC 当 RLO = 1 时跳转
- JCN 当 RLO = 0 时跳转
- JCB 当带 BR 位的 RLO = 1 时跳转
- JNB 当带 BR 位的 RLO = 0 时跳转

下列跳转指令可根据状态字中一个位的信号状态来中断程序中的逻辑流：

- JBI 当 BR = 1 时跳转
- JNBI 当 BR = 0 时跳转
- JO 当 OV = 1 时跳转
- JOS 当 OS = 1 时跳转

下列跳转指令可根据计算结果来中断程序中的逻辑流：

- JZ 当为零时跳转
- JN 当不为零时跳转
- JP 当为正时跳转
- JM 当为负时跳转
- JPZ 当为正或零时跳转
- JMZ 当为负或零时跳转
- JUO 无序时跳转

6.2 JU 无条件跳转

格式

JU <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

JU <跳转标签>中断线性程序扫描，并跳转到一个跳转目标，与状态字的内容无关。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的 -32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                          解释
        A       I 1.0
        A       I 1.2
        JC      DELE         //当 RLO=1 时，跳转到跳转标签 DELE。
        L       MB10
        INC     1
        T       MB10
        JU      FORW         //无条件跳转到跳转标签 FORW。
DELE:   L       0
        T       MB10
FORW:   A       I 2.1        //跳转到跳转标签 FORW 后，在此继续执行程序扫描。
```

6.3 JL 跳转到标签

格式

JL <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

JL <跳转标签> (通过跳转到列表进行跳转) 允许编程多次跳转。跳转目标列表 (最多 255 个条目) 从 **JL** 指令的下一行开始，到 JL 地址中引用的跳转标签的前一行结束。每个跳转目标由一个 **JU** 指令组成。跳转目标的数目 (0 - 255) 则从 ACCU 1-L-L 中获取。

只要 ACCU 的内容小于 **JL** 指令和跳转标签之间跳转目标的数目，**JL** 指令就跳转到 **JU** 指令中的一条。当 ACCU 1-L-L=0 时，跳转到第一条 JU 指令。当 ACCU 1-L-L=1 时，跳转到第二个 **JU** 指令，以此类推。如果跳转目标的数目太大，则在跳转到目标列表中的最后一条 **JU** 指令后，**JL** 指令跳转到第一条指令处。

跳转目标列表必须包含 **JU** 指令，该指令位于 JL 指令地址中引用的跳转标签之前。跳转列表内的所有其它指令都是非法的。

状态字

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

实例

```
STL                          解释
        L       MB0     //将跳转目标数目装载到 ACCU 1-L-L 中。
        JL      LSTX    //当 ACCU 1-L-L > 3 时的跳转目标。
        JU      SEG0    //当 ACCU 1-L-L = 0 时的跳转目标。
        JU      SEG1    //当 ACCU 1-L-L = 1 时的跳转目标。
        JU      COMM    //当 ACCU 1-L-L = 2 时的跳转目标
        JU      SEG3    //当 ACCU 1-L-L = 3 时的跳转目标
LSTX:   JU      COMM
SEG0:   *               //允许的指令
        *
        JU      COMM
SEG1:   *               //允许的指令
        *
        JU      COMM
SEG3:   *               //允许的指令。
        *
        JU      COMM
COMM:   *
        *
```

6.4 JC 当 RLO = 1 时跳转

格式

JC <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当逻辑运算的结果为 1 时，**JC <跳转标签>**就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。由跳转标签确定跳转目标。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

当逻辑运算的结果为 0 时，不执行跳转。将 RLO 设置为 1，继续对下一个语句执行程序扫描。

状态字

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

实例

```
STL                     解释
        A    I 1.0
        A    I 1.2
        JC   JOVR       //当 RLO=1 时，跳转到跳转标签 JOVR。
        L    IW8        //当不执行跳转时，在此继续执行程序扫描。
        T    MW22
JOVR:   A    I 2.1      //跳转到跳转标签 JOVR 后，在此继续执行程序扫描。
```

6.5 JCN 当 RLO = 0 时跳转

格式

JCN <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当逻辑运算的结果为 0 时，**JCN <跳转标签>** 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

当逻辑运算的结果为 1 时，不执行跳转。继续对下一个语句执行程序扫描。

状态字

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

实例

```
STL                          解释
      A    I 1.0
      A    I 1.2
      JCN  JOVR      //当 RLO=0 时，跳转到跳转标签 JOVR。
      L    IW8       //当不执行跳转时，在此继续执行程序扫描。
      T    MW22
JOVR: A    I 2.1     //跳转到跳转标签 JOVR 后，在此继续执行程序扫描。
```

6.6 JCB 当带 BR 位的 RLO = 1 时跳转

格式

JCB <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当逻辑运算的结果为 1 时，**JCB <跳转标签>**就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

当逻辑运算的结果为 0 时，不执行跳转。将 RLO 设置为 1，继续对下一个语句执行程序扫描。

将 RLO 复制到 BR 中，以执行 **JCB <跳转标签>**指令，而与 RLO 的状态无关。

状态字

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

实例

```
STL                          解释
      A    I 1.0
      A    I 1.2
      JCB  JOVR     //当 RLO=1 时，跳转到跳转标签 JOVR。将 RLO 位的内容复制到 BR 位。
      L    IW8      //当不执行跳转时，在此继续执行程序扫描。
      T    MW22
JOVR: A    I 2.1    //跳转到跳转标签 JOVR 后，在此继续执行程序扫描。
```

6.7 JNB 当带 BR 位的 RLO = 0 时跳转

格式

JNB <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当逻辑运算的结果为 0 时，**JNB <跳转标签>** 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

当逻辑运算的结果为 1 时，不执行跳转。将 RLO 设置为 1，继续对下一个语句执行程序扫描。

当存在一个 **JNB <跳转标签>** 指令时，将 RLO 复制到 BR，而与 RLO 的状态无关。

状态字

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

实例

```
STL                          解释
      A    I 1.0
      A    I 1.2
      JNB  JOVR     //当 RLO=0 时，跳转到跳转标签 JOVR。将 RLO 位的内容复制到 BR 位中。
      L    IW8      //当不执行跳转时，在此继续执行程序扫描。
      T    MW22
JOVR: A    I 2.1    //跳转到跳转标签 JOVR 后，在此继续执行程序扫描。
```

6.8 JBI 当 BR = 1 时跳转

格式

JBI <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 BR 为 1 时，**JBI <跳转标签>** 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。跳转标签最多四个字符，且第一个字符必须为字母。跳转标签后必须带有一个冒号“**:**”，且在行中必须位于程序语句之前。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

6.9 JNBI 当 BR = 0 时跳转

格式

JNBI <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 BR 为 0 时，**JNBI <跳转标签>** 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

6.10 JO 当 OV = 1 时跳转

格式

JO <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 OV 为 1 时，**JO <跳转标签>** 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。在一个组合的算术指令中，在每个单独的算术指令后检查是否发生溢出，以确保每个中间结果都位于允许范围内，或使用指令 **JOS**。

状态字

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

实例

```
STL                          解释
        L     MW10
        L     3
        *I                   //将 MW10 的内容乘以“3”。
        JO    OVER           //当结果超出最大范围 (OV=1) 时，跳转。
        T     MW10           //当不执行跳转时，在此继续执行程序扫描。
        A     M 4.0
        R     M 4.0
        JU    NEXT
OVER:   AN    M 4.0          //跳转到跳转标签 OVER 后，在此继续执行程序扫描。
        S     M 4.0
NEXT:   NOP 0                //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.11 JOS 当 OS = 1 时跳转

格式

JOS <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 OS 为 1 时，**JOS <跳转标签>** 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                              解释
          L       IW10
          L       MW12
          *I
          L       DBW25
          +I
          L       MW14
          -I
          JOS     OVER    //在计算 OS=1 期间，如果三个指令中有一个发生溢出，
                          //则跳转。 (参见注意事项)。
          T       MW16    //当不执行跳转时，在此继续执行程序扫描。
          A       M 4.0
          R       M 4.0
          JU      NEXT
OVER:     AN      M 4.0   //跳转到跳转标签 OVER 后，在此继续执行程序扫描。
          S       M 4.0
NEXT:     NOP 0           //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

注意

在这种情况下，不要使用 **JO** 指令。当发生溢出时，**JO** 指令只检查上一条 **-I** 指令。

6.12 JZ 当为零时跳转

格式

JZ <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 CC 1 = 0 且 CC 0 = 0 时，**JZ <跳转标签>** (当结果 = 0 时跳转) 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                     解释
        L       MW10
        SRW     1
        JZ      ZERO    //当被移出的位 = 0 时，跳转到跳转标签 ZERO。
        L       MW2     //当不执行跳转时，在此继续执行程序扫描。
        INC     1
        T       MW2
        JU      NEXT
ZERO:   L       MW4     //跳转到跳转标签 ZERO 后，在此继续执行程序扫描。
        INC     1
        T       MW4
NEXT:   NOP 0           //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.13 JN 当不为零时跳转

格式

JN <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当由状态位 CC 1 和 CC 0 指示的结果大于或小于零时 (CC 1=0/CC 0=1 或 CC 1=1/CC 0=0)，**JN <跳转标签>** (当结果 <> 0 时跳转) 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                          解释
        L       IW8
        L       MW12
        XOW
        JN      NOZE    //当 ACCU 1-L 的内容不等于零时跳转。
        AN      M 4.0   //当不执行跳转时，在此继续执行程序扫描。
        S       M 4.0
        JU      NEXT
NOZE:   AN      M 4.1   //跳转到跳转标签 NOZE 后，在此继续执行程序扫描。
        S       M 4.1
NEXT:   NOP 0           //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.14 JP 当为正时跳转

格式

JP <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 CC 1 = 1 且 CC 0 = 0 时，**JP <跳转标签>** (当结果 < 0 时跳转) 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                         解释
        L       IW8
        L       MW12
        -I                  //IW8 的内容减去 MW12 的内容。
        JP      POS         //当结果 > 0 (即 ACCU 1 > 0) 时跳转。
        AN      M 4.0       //当不执行跳转时，在此继续执行程序扫描。
        S       M 4.0
        JU      NEXT
pos:    AN      M 4.1       //跳转到跳转标签 POS 后，在此继续执行程序扫描。
        S       M 4.1
NEXT:   NOP 0               //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.15 JM 当为负时跳转

格式

JM <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 CC 1 = 0 且 CC 0 = 1 时，**JM <跳转标签>** 当结果 < 0 时跳转) 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                     解释
        L       IW8
        L       MW12
        -I              //IW8 的内容减去 MW12 的内容。
        JM      NEG     //当结果 < 0 (即，ACCU 1 的内容 < 0) 时跳转。
        AN      M 4.0   //当不执行跳转时，在此继续执行程序扫描。
        S       M 4.0
        JU      NEXT
neg:    AN      M 4.1   //跳转到跳转标签 NEG 后，在此继续执行程序扫描。
        S       M 4.1
NEXT:   NOP 0           //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.16 JPZ 当为正或零时跳转

格式

JPZ <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当由状态位 CC 1 和 CC 0 指示的结果大于或等于零时 (CC 1=0/CC 0=0 或 CC 1=1/CC 0=0)，
JPZ <跳转标签> (当结果 >= 0 时跳转) 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                          解释
        L       IW8
        L       MW12
        -I                   //IW8 的内容减去 MW12 的内容。
        JPZ     REG0         //当结果 >=0 (即，ACCU 1 的内容 >= 0) 时跳转。
        AN      M 4.0        //当不执行跳转时，在此继续执行程序扫描。
        S       M 4.0
        JU      NEXT
REG0:   AN      M 4.1        //跳转到跳转标签 REG0 后，在此继续执行程序扫描。
        S       M 4.1
NEXT:   NOP 0                //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.17 JMZ 当为负或零时跳转

格式

JMZ <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当由状态位 CC 1 和 CC 0 指示的结果小于或等于零时 (CC 1=0/CC 0=0 或 CC 1=0/CC 0=1)，**JMZ <跳转标签>** (当结果 <= 0 时跳转) 就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

实例

```
STL                          解释
        L       IW8
        L       MW12
        -I                   //IW8 的内容减去 MW12 的内容。
        JMZ     RGE0         //当结果 <=0 时跳转 (即，ACCU 1 的内容 <= 0)。
        AN      M 4.0        //当不执行跳转时，在此继续执行程序扫描。
        S       M 4.0
        JU      NEXT
RGE0:   AN      M 4.1        //跳转到跳转标签 RGE0 后，在此继续执行程序扫描。
        S       M 4.1
NEXT:   NOP 0                //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.18 JUO 无序时跳转

格式

JUO <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

当状态位 CC 1 = 1 且 CC 0 = 1 时，**JUO <跳转标签>**就中断线性程序扫描，并跳转到一个跳转目标。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

在下列情况下，状态位 CC 1 = 1 且 CC 0 = 1：

- 发生被零除时
- 使用了非法指令时
- 浮点比较的结果为“无序”，即，使用了一种无效格式时。

状态字

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

实例

```
STL                              解释
        L       MD10
        L       ID2
        /D                       //MD10 的内容除以 ID2 的内容
        JUO     ERRO             //当被零除时跳转 (即，ID2 = 0)。
        T       MD14             //当不执行跳转时，在此继续执行程序扫描。
        A       M 4.0
        R       M 4.0
        JU      NEXT
ERRO:   AN      M 4.0            //跳转到跳转标签 ERROR 后，在此继续执行程序扫描。
        S       M 4.0
NEXT:   NOP 0                    //跳转到跳转标签 NEXT 后，在此继续执行程序扫描。
```

6.19 LOOP 循环

格式

LOOP <跳转标签>

地址	描述
<跳转标签>	跳转目标的符号名。

描述

LOOP <跳转标签> (对 ACCU 1-L 进行减 1 操作，并在 ACCU 1-L <> 0 时跳转) 可简化循环编程。ACCU 1-L 中包含循环计数器。指令跳转到指定的跳转目标。只要 ACCU 1-L 的内容不等于 0，就一直执行跳转。线性程序扫描在跳转目标处继续执行。跳转目标由跳转标签确定。允许向前跳转和向后跳转。只能在一个块内执行跳转，即，跳转指令和跳转目标必须位于同一个块内。跳转目标在该块内必须唯一。最大跳转距离为程序代码的-32768 或+32767 个字。可以跳过的实际语句的最大数目取决于程序中使用的语句组合 (单字、双字或三字语句)。

状态字

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

计算因子为 5 的实例

```
STL                     解释
        L       1#1     //将整型常数 (32 位) 装载到 ACCU 1 中。
        T       MD20    //将 ACCU1 的内容传送给 MD20 (初始化)。
        L       5       //将循环周期的数目装载到 ACCU 1-L 中。
NEXT:   T       MW10    //跳转标签 = 循环开始/将 ACCU 1-L 传送给循环计数器。
        L       MD20
        *       D       //MD20 的当前内容乘以 MB10 的当前内容。
        T       MD20    //将相乘结果传送给 MD20。
        L       MW10    //将循环计数器的内容装载到 ACCU 1 中。
        LOOP    NEXT    //对 ACCU 1 的内容进行减 1 操作，当 ACCU 1-L > 0 时，
                        //跳转到 NEXT 跳转标签。
        L       MW24    //完成循环后，在此继续执行程序扫描。
        L       200
        >I
```

7 整数运算指令

7.1 整数算术指令概述

描述

算术指令组合累加器 1 和累加器 2 的内容。对于带有两个累加器的 CPU，累加器 2 的内容保持不变。

对于带 4 个累加器的 CPU，将累加器 3 的内容复制到累加器 2 中，将累加器 4 的内容复制到累加器 3 中。累加器 4 中较早的内容保持不变。

使用整数算术，可以对**两个整数** (16 位和 32 位) 执行下列运算：

- +I ACCU 1 + ACCU 2，整型 (16 位)
- -I ACCU 2 - ACCU 1，整型 (16 位)
- *I ACCU 1 * ACCU 2，整型 (16 位)
- /I ACCU 2 / ACCU 1，整型 (16 位)
- + 整型常数相加 (16、32 位)
- +D ACCU 1 + ACCU 2，长整型 (32 位)
- -D ACCU 2 - ACCU 1，长整型 (32 位)
- *D ACCU 1 * ACCU 2，长整型 (32 位)
- /D ACCU 2 / ACCU 1，长整型 (32 位)
- MOD 除法余数，长整型 (32 位)

7.2 使用整数算术指令时得出状态字的位数值

描述

整数算术指令影响状态字中的下列位：CC1 和 CC0、OV 和 OS。

下表显示了状态字中用于表示整数指令结果 (16 位和 32 位) 的位的信号状态：

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：被 0 除	1	1	1	1

7.3 +I ACCU 1 + ACCU 2，整型 (16 位)

格式

+I

描述

+I (16 位整数相加) 将 ACCU 1-L 的内容与 ACCU 2-L 中的内容相加，并将结果存储在 ACCU 1-L 中。将 ACCU 1-L 和 ACCU 2-L 的内容解释为 16 位整数。执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。在发生溢出/下溢时，该指令生成一个 16 位整数，而不是一个 32 位整数。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
总和 = 0	0	0	0	-
-32768 <= 总和 < 0	0	1	0	-
32767 >= 总和 > 0	1	0	0	-
总和 = -65536	0	0	1	1
65534 >= 总和 > 32767	0	1	1	1
-65535 <= 总和 < -32768	1	0	1	1

实例

```
STL                 解释
L     IW10          //将 IW10 的数值装载到 ACCU 1-L 中。
L     MW14          //将 ACCU 1-L 的内容装载到 ACCU 2-L 中。将 MW14 的数值装载到
                    //ACCU 1-L 中。
+I                  //ACCU 2-L 和 ACCU 1-L 相加；结果存储在 ACCU 1-L 中。
T     DB1.DBW25     //将 ACCU 1-L (结果) 的内容传送到 DB1 的 DBW25 中。
```

7.4 -I ACCU 2 - ACCU 1，整型 (16 位)

格式

-I

描述

-I (16 位整数相减) 从 ACCU 2-L 的内容中减去 ACCU 1-L 的内容，并将结果存储在 ACCU 1-L 中。将 ACCU 1-L 和 ACCU 2-L 的内容解释为 16 位整数。执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。在发生溢出/下溢时，该指令生成一个 16 位整数，而不是一个 32 位整数。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
差 = 0	0	0	0	-
-32768 <= 差 < 0	0	1	0	-
32767 >= 差 > 0	1	0	0	-
65535 >= 差 > 32767	0	1	1	1
-65535 <= 差 < -32768	1	0	1	1

实例

```
STL             解释
L     IW10      //将 IW10 的数值装载到 ACCU 1-L 中。
L     MW14      //将 ACCU 1-L 的内容装载到 ACCU 2-L 中。将 MW14 的数值装载到
                //ACCU 1-L 中。
-I              //从 ACCU 2-L 中减去 ACCU 1-L；结果存储在 ACCU 1-L 中。
T     DB1.DBW25 //将 ACCU 1-L (结果) 的内容传送到 DB1 的 DBW25 中。
```

7.5 *I ACCU 1 * ACCU 2，整型 (16 位)

格式

***I**

描述

***I** (乘以 16 位整数) ACCU 2-L 的内容乘以 ACCU 1-L 的内容。将 ACCU 1-L 和 ACCU 2-L 的内容解释为 16 位整数。结果作为一个 32 位整数存储在 ACCU 1 中。当状态字的位为 OV1 = 1 和 OS = 1 时，表示结果超出 16 位整数范围。

执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
乘积 = 0	0	0	0	-
-32768 <= 乘积 < 0	0	1	0	-
32767 >= 乘积 > 0	1	0	0	-
1073741824 >= 乘积 > 32767	1	0	1	1
-1073709056<=乘积<-32768	0	1	1	1

实例

```
STL             解释
L     IW10      //将 IW10 的数值装载到 ACCU 1-L 中。
L     MW14      //将 ACCU 1-L 的内容装载到 ACCU 2-L 中。将 MW14 的内容装载到 ACCU 1-L 中。
*I              //ACCU 2-L 和 ACCU 1-L 相乘；将结果存储在 ACCU 1 中。
T     DB1.DBD25 //将 ACCU 1 (结果) 的内容传送到 DB1 的 DBD25 中。
```

7.6 /I ACCU 2 / ACCU 1，整型 (16 位)

格式

/I

描述

/I (16 位整数相除) ACCU 2-L 的内容除以 ACCU 1-L 的内容。将 ACCU 1-L 和 ACCU 2-L 的内容解释为 16 位整数。结果存储在 ACCU 1 中，包含两个 16 位整数，即商和余数。在 ACCU 1-L 中存储商，在 ACCU 1-H 中存储余数。执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
商 = 0	0	0	0	-
-32768 <= 商 < 0	0	1	0	-
32767 >= 商 > 0	1	0	0	-
商 = 32768	1	0	1	1
被零除	1	1	1	1

实例

```
STL             解释
L     IW10      //将 IW10 的数值装载到 ACCU 1-L 中。
L     MW14      //将 ACCU 1-L 的内容装载到 ACCU 2-L 中。将 MW14 的数值装载到
                //ACCU 1-L 中。
/I              //ACCU 2-L 除以 ACCU 1-L；结果存储在 ACCU 1 中：ACCU 1-L：商、
                //ACCU 1-H：余数
T     MD20      //将 ACCU 1 (结果) 的内容传送到 MD20 中。
```

实例：13 除以 4

执行指令 (IW10) 前，ACCU 2-L 的内容：	“13”
执行指令 (MW14) 前，ACCU 1-L 的内容：	“4”
指令 /I (ACCU 2-L / ACCU 1-L)：	“13/4”
执行指令后，ACCU 1-L 的内容 (商)：	“3”
执行指令后，ACCU 1-H 的内容 (余数)：	“1”

7.7 + 整型常数相加 (16、32 位)

格式

+ <整型常数>

地址	数据类型	描述
<整型常数>	(16 位或 32 位整数)	要相加的整数

描述

+ <整型常数>将整型常数加到 ACCU 1 的内容中，并将结果存储在 ACCU 1 中。执行该指令时不涉及，也不会影响状态字的位。

+ <16 位整型常数>：将一个 16 位整型常数 (范围为-32768 至+32767) 加到 ACCU 1-L 的内容中，然后将结果存储在 ACCU 1-L 中。

+ <32 位整型常数>：将一个 32 位整型常数 (范围为-2,147,483,648 至 2,147,483,647) 加到 ACCU 1 的内容中，然后将结果存储在 ACCU 1 中。

状态字

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

实例 1

```
STL                 解释
L     IW10          //将 IW10 的数值装载到 ACCU 1-L 中。
L     MW14          //将 ACCU 1-L 的内容装载到 ACCU 2-L 中。//将 MW14 的数值装载到 ACCU 1-L 中。
+I                  //ACCU 2-L 和 ACCU 1-L 相加；结果存储在 ACCU 1-L 中。
+     25            //ACCU 1-L 与 25 相加；结果存储在 ACCU 1-L 中。
T     DB1.DBW25     //将 ACCU 1-L (结果) 的内容传送到 DB1 的 DBW25 中。
```

实例 2

```
STL                 解释
L     IW12
L     IW14
+     100           //ACCU 1-L 与 100 相加；结果存储在 ACCU 1-L 中。
>I                  //当 ACCU 2 > ACCU 1 或 IW12 > (IW14 + 100) 时
JC    NEXT          //则条件跳转到跳转标签 NEXT。
```

实例 3

```
STL                 解释
L     MD20
L     MD24
+D                  //ACCU 1 和 ACCU 2 相加；将结果存储在 ACCU 1 中。
+     1# -200       //ACCU 1 和-200 相加；将结果存储在 ACCU 1 中。
T     MD28
```

7.8 +D ACCU 1 + ACCU 2，长整型 (32 位)

格式

+D

描述

+D (32 位整数相加) 将 ACCU 1 的内容与 ACCU 2 中的内容相加，并将结果存储在 ACCU 1 中。将 ACCU 1 和 ACCU 2 的内容解释为 32 位整数。执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
总和 = 0	0	0	0	-
-2147483648 <= 总和 < 0	0	1	0	-
2147483647 >= 总和 > 0	1	0	0	-
总和 = -4294967296	0	0	1	1
4294967294 >= 总和 > 2147483647	0	1	1	1
-4294967295 <= 总和 < -2147483648	1	0	1	1

实例

```
STL             解释
L     ID10      //将 ID10 的值装载到 ACCU 1 中。
L     MD14      //将 ACCU 1 的内容装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
+D              //ACCU 2 和 ACCU 1 相加，将结果存储在 ACCU 1 中。
T     DB1.DBD25 //将 ACCU 1 (结果) 的内容传送到 DB1 的 DBD25 中。
```

7.9 -D ACCU 2 - ACCU 1，长整型 (32 位)

格式

-D

描述

-D (32 位整数相减) 从 ACCU 2 的内容中减去 ACCU 1 的内容，并将结果存储在 ACCU 1 中。将 ACCU 1 和 ACCU 2 的内容解释为 32 位整数。执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
差 = 0	0	0	0	-
-2147483648 <= 差 < 0	0	1	0	-
2147483647 >= 差 > 0	1	0	0	-
4294967295 >= 差 > 2147483647	0	1	1	1
-4294967295 <= 差 < -2147483648	1	0	1	1

实例

```
STL                 解释
L     ID10          //将 ID10 的值装载到 ACCU 1 中。
L     MD14          //将 ACCU 1 的内容装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
-D                  //ACCU 2 减 ACCU 1，将结果存储在 ACCU 1 中。
T     DB1.DBD25     //将 ACCU 1 (结果) 的内容传送到 DB1 的 DBD25 中。
```

7.10 *D ACCU 1 * ACCU 2，长整型 (32 位)

格式

***D**

描述

***D** (乘以 32 位整数) ACCU 2 的内容乘以 ACCU 1 的内容。将 ACCU 1 和 ACCU 2 的内容解释为 32 位整数。结果作为一个 32 位整数存储在 ACCU 1 中。当状态字的位为 OV1 = 1 和 OS = 1 时，表示结果超出 32 位整数范围。

执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
乘积 = 0	0	0	0	-
-2147483648 <= 乘积 < 0	0	1	0	-
2147483647 >= 乘积 > 0	1	0	0	-
乘积 > 2147483647	1	0	1	1
乘积 < -2147483648	0	1	1	1

实例

```
STL                 解释
L     ID10          //将 ID10 的值装载到 ACCU 1 中。
L     MD14          //将 ACCU 1 的内容装载到 ACCU 2 中。将 MD14 的内容装载到 ACCU 1 中。
*D                  //ACCU 2 和 ACCU 1 相乘，将结果存储在 ACCU 1 中。
T     DB1.DBD25     //将 ACCU 1 (结果) 的内容传送到 DB1 的 DBD25 中。
```

7.11 /D ACCU 2 / ACCU 1，长整型 (32 位)

格式

/D

描述

/D (32 位整数相除) ACCU 2 的内容除以 ACCU 1。将 ACCU 1 和 ACCU 2 的内容解释为 32 位整数。在 ACCU 1 中存储该指令的结果。结果只给出商，不给出余数。 (指令 MOD 可用于获取余数。)

执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
商 = 0	0	0	0	-
-2147483648 <= 商 < 0	0	1	0	-
2147483647 >= 商 > 0	1	0	0	-
商 = 2147483648	1	0	1	1
被零除	1	1	1	1

实例

```
STL          解释
L     ID10   //将 ID10 的值装载到 ACCU 1 中。
L     MD14   //将 ACCU 1 的内容装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
/D           //ACCU 2 除以 ACCU 1；结果存储在 ACCU 1 中 (商)。
 T    MD20   //将 ACCU 1 (结果) 的内容传送到 MD20 中。
```

实例：13 除以 4

执行指令 (ID10) 前，ACCU 2 的内容： “13”
执行指令 (MD14) 前，ACCU 1 的内容： “4”
指令/D (ACCU 2 / ACCU 1)： “13/4”
执行指令后，ACCU 1 的内容 (商)： “3”

7.12 MOD 除法余数，长整型 (32 位)

格式

MOD

描述

MOD (32 位整数除法的余数) ACCU 2 的内容除以 ACCU 1 的内容。将 ACCU 1 和 ACCU 2 的内容解释为 32 位整数。在 ACCU 1 中存储该指令的结果。结果只给出除法的余数，不给出商。(指令/D 可用于获取商。)

执行指令时既不考虑也不会影响 RLO。作为指令运算结果的一个功能，将对状态字的位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

状态字

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

生成状态位	CC 1	CC 0	OV	OS
余数 = 0	0	0	0	-
-2147483648 <= 余数 < 0	0	1	0	-
2147483647 >= 余数 > 0	1	0	0	-
被零除	1	1	1	1

实例

```
STL                 解释
L     ID10          //将 ID10 的值装载到 ACCU 1 中。
L     MD14          //将 ACCU 1 的内容装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
MOD                 //ACCU 2 除以 ACCU 1；结果存储在 ACCU 1 中 (余数)。
T     MD20          //将 ACCU 1 (结果) 的内容传送到 MD20 中。
```

实例：13 除以 4

执行指令 (ID10) 前，ACCU 2 的内容： “13”
执行指令 (MD14) 前，ACCU 1 的内容： “4”
指令 MOD (ACCU 2 / ACCU 1)： “13/4”
执行指令后，ACCU 1 的内容 (余数)： “1”

8 浮点型数学运算指令

8.1 浮点运算指令概述

描述

该运算指令组合累加器 1 和 2 的内容。对于带有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2 中，并将累加器 4 的内容复制到累加器 3 中。累加器 4 中较早的内容保持不变。

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：

- +R　　ACCU 1 加 ACCU
- -R　　ACCU 1 减 ACCU 2
- *R　　ACCU 1 乘 ACCU 2
- /R　　ACCU 2 除 ACCU 1

利用浮点运算，可用**一个 32 位 IEEE 浮点数**执行下列运算：

- ABS　　绝对值
- SQR　　计算平方
- SQRT　　计算平方根
- EXP　　计算指数值
- LN　　计算自然对数
- SIN　　计算角的正弦值
- COS　　计算角的余弦值
- TAN　　计算角的正切值
- ASIN　　计算反正弦值
- ACOS　　计算反余弦值
- ATAN　　计算反正切值

8.2 使用浮点运算指令时得出状态字的位数值

描述

基本运算类型会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 (空)	0	0	0	*
-3.402823E+38 < 结果 < -1.175494E-38 (负数)	0	1	0	*
+1.175494E-38 < 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 < 结果 < - 1.401298E-45 (负数)	0	0	1	1
下溢 +1.401298E-45 < 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 浮点型数学运算指令：基本

8.3.1 +R 将 ACCU 1 和 ACCU 2 作为浮点数 (32 位 IEEE 754) 相加

格式

+R

指令描述

+R (加 32 位 IEEE 浮点数) 将累加器 1 与累加器 2 的内容相加，并将结果存储到累加器 1 中。将累加器 1 和累加器 2 的内容解释为 32 位 IEEE 浮点数。执行指令时既不考虑也不会影响 RLO。结果会对状态位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-无穷大	0	1	1	1	溢出
-qNaN	1	1	1	1	

状态字

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

实例

STL		解释
OPN	DB10	
L	ID10	//将 ID10 的值装载到 ACCU 1 中。
L	MD14	//将 ACCU 1 的值装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
+R		//ACCU 2 和 ACCU 1 相加，将结果存储在 ACCU 1 中。
T	DBD25	//将 ACCU 1 的内容 (结果) 传送到 DB10 的 DBD25 中。

8.3.2 -R 以浮点数 (32 位 IEEE 754) 的形式从 ACCU 2 减去 ACCU 1

格式

-R

描述

-R (减 32 位 IEEE 浮点数) 从累加器 2 减去累加器 1 的内容，并将结果存储到累加器 1 中。将累加器 1 和累加器 2 的内容解释为 32 位 IEEE 浮点数。结果存储在累加器 1 中。执行指令时既不考虑也不会影响 RLO。结果会对状态位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-无穷大	0	1	1	1	溢出
-qNaN	1	1	1	1	

状态字

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

实例

```
STL             解释
OPN   DB10
L     ID10      //将 ID10 的值装载到 ACCU 1 中。
L     MD14      //将 ACCU 1 的值装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
-R              //ACCU 2 减 ACCU 1，将结果存储在 ACCU 1 中。
T     DBD25     //将 ACCU 1 的内容 (结果) 传送到 DB10 的 DBD25 中。
```

8.3.3 *R 将 ACCU 1 与 ACCU 2 作为浮点数 (32 位 IEEE 754) 相乘

格式

***R**

指令描述

***R** (乘 32 位 IEEE 浮点数) 累加器 2 与累加器 1 的内容相乘。将累加器 1 和累加器 2 的内容解释为 32 位 IEEE 浮点数。结果作为 32 位 IEEE 浮点数存储在累加器 1 中。执行指令时既不考虑也不会影响 RLO。结果会对状态位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。累加器 4 的内容保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-无穷大	0	1	1	1	溢出
-qNaN	1	1	1	1	

状态字

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

实例

```
STL             解释
OPN   DB10
L     ID10      //将 ID10 的值装载到 ACCU 1 中。
L     MD14      //将 ACCU 1 的值装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
*R              //ACCU 2 和 ACCU 1 相乘，将结果存储在 ACCU 1 中。
T     DBD25     //将 ACCU 1 的内容 (结果) 传送到 DB10 的 DBD25 中。
```

8.3.4 /R 以浮点数 (32 位 IEEE 754) 的形式用 ACCU 1 除 ACCU 2

格式

/R

指令描述

/R (除 32 位 IEEE 浮点数) 将累加器 2 的内容除以累加器 1 的内容。将累加器 1 和累加器 2 的内容解释为 32 位 IEEE 浮点数。执行指令时既不考虑也不会影响 RLO。结果会对状态位 CC 1、CC 0、OS 和 OV 进行设置。

对于具有两个累加器的 CPU，累加器 2 的内容保持不变。

对于具有四个累加器的 CPU，则会将累加器 3 的内容复制到累加器 2，并将累加器 4 的内容复制到累加器 3。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-无穷大	0	1	1	1	溢出
-qNaN	1	1	1	1	

状态字

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

实例

```
STL             解释
OPN   DB10
L     ID10      //将 ID10 的值装载到 ACCU 1 中。
L     MD14      //将 ACCU 1 的内容装载到 ACCU 2 中。将 MD14 的数值装载到 ACCU 1 中。
/R              //ACCU 2 除以 ACCU 1；将结果存储在 ACCU 1 中。
T     DBD20     //将 ACCU 1 的内容 (结果) 传送到 DB10 的 DBD20 中。
```

8.3.5 ABS 浮点数 (32 位 IEEE 754) 的绝对值

格式

ABS

描述

ABS (32 位 IEEE FP 的绝对值) 在 ACCU 1 中计算浮点数 (32 位 IEEE 浮点数) 的绝对值。结果存储在累加器 1 中。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL             解释
L     ID8       //将值载入 ACCU 1 (实例：ID8 = -1.5E+02).
ABS             //生成绝对值；将结果存储在 ACCU 1 中。
T     MD10      //将结果传送到 MD10 (实例：结果 = 1.5E+02).
```

8.4 浮点型数学运算指令：扩充

8.4.1 SQR 计算浮点数 (32 位) 的平方

格式

SQR

指令描述

SQR (生成 IEEE 754 32 位浮点数的平方) 计算 ACCU 1 中的浮点数 (32 位，IEEE 754) 的平方。结果存储在累加器 1 中。此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-qNaN	1	1	1	1	

实例

```
STL                     解释
      OPN   DB17        //打开数据块 DB17。
      L     DBD0        //将数据双字 DBD0 中的值载入到 ACCU 1 中。//(该值必须为浮点数格式。)
      SQR               //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的平方。将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是“0”。
      JC    OK          //如果在执行 SQR 指令期间未出错，
                        //则跳转到 OK 跳转标签。
BEU                     //如果执行 SQR 指令时出错，则块无条件结束。
OK:   T     DBD4        //将结果从 ACCU 1 传送到数据块中的双字 DBD4。
```

8.4.2 SQRT 计算浮点数 (32 位) 的平方根

格式

SQRT

指令描述

SQRT (生成 32 位，IEEE 754 浮点数的平方根) 计算 ACCU 1 中的浮点数 (32 位，IEEE 754) 的平方根。结果存储在累加器 1 中。输入值必须大于或等于零。然后得出结果也是正数。唯一的例外是-0 的平方根是-0。此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-零	0	0	0	-	
-qNaN	1	1	1	1	

实例

```
STL                     解释
      L     MD10        //将存储器双字 MD10 中的值载入到 ACCU 1 中。//(该值必须为浮点数格式。)
      SQRT              //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的平方根。将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是“0”。
      JC    OK          //如果在执行 SQRT 指令期间未出错，
                        //则跳转到 OK 跳转标签。
BEU                     //如果执行 SQRT 指令时出错，则块无条件结束。
OK:   T     MD20        //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.3 EXP 计算浮点数 (32 位) 的指数值

格式

EXP

指令描述

EXP (生成 32 位，IEEE 754 浮点数的指数值) 计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的指数值 (以 *e* 为底的指数值)。结果存储在累加器 1 中。此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-qNaN	1	1	1	1	

实例

```
STL                     解释
      L     MD10        //将存储器双字 MD10 中的值载入到 ACCU 1 中。//(该值必须为浮点数格式。)
      EXP               //在 ACCU 1 中计算浮点数 (32 位，IEEE 754) 的指数值，
                        //底数为 e。将结果存储在 ACCU 1 中。将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是"0"。
      JC    OK          //如果在执行 EXP 指令期间未出错，
                        //则跳转到 OK 跳转标签。
BEU                     //如果执行 EXP 指令时出错，则块无条件结束。
OK:   T     MD20        //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.4 LN 计算浮点数 (32 位) 的自然对数

格式

LN

指令描述

LN (生成 IEEE 754 32 位浮点数的自然对数) 计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的自然对数 (以 *e* 为底的对数)。结果存储在累加器 1 中。输入值必须大于或等于零。此指令影响 CC 1、CC 0、UO 和 OV 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-无穷大	0	1	1	1	溢出
-qNaN	1	1	1	1	

实例

```
STL                     解释
      L     MD10        //将存储器双字 MD10 中的值载入到 ACCU 1 中。
                        //(该值必须为浮点数格式。)
      LN                //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的自然对数。将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是“0”。
      JC    OK          //如果在执行此指令期间未出错，则跳转到 OK 跳转标签。
BEU                     //如果执行此指令时出错，则块无条件结束。
OK:   T     MD20        //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.5 SIN 计算浮点数 (32 位) 角度的正弦值

格式

SIN

指令描述

SIN (计算 32 位 IEEE 754 浮点数角度的正弦) 计算用弧度表示的角度的正弦。角度必须表示为 ACCU 1 中的浮点数。结果存储在累加器 1 中。此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+规格化	1	0	0	-	
+非规格化	0	0	1	1	溢出
+零	0	0	0	-	
+无穷大	1	0	1	1	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-qNaN	1	1	1	1	

参见评估浮点数功能状态字的位

实例

```
STL                 解释
L     MD10          //将存储器双字 MD10 中的值载入到 ACCU 1 中。
                    //(该值必须为浮点数格式。)
SIN                 //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的正弦值。
                    //将结果存储在 ACCU 1 中。
T     MD20          //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.6 COS 计算浮点数 (32 位) 角度的余弦值

格式

COS

指令描述

COS (计算 32 位 IEEE 754 浮点数角度的余弦) 计算用弧度表示的角度的余弦值。角度必须表示为 ACCU 1 中的浮点数。结果存储在累加器 1 中。此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+规格化	1	0	0	-	
+非规格化	0	0	1	1	溢出
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-qNaN	1	1	1	1	

实例

```
STL             解释
L     MD10      //将存储器双字 MD10 中的值载入到 ACCU 1 中。
                //(该值必须为浮点数格式。)
COS             //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的余弦值。
                //将结果存储在 ACCU 1 中。
T     MD20      //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.7 TAN 计算浮点数 (32 位) 角度的正切值

格式

TAN

指令描述

TAN (计算 32 位 IEEE 754 浮点数角度的正切) 计算用弧度表示的角度的正切值。角度必须表示为 ACCU 1 中的浮点数。结果存储在累加器 1 中。此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+无穷大	1	0	1	1	溢出
+规格化	1	0	0	-	
+非规格化	0	0	1	1	下溢
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-无穷大	0	1	1	1	溢出
-qNaN	1	1	1	1	

实例

```
STL                     解释
      L     MD10        //将存储器双字 MD10 中的值载入到 ACCU 1 中。
                        //(该值必须为浮点数格式。)
      TAN               //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的正切值。将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是“0”。
      JC    OK          //如果在执行 TAN 指令期间未出错，跳转到 OK 跳转标签。
BEU                     //如果执行 TAN 指令时出错，块无条件结束。
OK:   T     MD20        //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.8 ASIN 计算浮点数 (32 位) 的反正弦值

格式

ASIN

指令描述

ASIN (生成 32 位，IEEE 754 浮点数的反正弦值) 计算 ACCU 1 中浮点数的反正弦值。输入值的允许值范围

-1 <= 输入值 <= +1

结果是以弧度表示的角度。值位于下列范围

$-\pi / 2 <= \text{arc sine (ACCU1)} <= +\pi / 2$, with $\pi = 3.14159...$

此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+规格化	1	0	0	-	
+非规格化	0	0	1	1	溢出
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-qNaN	1	1	1	1	

实例

```
STL                     解释
      L     MD10        //将存储器双字 MD10 中的值载入到 ACCU 1 中。
                        //(该值必须为浮点数格式。)
      ASIN              //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的反正弦值。将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是“0”。
      JC    OK          //如果在执行 ASIN 指令期间未出错，则跳转到 OK 跳转标签。
BEU                     //如果执行 ASIN 指令时出错，则块无条件结束。
OK:   T     MD20        //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.9 ACOS 计算浮点数 (32 位) 的反余弦值

格式

ACOS

指令描述

ACOS (生成 32 位，IEEE 754 浮点数的反余弦值) 计算 ACCU 1 中浮点数的反余弦值。输入值的允许值范围

-1 <= 输入值 <= +1

结果是用弧度表示的角度。值位于下列范围

0 <= arc cosine (ACCU1) <= π, with π = 3.14159...

此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+规格化	1	0	0	-	
+非规格化	0	0	1	1	溢出
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-qNaN	1	1	1	1	

实例

```
STL                     解释
      L     MD10        //将存储器双字 MD10 中的值载入到 ACCU 1 中。
                        //(该值必须为浮点数格式。)
      ACOS              //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的反余弦值。将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是“0”。
      JC    OK          //如果在执行 ACOS 指令期间未出错，则跳转到 OK 跳转标签。
BEU                     //如果执行 ACOS 指令时出错，则块无条件结束。
OK:   T     MD20        //将结果从 ACCU 1 传送到存储器双字 MD20。
```

8.4.10 ATAN 计算浮点数 (32 位) 的反正切值

格式

ATAN

指令描述

ATAN (计算 32 位 IEEE 754 浮点数的反正切值) 在 ACCU 1 中计算浮点数的反正切值。结果是以弧度表示的角度。值位于下列范围内

$-\pi / 2 <= \text{arc tangent (ACCU1)} <= +\pi / 2$, with $\pi = 3.14159...$

此指令影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及具有四个累加器的 CPU 的累加器 3 和累加器 4 的内容) 保持不变。

结果

ACCU 1 中的结果为	CC 1	CC 0	OV	OS	注意
+qNaN	1	1	1	1	
+规格化	1	0	0	-	
+非规格化	0	0	1	1	溢出
+零	0	0	0	-	
-零	0	0	0	-	
-非规格化	0	0	1	1	下溢
-规格化	0	1	0	-	
-qNaN	1	1	1	1	

实例

```
STL                     解释
      L     MD10        //将存储器双字 MD10 中的值载入到 ACCU 1 中。
                        //(该值必须为浮点数格式。)
      ATAN              //计算 ACCU 1 中浮点数 (32 位，IEEE 754) 的反正切值将结果存储在 ACCU 1 中。
      AN    OV          //扫描状态字中的 OV 位，确定是否是“0”，
      JC    OK          //如果在执行 ATAN 指令期间未出错，
                        //则跳转到 OK 跳转标签。
BEU                     //如果执行 ATAN 指令时出错，则块无条件结束
OK:   T     MD20        //将结果从 ACCU 1 传送到存储器双字 MD20。
```

9 装载和传送指令

9.1 装载和传送指令概述

描述

可使用装载 (L) 和传送 (T) 指令进行编程，以在输入或输出模块与存储区之间、或在各存储区之间进行信息交换。CPU 在每个扫描周期中将这些指令作为无条件指令执行，也就是说，它们不受语句逻辑运算结果的影响。

可用下列装载和传送指令：

- L 装载
- L STW 将状态字加载到 ACCU 1 中
- LAR1 AR2 从地址寄存器 2 装载地址寄存器 1
- LAR1 <D> 用长整型 (32 位指针) 装载地址寄存器 1
- LAR1 从 ACCU 1 装载地址寄存器 1
- LAR2 <D> 用长整型 (32 位指针) 装载地址寄存器 2
- LAR2 从 ACCU 1 装载地址寄存器 2

- T 传送
- T STW 将 ACCU 1 传送至状态字
- TAR1 AR2 将地址寄存器 1 传送至地址寄存器 2
- TAR1 <D> 将地址寄存器 1 传送至目标地址 (32 位指针)
- TAR2 <D> 将地址寄存器 2 传送至目标地址 (32 位指针)
- TAR1 将地址寄存器 1 传送至 ACCU 1
- TAR2 将地址寄存器 2 传送至 ACCU 1
- CAR 交换地址寄存器 1 和地址寄存器 2

9.2 L 装载

格式

L <地址>

地址	数据类型	存储区域	源地址
<地址>	BYTE WORD DWORD	E、A、PE、M、L、D、指针、参数	0...65535 0...65534 0...65532

描述

在将 ACCU 1 的原有内容保存到 ACCU 2 中，并将 ACCU 1 复位到“0”后，L **<地址>** 会将被寻址的字节、字或双字装载到 ACCU 1 中。

状态字

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

实例

```
STL             解释
L     IB10      //将输入字节 IB10 装载到 ACCU 1-L-L 中。
L     MB120     //将存储器字节 MB120 装载到 ACCU 1-L-L 中。
L     DBB12     //将数据字节 DBB12 装载到 ACCU 1-L-L 中。
L     DIW15     //将实例数据字 DIW15 装载到 ACCU 1-L 中。
L     LD252     //将本地数据双字 LD252 装载到 ACCU 1 中。
L     P# I 8.7  //将指针装载到 ACCU 1 中。
L     OTTO      //将参数“OTTO”装载到 ACCU 1 中。
L     P# ANNA   //将指针装载到 ACCU 1 的指定参数中。
                //(此指令装载指定参数的相对地址偏移量。要在多重实例 FB 中计算实例数据块中的绝对偏移量，
                //必须将 AR2 寄存器的内容添加到此值。
```

ACCU 1 的内容

ACCU 1 的内容	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
执行装载指令前	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
执行 **L MB10** (L <字节>) 后	00000000	00000000	00000000	<MB10>
执行 **L MW10** (L <字>) 后	00000000	00000000	<MB10>	<MB11>
执行 **L MD10** (L <双字>) 后	<MB10>	<MB11>	<MB12>	<MB13>
执行 **L P# ANNA** (在 FB 中) 后	<86>	<ANNA 相对于 FB 开始的位偏移量>。 要在多重实例 FB 中计算实例数据块中的绝对偏移量，必须将 AR2 寄存器的内容添加到此值。		
执行 **L P# ANNA** (在 FC 中) 后	<传送至 ANNA 的数据的跨区域地址>			
	X = “1” 或 “0”			

9.3 L STW 将状态字加载到 ACCU 1 中

格式

L STW

描述

L STW (带有地址 STW 的指令 L) 用状态字内容装载 ACCU 1。该指令的执行与状态位无关，对状态位也没有影响。

注意

对于 S7-300 系列 CPU，**L STW** 语句不装载状态字的 FC、STA 和 OR 位。只有第 1、4、5、6、7 和第 8 位装载到累加器 1 低字的相应位中。

状态字

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

实例

```
STL          解释
L    STW     //将状态字内容装载到 ACCU 1 中。
```

执行 **L STW** 后，ACCU 1 的内容为：

位	31-9	8	7	6	5	4	3	2	1	0
内容:	0	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC

9.4 LAR1 从 ACCU 1 装载地址寄存器 1

格式

LAR1

描述

LAR1 用 ACCU 1 的内容 (32 位指针) 装载地址寄存器 AR1。ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

9.5 LAR1 <D> 用长整型 (32 位指针) 装载地址寄存器 1

格式

LAR1<D>

地址	数据类型	存储区域	源地址
<D>	DWORD 指针常量	D、M、L	0...65532

描述

LAR1 <D>用所寻址的双字 <D> 的内容或指针常量装载地址寄存器 AR1。ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例：直接地址

```
STL             解释
LAR1   DBD20    //用数据双字 DBD20 中的指针装载 AR1。
LAR1   DID30    //用实例数据双字 DID30 中的指针装载 AR1。
LAR1   LD180    //用本地数据双字 LD180 中的指针装载 AR1。
LAR1   MD24     //用存储器双字 MD24 的内容装载 AR1。
```

实例：指针常量

```
STL             解释
LAR1   P#M100.0 //用 32 位指针常量装载 AR1。
```

9.6 LAR1 AR2 从地址寄存器 2 装载地址寄存器 1

格式

LAR1 AR2

描述

LAR1 AR2 (带有地址 AR2 的指令 LAR1) 用地址寄存器 AR2 的内容装载地址寄存器 1。ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

9.7 LAR2 从 ACCU 1 装载地址寄存器 2

格式

LAR2

描述

LAR2 用 ACCU 1 的内容 (32 位指针) 装载地址寄存器 AR2。

ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

9.8 LAR2 <D> 用长整型 (32 位指针) 装载地址寄存器 2

格式

LAR2 <D>

地址	数据类型	存储区域	源地址
<D>	DWORD 指针常量	D、M、L	0...65532

描述

LAR2 <D> 用所寻址的双字 <D> 的内容或指针常量装载地址寄存器 AR2。ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例：直接地址

```
STL             解释
LAR2   DBD 20   //用数据双字 DBD20 中的指针装载 AR2。
LAR2   DID 30   //用实例数据双字 DID30 中的指针装载 AR2。
LAR2   LD 180   //用本地数据双字 LD180 中的指针装载 AR2。
LAR2   MD 24    //用存储器双字 MD24 中的指针装载 AR2。
```

实例：指针常量

```
STL             解释
LAR2   P#M100.0 //用 32 位指针常量装载 AR2。
```

9.9 T 传送

格式

T <地址>

地址	数据类型	存储区域	源地址
<地址>	BYTE	I、Q、PQ、M、L、D	0...65535
	WORD		0...65534
	DWORD		0...65532

描述

如果主控继电器打开 (MCR = 1)，**T <地址>** 会将 ACCU 1 的内容传送 (复制) 到目标地址。如果 MCR = 0，则用 0 写目标地址。从 ACCU 1 复制的字节数取决于目标地址中表明的长度。传送后，ACCU 1 还保存此数据。当传送至直接 I/O 区域 (存储器类型 PQ) 时，还将 ACCU 1 的内容或“0” (如果 MCR=0) 传送至过程映像输出表 (存储器类型 Q) 中的相应地址。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL             解释
T     QB10      //将 ACCU 1-L-L 的内容传送给输出字节 QB10。
T     MW14      //将 ACCU 1-L 的内容传送给存储器字 MW14。
T     DBD2      //将 ACCU 1 的内容传送给数据双字 DBD2。
```

9.10 T STW 将 ACCU 1 传送至状态字

格式

T STW

描述

T STW (带有地址 STW 的指令 T) 将 ACCU 1 的 0 至 8 位传送给状态字。

该指令的执行与状态位无关。

注意：对于 S7-300 系列的 CPU，不由 T STW 指令写入状态字/ER、STA 和 OR 的位。根据 ACCU1 的位设置，仅写入位 1、4、5、6、7 和 8。

状态字

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

实例

```
STL                 解释
T     STW           //将 ACCU 1 的 0 至 8 位传送给状态字。
```

ACCU 1 的各个位包含以下状态位：

位	31-9	8	7	6	5	4	3	2	1	0
内容：	*)	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC

*) 未传送位。

9.11 CAR 交换地址寄存器 1 和地址寄存器 2

格式

CAR

描述

CAR (交换地址寄存器) 交换地址寄存器 AR1 和 AR2 的内容。该指令的执行与状态位无关，对状态位也没有影响。

地址寄存器 AR1 的内容移到地址寄存器 AR2 中，
地址寄存器 AR2 的内容移到地址寄存器 AR1 中。

状态字

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

9.12 TAR1 将地址寄存器 1 传送至 ACCU 1

格式

TAR1

描述

TAR1 将地址寄存器 AR1 的内容传送给 ACCU 1 (32 位指针)。ACCU 1 中的原有内容保存在 ACCU 2 中。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

9.13 TAR1 <D> 将地址寄存器 1 传送至目标地址 (32 位指针)

格式

TAR1 <D>

地址	数据类型	存储区域	源地址
<D>	DWORD	D、M、L	0...65532

描述

TAR1 <D> 将地址寄存器 AR1 的内容传送给寻址的双字 <D>。目标区域可以为存储器双字 (MD) 、本地数据双字 (LD) 、数据双字 (DBD) 和实例数据字 (DID)。

ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL                 解释
TAR1   DBD20        //将 AR1 的内容传送给数据双字 DBD20。
TAR1   DID30        //将 AR1 的内容传送给实例数据双字 DID30。
TAR1   LD18         //将 AR1 的内容传送给本地数据双字 LD18。
TAR1   MD24         //将 AR1 的内容传送给存储器双字 MD24。
```

9.14 TAR1 AR2 将地址寄存器 1 传送至地址寄存器 2

格式

TAR1 AR2

描述

TAR1 AR2 (带有地址 AR2 的指令 TAR1) 将地址寄存器 AR1 的内容传送给地址寄存器 AR2。

ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

9.15 TAR2 将地址寄存器 2 传送至 ACCU 1

格式

TAR2

描述

TAR2 将地址寄存器 AR2 的内容传送给 ACCU 1 (32 位指针)。ACCU 1 的内容提前保存到 ACCU 2 中。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

9.16 TAR2 <D> 将地址寄存器 2 传送至目标地址 (32 位指针)

格式

TAR2 <D>

地址	数据类型	存储区域	源地址
<D>	DWORD	D、M、L	0...65532

描述

TAR2 <D> 将地址寄存器 AR2 的内容传送给寻址的双字 <D>。
目标区域可以为存储器双字 (MD) 、本地数据双字 (LD) 、数据双字 (DBD) 和实例双字 (DID)。

ACCU 1 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL             解释
TAR2   DBD20    //将 AR2 的内容传送给数据双字 DBD20。
TAR2   DID30    //将 AR2 的内容传送给实例双字 DID30。
TAR2   LD18     //将 AR2 的内容传送给本地数据双字 LD18。
TAR2   MD24     //将 AR2 的内容传送给存储器双字 MD24。
```

10 程序控制指令

10.1 程序控制指令总览

描述

下列指令可用于执行程序控制指令：

- BE 块结束
- BEC 有条件的块结束
- BEU 无条件的块结束
- CALL 块调用
- CC 条件调用
- UC 无条件调用

- 调用 FB
- 调用 FC
- 调用 SFB
- 调用 SFC
- 调用多重情景
- 调用来自库的块

- MCR (主控继电器)
- 关于使用 MCR 功能的重要注意事项
- MCR(将 RLO 保存到 MCR 堆栈中，开始 MCR
-)MCR 结束 MCR
- MCRA 激活 MCR 区域
- MCRD 取消激活 MCR 区域

10.2 BE 块结束

格式

BE

描述

BE (块结束) 终止当前块中的程序扫描，并导致跳转到调用当前块的那个块中。程序扫描继续执行调用程序中紧跟块调用声明之后的第一条指令。当前本地数据区将被释放，前一个本地数据区将变为当前本地数据区。当块被调用时所打开的数据块被重新打开。此外，还将恢复调用块的 MCR 依存关系，而 RLO 被从当前块移送到调用当前块的那个块中。BE 不依赖于任何条件。然而，如果 BE 指令被跳过，则当前程序扫描将不会结束，将从块中跳转目标处开始继续执行。

BE 指令对于 S5 软件有所不同。在 S7 硬件中，该指令使用时与 BEU 的功能相同。

状态字

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

实例

```
STL                         解释
        A    I 1.0
        JC   NEXT       //如果 RLO = 1 (I 1.0 = 1)，则跳转到 NEXT 跳转标签。
        L    IW4        //若未执行跳转，则从此处继续。
        T    IW10
        A    I 6.0
        A    I 6.1
        S    M 12.0
        BE              //块结束
NEXT:   NOP             //若执行跳转，则从此处继续。
        0
```

10.3 BEC 有条件的块结束

格式

BEC

描述

如果 RLO = 1，则 **BEC (**有条件的块结束) 中断当前块中的程序扫描，导致跳转至调用当前的那个块。程序扫描继续执行块调用之后的第一条指令。当前本地数据区将被释放，前一个本地数据区将变为当前本地数据区。在块被调用时为当前数据块的数据块被重新打开。恢复调用块的 MCR 依存关系。

RLO (= 1) 被从已终止的块传送到被调用的块。如果 RLO = 0，则不执行 BEC。RLO 被设为 1，程序扫描继续执行 BEC 之后的指令。

状态字

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

实例

```
STL                 解释
A     I 1.0         //更新 RLO。
BEC                 //如果 RLO = 1，则结束块。
L     IW4           //如果未执行 BEC，则从此处继续，RLO = 0。
T     MW10
```

10.4 BEU 无条件的块结束

格式

BEU

描述

BEU (无条件的块结束) 终止当前块中的程序扫描，并导致跳转到调用当前块的那个块中。程序扫描继续执行块调用之后的第一条指令。当前本地数据区将被释放，前一个本地数据区将变为当前本地数据区。当块被调用时所打开的数据块被重新打开。此外，还将恢复调用块的 MCR 依存关系，而 RLO 被从当前块移送到调用当前块的那个块中。BEU 不依赖于任何条件。然而，如果 BEU 指令被跳过，当前程序扫描将不会结束，将从块中跳转目标处开始继续执行。

状态字

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

实例

```
STL                          解释
        A       I 1.0
        JC      NEXT         //如果 RLO = 1 (I 1.0 = 1)，则跳转到 NEXT 跳转标签。
        L       IW4          //若未执行跳转，则从此处继续。
        T       IW10
        A       I 6.0
        A       I 6.1
        S       M 12.0
        BEU                  //无条件的块结束。
NEXT:   NOP 0                //若执行跳转，则从此处继续。
```

10.5 CALL 块调用

格式

CALL <逻辑块标识符>

描述

CALL <逻辑块标识符>用于调用功能 (FC) 或功能块 (FB) 、系统功能 (SFC) 或系统功能块 (SFB)，或调用由 Siemens 提供的标准预编程的程序块。CALL 指令调用作为地址输入的 FC 和 SFC，或 FB 和 SFB，它与 RLO 或任何其它条件无关。如果使用 CALL 调用 FB 或 SFB，必须为块提供相关的背景数据块。在被调用块处理完成之后，调用块程序将继续逻辑处理。可以绝对方式或符号方式指定逻辑块的地址。在 SFB/SFC 调用结束之后，将恢复寄存器内容。

实例：CALL FB1, DB1 或 CALL FILLVAT1, RECIPE1

逻辑块	块类型	绝对地址调用语法
FC	功能	CALL FCn
SFC	系统功能	CALL SFCn
FB	功能块	CALL FBn1,DBn2
SFB	系统功能块	CALL SFBn1,DBn2

注意

在使用 STL 编辑器时，上表中的引用 (n、n1 和 n2) 必须指向有效存在的块。否则，必须在使用前定义符号名。

传递参数 (增量编辑模式)

调用块可通过变量表与被调用块交换参数。在输入有效的 CALL 声明后，变量表将自动在 STL 程序中扩展。

如果调用 FB、SFB、FC 或 SFC，且被调用块的变量声明表具有 IN、OUT 和 IN_OUT 声明，这些变量将作为形式参数表被添加到调用块中。

当调用 FC 和 SFC 时，必须从调用逻辑块将实际参数分配给形式参数。

当调用 FB 和 SFB 时，必须仅指定那些必须由前一个调用更改的实际参数。在处理完 FB 之后，实际参数将被保存在背景数据块中。如果实际参数是数据块，必须指定完整的绝对地址，例如：DB1、DBW2。

IN 参数可作为常数、绝对或符号地址来指定。而 OUT 和 IN_OUT 参数必须作为绝对或符号地址来指定。必须确保所有地址和常数符合要传送的数据类型。

CALL 保存返回地址 (选择器和相对地址) 、两个当前数据块的选择器以及 B (块) 堆栈中的 MA 位。此外，CALL 将取消激活 MCR 依存关系，然后创建要传送的块的本地数据区。

状态字

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

实例 1：为 FC6 调用分配参数

```
CALL    FC6
        形式参数              实际参数
        NO OF TOOL            := MW100
        TIME OUT              := MW110
        FOUND                 := Q 0.1
        ERROR                 := Q 100.0
```

实例 2：不带参数调用 SFC

STL		解释
CALL	SFC43	//调用 SFC43，重新触发监视狗定时器 (无参数)。

实例 3：使用背景数据块 DB1 调用 FB99

```
CALL    FB99,DB1
        形式参数            实际参数
        MAX_RPM             := #RPM1_MAX
        MIN_RPM             := #RPM1
        MAX_POWER           := #POWER1
        MAX_TEMP            := #TEMP1
```

实例 4：使用背景数据块 DB2 调用 FB99

```
CALL    FB99,DB2
        形式参数            实际参数
        MAX_RPM             := #RPM2_MAX
        MIN_RPM             := #RPM2
        MAX_POWER           := #POWER2
        MAX_TEMP            := #TEMP2
```

注意

每个 FB 或 SFB 调用都必须有一个背景数据块。在上面的实例中，块 DB1 和 DB2 在调用前必须始终存在。

10.6 调用 FB

格式

CALL FB n1, DB n1

描述

该指令用于调用用户自定义的功能块 (FB)。CALL 指令调用作为地址输入的功能块，与 RLO 或其它条件无关。如果使用 CALL 调用功能块，必须为其提供一个背景数据块。在处理完所调用的块之后，处理过程将继续处理调用该块的程序。可以绝对方式或符号方式指定逻辑块的地址。

传递参数 (增量编辑模式)

调用块可通过变量表与被调用块交换参数。在输入有效的 CALL 指令后，变量表将自动在语句表程序中扩展。

如果调用功能块，且被调用块的变量声明表具有 IN、OUT 和 IN_OUT 声明，这些变量将作为形式参数表被添加到调用块的程序中。

当调用功能块时，由于在处理功能块之后实际参数被保存在背景数据块中，因而只需指定那些必须由前一个调用更改的实际参数。如果实际参数是数据块，必须指定完整的绝对地址，例如：DB1、DBW2。

IN 参数可作为常数、绝对或符号地址来指定。而 OUT 和 IN_OUT 参数必须作为绝对或符号地址来指定。必须确保所有地址和常数符合要传送的数据类型。

CALL 保存返回地址 (选择器和相对地址) 、两个打开的数据块的选择器以及 B (块) 堆栈中的 MA 位。此外，CALL 将取消激活 MCR 依存关系，然后创建要传送的块的本地数据区。

状态字

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

实例 1：使用背景数据块 DB1 调用 FB99

```
CALL    FB99,DB1
        形式参数            实际参数
        MAX_RPM             := #RPM1_MAX
        MIN_RPM             := #RPM1
        MAX_POWER           := #POWER1
        MAX_TEMP            := #TEMP1
```

实例 2：使用背景数据块 DB1 调用 FB99

```
CALL    FB99,DB2
        形式参数            实际参数
        MAX_RPM             := #RPM2_MAX
        MIN_RPM             := #RPM2
        MAX_POWER           := #POWER2
        MAX_TEMP            := #TEMP2
```

注意

每个功能块调用都必须有一个背景数据块。在上面的实例中，块 DB1 和 DB2 在调用前必须始终存在。

10.7 调用 FC

格式

CALL FC n

注意

如果使用的是 STL 编辑器，则引用 (n) 必须相对于现有的有效块。还必须在使用前定义符号名。

描述

该指令用于调用功能 (FC)。CALL 指令调用作为地址输入的 FC，与 RLO 或其它条件无关。在处理完所调用的块之后，处理过程将继续处理调用该块的程序。可以绝对方式或符号方式指定逻辑块的地址。

传递参数 (增量编辑模式)

调用块可通过变量表与被调用块交换参数。在输入有效的 CALL 指令后，变量表将自动在语句表程序中扩展。

如果调用功能，且被调用块的变量声明表具有 IN、OUT 和 IN_OUT 声明，这些变量将作为形式参数表被添加到调用块的程序中。

当调用功能时，必须从调用逻辑块将实际参数分配给形式参数。

IN 参数可作为常数、绝对或符号地址来指定。而 OUT 和 IN_OUT 参数必须作为绝对或符号地址来指定。必须确保所有地址和常数符合要传送的数据类型。

CALL 保存返回地址 (选择器和相对地址) 、两个打开的数据块的选择器以及 B (块) 堆栈中的 MA 位。此外，CALL 将取消激活 MCR 依存关系，然后创建要传送的块的本地数据区。

状态字

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

实例：为 FC6 调用分配参数

```
CALL    FC6
        形式参数              实际参数
        NO OF TOOL          := MW100
        TIME OUT            := MW110
        FOUND               := Q0.1
        ERROR               := Q100.0
```

10.8 调用 SFB

格式

CALL SFB n1, DB n2

描述

该指令用于调用由 Siemens 提供的标准功能块 (SFB)。CALL 指令调用作为地址输入的 SFB，与 RLO 或其它条件无关。如果使用 CALL 调用系统功能块，必须为其提供一个背景数据块。在处理完所调用的块之后，处理过程将继续处理调用该块的程序。可以绝对方式或符号方式指定逻辑块的地址。

传递参数 (增量编辑模式)

调用块可通过变量表与被调用块交换参数。在输入有效的 CALL 指令后，变量表将自动在语句表程序中扩展。

如果调用系统功能块，且被调用块的变量声明表具有 IN、OUT 和 IN_OUT 声明，这些变量将作为形式参数表被添加到调用块的程序中。

当调用系统功能块时，由于在处理系统功能块之后实际参数被保存在背景数据块中，因而只需指定那些必须由前一个调用更改的实际参数。如果实际参数是数据块，必须指定完整的绝对地址，例如：DB1、DBW2。

IN 参数可作为常数、绝对或符号地址来指定。而 OUT 和 IN_OUT 参数必须作为绝对或符号地址来指定。必须确保所有地址和常数符合要传送的数据类型。

CALL 保存返回地址 (选择器和相对地址) 、两个打开的数据块的选择器以及 B (块) 堆栈中的 MA 位。此外，CALL 将取消激活 MCR 依存关系，然后创建要传送的块的本地数据区。

状态字

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

实例

```
CALL    SFB4,DB4
        形式参数            实际参数
        IN:                 I0.1
        PT:                 T#20s
        Q:                  M0.0
        ET:                 MW10
```

注意

每个系统功能块调用都必须有一个背景数据块。在上面的实例中，块 SFB4 和 DB4 在调用前必须始终存在。

10.9 调用 SFC

格式

CALL SFC n

注意

如果使用的是 STL 编辑器，则引用 (n) 必须相对于现有的有效块。还必须在使用前定义符号名。

描述

该指令用于调用由 Siemens 提供的标准功能 (SFC)。CALL 指令调用作为地址输入的 SFC，与 RLO 或其它条件无关。在处理完所调用的块之后，处理过程将继续处理调用该块的程序。可以绝对方式或符号方式指定逻辑块的地址。

传递参数 (增量编辑模式)

调用块可通过变量表与被调用块交换参数。在输入有效的 CALL 指令后，变量表将自动在语句表程序中扩展。

如果调用系统功能，且被调用块的变量声明表具有 IN、OUT 和 IN_OUT 声明，这些变量将作为形式参数表被添加到调用块的程序中。

当调用系统功能时，必须从调用逻辑块将实际参数分配给形式参数。

IN 参数可作为常数、绝对或符号地址来指定。而 OUT 和 IN_OUT 参数必须作为绝对或符号地址来指定。必须确保所有地址和常数符合要传送的数据类型。

CALL 保存返回地址 (选择器和相对地址) 、两个打开的数据块的选择器以及 B (块) 堆栈中的 MA 位。此外，CALL 将取消激活 MCR 依存关系，然后创建要传送的块的本地数据区。

状态字

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

实例：不带参数调用 SFC

```
STL             解释
CALL    SFC43   //调用 SFC43，重新触发监视狗定时器 (无参数)。
```

10.10 调用多重情景

格式

CALL # 变量名称

描述

使用功能块的数据类型声明静态变量可创建多重背景。只有已声明的多重背景才会被包括在程序元素目录中。

状态字

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

10.11 调用来自库的块

可在此使用 SIMATIC 管理器中提供的库来选择以下块：

- 集成在 CPU 操作系统中的块（“标准库”）
- 保存在库中，以备日后使用。

10.12 CC 条件调用

格式

CC <逻辑块标识符>

描述

CC <逻辑块标识符> (有条件的块调用) 在 RLO=1 时调用逻辑块。CC 用于调用不带参数的 FC 或 FB 类型的逻辑块。CC 的使用方式与 **CALL** 指令相同，除了无法在调用程序时传送参数。指令将返回地址 (选择器和相对地址) 、两个当前数据块的选择器以及 MA 位保存在 B (块) 堆栈中，取消激活 MCR 依存关系，创建被调用块的本地数据区，开始执行被调用的代码。可以绝对方式或符号方式指定逻辑块的地址。

状态字

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

实例

```
STL             解释
A    I 2.0      //检查输入 I 2.0 的信号状态。
CC   FC6        //如果 I 2.0 为“1”，调用功能 FC6。
A    M 3.0      //当从被调用功能返回后执行 (I 2.0 = 1)，或如果 I 2.0 = 0，直接在 A I 2.0 声明之后执行。
```

注意

如果 **CALL** 指令调用功能块 (FB) 或系统功能块 (SFB)，必须在声明中指定背景数据块 (DB 编号)。不得与 **CC** 指令一起使用“BlockFB”或“BlockFC”类型的变量。由于无法使用声明的地址中的 **CC** 指令将数据块分配给调用，只能将该指令用于不带块参数和静态本地数据的块。

在将梯形图逻辑编程语言转换为语句表编程语言期间，取决于所使用的程序段，程序编辑器将生成 **UC** 指令或 **CC** 指令。您应当尝试使用 **CALL** 指令来代替，以避免程序发生错误。

10.13 UC 无条件调用

格式

UC <逻辑块标识符>

描述

UC <逻辑块标识符> (无条件调用块) 调用 FC 或 SFC 类型的逻辑块。UC 与 CALL 指令类似，除了无法使用被调用块传送参数。指令将返回地址 (选择器和相对地址) 、两个当前数据块的选择器以及 MA 位保存在 B (块) 堆栈中，取消激活 MCR 依存关系，创建被调用块的本地数据区，开始执行被调用的代码。

状态字

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

实例 1

```
STL                 解释
UC    FC6           //调用功能 FC6 (不带参数)。
```

实例 2

```
STL                 解释
UC    SFC43         //调用系统功能 SFC43 (不带参数)。
```

注意

当使用 **CALL** 指令调用 FB 或 SFB，必须在指令中指定背景数据块 (DB 编号)。不得与 **UC** 指令一起使用“BlockFB”或“BlockFC”类型的变量。由于无法使用声明的地址中的 **UC** 指令将数据块分配给调用，只能将该指令用于不带块参数和静态本地数据的块。

在将梯形图逻辑编程语言转换为语句表编程语言期间，取决于所使用的程序段，程序编辑器将生成 **UC** 指令或 **CC** 指令。您应当尝试使用 **CALL** 指令来代替，以避免程序发生错误。

10.14 MCR (主控继电器)

关于使用 MCR 功能的重要注意事项

警告

为了防止人员受伤或财产损失，切勿使用 MCR 来代替紧急停止功能硬接线的机械主控继电器。

描述

主控继电器 (MCR) 是一个用于通电和断电的继电器梯形图逻辑主站开关。由下列位逻辑触发的指令和传送指令取决于 MCR：

- **=** <位>
- **S** <位>
- **R** <位>
- **T** <字节>、**T** <字>、**T** <双字>

如果 MCR 为 0，**T** 指令 (用于字节、字和双字) 将 0 写入到存储器。**S** 和 **R** 指令则不改变现有值。指令**=**将“0”写入所给定地址的位中。

取决于 MCR 的指令和它们对 MCR 信号状态的响应

MCR 的信号状态	= <位>	S <位>, R <位>	T <字节>, T <字> T <双字>
0 (“OFF”)	写入 0。 (模拟继电器在移除电压后进入其静止状态。)	不写入。 (模拟继电器在移除电压后保持其当前状态。)	写入 0。 (模拟组件在移除电压后生成一个 0 值。)
1 (“ON”)	正常处理	正常处理	正常处理

MCR(- 开始 MCR 区域，)MCR - 结束 MCR 区域

MCR 由一个一位宽八位深的堆栈控制。当所有八个条目都为 1 时，MCR 激活。MCR(指令将 RLO 位复制到 MCR 堆栈中。)MCR 指令从堆栈中移除最后一个条目，并将空出来的位置设为 1。MCR(和)MCR 指令必须始终成对使用。当发生故障时，即当连续使用超过八个 MCR(指令，或者试图在 MCR 堆栈为空时执行 MCR)指令时，将触发 MCRF 出错消息。

MCRA - 激活 MCR 区域，MCRD - 取消激活 MCR 区域

MCRA 和 MCRD 必须始终成对使用。在 MCRA 和 MCRD 之间所编的指令将取决于 MCR 位的状态。在 MCRA-MCRD 序列之外所编的指令与 MCR 位状态无关。

必须在被调用块中使用 MCRA 指令编定块中功能 (FC) 和功能块 (FB) 的 MCR 依存关系。

10.15 关于使用 MCR 功能的重要注意事项

请小心使用那些已用 MCRA 激活主控继电器的块

- 如果 MCR 已取消激活，将由 **MCR(**和**)MCR** 之间的程序段中的所有赋值指令 (T、=) 写入 0 值。
- 如果在 **MCR(**指令之前，RLO = 0，则 MCR 将取消激活。

危险：PLC 处于 STOP 模式或未定义的运行特性！

编译器也使用在 VAR_TEMP 中为计算地址而定义的临时变量，对本地数据进行写访问。这意味着，下列命令序列将把 PLC 设为 STOP 模式，或导致未定义的运行特性：

形式参数访问

- 访问 STRUCT、UDT、ARRAY、STRING 类型的复杂 FC 参数的构成成分。
- 访问具有多重背景能力的块 (版本 2 型块) 中 IN_OUT 区域的 STRUCT、UDT、ARRAY、STRING 类型的复杂 FC 参数的构成成分。
- 如果其地址大于 8180.0，访问具有多重背景功能的功能块 (版本 2 型块) 的参数。
- 访问具有多重背景功能的功能块 (版本 2 型块) 的 BLOCK_DB 类型的参数，打开 DB0。其后任何数据访问都将会把CPU 设为STOP 模式。T 0、C 0、FC0 或FB0 也将始终用于TIMER、COUNTER、BLOCK_FC 和BLOCK_FB。

参数传递

- 进行参数传送的调用。

LAD/FBD

- 梯形图中的 T 分支和中线输出或 FBD 以 RLO = 0 开始。

纠正方法

解除上述命令的 MCR 依存关系：

1. 在有关的声明或程序段之前使用 MCRD 指令取消激活主控继电器。
2. 在有关的声明或程序段之后使用 MCRA 指令重新激活主控继电器。

10.16 MCR(　将 RLO 保存到 MCR 堆栈中，开始 MCR

关于使用 MCR 功能的重要注意事项

格式

MCR(

描述

MCR((打开 MCR 区域) 将 RLO 保存到 MCR 堆栈中并打开一个 MCR 区域。MCR 区域是介于指令 **MCR(**和相应的指令**)MCR** 之间的指令。指令 **MCR(**必须始终与指令**)MCR** 成对使用。

如果 RLO=1，则 MCR 为“on”。该 MCR 区域中取决于 MCR 的指令正常执行。

如果 RLO=0，则 MCR 为“off”。

该 MCR 区域中取决于 MCR 的指令将按照下表所述执行。

取决于 MCR 位状态的指令

MCR 的信号状态	= <位>	S <位>, R <位>	T <字节>, T <字> T <双字>
0 （“OFF”）	写入 0。 (模拟继电器在移除电压后进入其静止状态。)	不写入。 (模拟继电器在移除电压后保持其当前状态。)	写入 0。 (模拟组件在移除电压后生成一个 0 值。)
1 （“ON”）	正常处理	正常处理	正常处理

MCR(和)MCR 指令可嵌套使用。最大嵌套深度为八个指令。最多允许的堆栈条目数为八。在堆栈已满的情况下执行 MCR(将产生 MCR 堆栈故障 (MCRF)。

状态字

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

实例

```
STL                 解释
MCRA                //激活 MCR 区域。
A       I 1.0
MCR(                //将 RLO 保存到 MCR 堆栈中，打开 MCR 区域。当 RLO=1 (I 1.0 =“1”) 时，MCR = “on”；
                    //当 RLO=0 (I 1.0 =“0”) 时，MCR = “off”。
A       I 4.0
=       Q 8.0       //如果 MCR = “off”，则无论 I 4.0 为何值，Q 8.0 都将被设为“0”。
L       MW20
T       QW10        //如果 MCR = “off”，则将“0”传送到 QW10。
)MCR                //MCR 区域结束。
MCRD                //取消激活 MCR 区域。
A       I 1.1
=       Q 8.1       //这些指令位于 MCR 区域之外，不依赖于 MCR 位。
```

10.17)MCR 结束 MCR

关于使用 MCR 功能的重要注意事项

格式

)MCR

描述

)MCR (结束 MCR 区域) 从 MCR 堆栈中移除条目，结束 MCR 区域。上一个 MCR 堆栈位置被释放出来并被设为 1。指令 MCR(必须始终与指令)MCR 成对使用。在堆栈为空的情况下执行)MCR 指令将产生 MCR 堆栈故障 (MCRF)。

状态字

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

实例

```
STL             解释
MCRA            //激活 MCR 区域。
A       I 1.0
MCR(            //将 RLO 保存到 MCR 堆栈中；打开 MCR 区域。当 RLO=1 (I 1.0 =“1”) 时，MCR = “on”；
                //当 RLO=0 (I 1.0 =“0”) 时，MCR = “off”。
A       I 4.0
=       Q 8.0   //如果 MCR = “off”，则无论 I 4.0 为何值，Q 8.0 都将被设为“0”。
L       MW20
T       QW10    //如果 MCR = “off”，则将“0”传送到 QW10。
)MCR            //MCR 区域结束。
MCRD            //取消激活 MCR 区域。
A       I 1.1
=       Q 8.1   //这些指令位于 MCR 区域之外，不依赖于 MCR 位。
```

10.18 MCRA 激活 MCR 区域

关于使用 MCR 功能的重要注意事项

格式

MCRA

描述

MCRA (主控继电器激活) 激活跟在其后的指令的 MCR 依存关系。指令 MCRA 必须始终与指令 MCRD (主控继电器取消激活) 成对使用。在 MCRA 和 MCRD 之间所编的指令将取决于 MCR 位的信号状态。

执行该指令时不涉及，也不会影响状态字的位。

状态字

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

实例

```
STL             解释
MCRA            //激活 MCR 区域。
A       I 1.0
MCR(            //将 RLO 保存到 MCR 堆栈中，打开 MCR 区域。当 RLO=1 (I 1.0 =“1”) 时，MCR = “on”；
                //当 RLO=0 (I 1.0 =“0”) 时，MCR = “off”。
A       I 4.0
=       Q 8.0   //如果 MCR = “off”，则无论 I 4.0 为何值，Q 8.0 都将被设为“0”。
L       MW20
T       QW10    //如果 MCR = “off”，则将“0”传送到 QW10。
)MCR            //MCR 区域结束。
MCRD            //取消激活 MCR 区域。
A       I 1.1
=       Q 8.1   //这些指令位于 MCR 区域之外，不依赖于 MCR 位。
```

10.19 MCRD 取消激活 MCR 区域

关于使用 MCR 功能的重要注意事项

格式

MCRD

描述

MCRD (主控继电器取消激活) 取消激活跟在其后的指令的 MCR 依存关系。指令 MCRA (主控继电器激活) 必须始终与指令 MCRD (主控继电器取消激活) 成对使用。在 MCRA 和 MCRD 之间所编的指令将取决于 MCR 位的信号状态。

执行该指令时不涉及，也不会影响状态字的位。

状态字

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

实例

```
STL             解释
MCRA            //激活 MCR 区域。
A       I 1.0
MCR(            //将 RLO 保存到 MCR 堆栈中，打开 MCR 区域。当 RLO=1 (I 1.0 =“1”) 时，MCR = “on”；
                //当 RLO=0 (I 1.0 =“0”) 时，MCR = “off”。
A       I 4.0
=       Q 8.0   //如果 MCR = “off”，则无论 I 4.0 为何值，Q 8.0 都将被设为“0”。
L       MW20
T       QW10    //如果 MCR = “off”，则将“0”传送到 QW10。
)MCR            //MCR 区域结束。
MCRD            //取消激活 MCR 区域。
A       I 1.1
=       Q 8.1   //这些指令位于 MCR 区域之外，不依赖于 MCR 位。
```

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

可使用移位指令逐位左移或右移累加器 1 中低字的内容或整个累加器的内容 (参见 CPU 寄存器)。左移 n 位相当于将累加器的内容乘以“2 n”；右移 n 位相当于将累加器的内容除以“2 n”。例如，将以二进制格式表示的十进制数 3 左移 3 位时，在累加器中出现相当于十进制数 24 的二进制编码。将以二进制格式表示的十进制数 16 右移 2 位时，在累加器中出现相当于十进制数 4 的二进制编码。

移位指令后的数字或在累加器 2 的低字低字节中的数值表示要移位的数目。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。将最后一个移出的位装载到状态字的 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。移位运算是无条件的，即，它们的执行不需要任何特殊的条件，且不影响逻辑运算的结果。

下列移位指令可用：

- SSI 带符号整型移位 (16 位)
- SSD 带符号长整型移位 (32 位)
- SLW 左移字 (16 位)
- SRW 右移字 (16 位)
- SLD 左移双字 (32 位)
- SRD 右移双字 (32 位)

11.1.2 SSI 带符号整型移位 (16 位)

格式

SSI
SSI <数目>

地址	数据类型	描述
<数字>	整型、无符号	要移位的位数目，范围为 0 - 15

描述

SSI (右移带符号整型) 只逐位向右移动 ACCU 1-L 的内容。由符号位 (位 15) 的信号状态填充移位指令空出的位。将最后一个移出的位装载到状态字的 CC 1 位中。地址<数目>或 ACCU 2-L-L 中的数值指定要移位的位数目。

SSI <数目>：地址<数目>指定移位数目。允许的数值范围为 0 - 15。当<数目>大于 0 时，复位状态字的位 CC 0 和 OV 为 0。当<数目>等于 0 时，则将此移位指令视为 **NOP** 操作。

SSI：移位数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 -255。移位数目大于 16 时，始终产生相同的结果 (ACCU 1 = 16#0000、CC 1 = 0 或 ACCU 1 = 16#FFFF、CC 1 = 1)。当移位数目大于 0 时，复位状态字的位 CC 0 和 OV 为 0。当移位数目为 0 时，则将移位指令视为 **NOP** 操作。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **SSI6** 前	0101	1111	0110	0100	1001	1101	0011	1011
执行 **SSI6** 后	0101	1111	0110	0100	1111	1110	0111	0100

实例 1

STL		解释
L	MW4	//将值载入 ACCU 1 中。
SRW	6	//将 ACCU 1 中的带符号的位向右移动 6 位。
T	MW8	//将结果传送到 MW8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MW20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MW20 的数值装载到 ACCU 1 中。
SRW		//移位数目为 ACCU 2-L-L 的数值 => 将 ACCU 1-L 中的带符号的位右移 3 位；
		//用符号位的状态填充空闲位。
JP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.1.3 SSD 带符号长整型移位 (32 位)

格式

SSD
SSD <数目>

地址	数据类型	描述
<数字>	整型、无符号	要移位的位数目，范围为 0 - 32

描述

SSD (右移带符号的长整型) 逐位向右移动 ACCU 1 的整个内容。由符号位的信号状态填充移位指令空出的位。将最后一个移出的位装载到状态字的 CC 1 位中。地址<数目>或 ACCU 2-L-L 中的数值指定要移位的位数目。

SSD <数目>：地址<数目>指定移位数目。允许的数值范围为 0 - 32。当<数目>大于 0 时，复位状态字的位 CC 0 和 OV 为 0。当<数目>等于 0 时，则将此移位指令视为 **NOP** 操作。

SSD：移位数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 -255。移位数目大于 32 时，始终产生相同的结果 (ACCU 1 = 32#00000000、CC 1 = 0 或 ACCU 1 = 32#FFFFFFFF、CC 1 = 1)。当移位数目大于 0 时，复位状态字的位 CC 0 和 OV 为 0。当移位数目为 0 时，则将移位指令视为 **NOP** 操作。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **SSD 7** 前	1000	1111	0110	0100	0101	1101	0011	1011
执行 **SSD 7** 后	1111	1111	0001	1110	1100	1000	1011	1010

实例 1

STL		解释
L	MD4	//将值载入 ACCU 1 中。
SSD	7	//根据符号，将 ACCU 1 中的位右移 7 位。
T	MD8	//将结果传送到 MD8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MD20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MD20 的数值装载到 ACCU 1 中。
SSD		//移位数目为 ACCU 2-L-L 的数值 => 将 ACCU 1 中的带符号的位右移 3 位， //用符号位的状态填充空闲位。
JP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.1.4 SLW 左移字 (16 位)

格式

SLW
SLW <数目>

地址	数据类型	描述
<数字>	整型、无符号	要移位的位数目，范围为 0 - 15

描述

SLW (左移字) 只逐位向左移动 ACCU 1-L 的内容。由零填充移位指令空出的位。将最后一个移出的位装载到状态字的 CC 1 位中。地址<数目>或 ACCU 2-L-L 中的数值指定要移位的位数目。

SLW <数目>：地址<数目>指定移位数目。允许的数值范围为 0 - 15。当<数目>大于 0 时，复位状态字的位 CC 0 和 OV 为 0。当<数目>等于 0 时，则将此移位指令视为 **NOP** 操作。

SLW：移位数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 -255。移位数目大于 16 时，始终产生相同的结果：ACCU 1-L = 0、CC 1 = 0、CC 0 = 0 和 OV = 0。当 0 < 移位数目 <= 16 时，复位状态字的位 CC 0 和 OV 为 0。当移位数目为 0 时，则将移位指令视为 **NOP** 操作。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **SLW 5** 前	0101	1111	0110	0100	0101	1101	0011	1011
执行 **SLW 5** 后	0101	1111	0110	0100	1010	0111	0110	0000

实例 1

STL		解释
L	MW4	//将值载入 ACCU 1 中。
SLW	5	//将 ACCU 1 中的位向左移动 5 位。
T	MW8	//将结果传送到 MW8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MW20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MW20 的数值装载到 ACCU 1 中。
SLW		//移位数目是 ACCU 2-L-L 的数值 => 将 ACCU 1-L 中的位向左移动 3 位。
JP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.1.5 SRW 右移字 (16 位)

格式

SRW
SRW <数目>

地址	数据类型	描述
<数字>	整型、无符号	要移位的位数目，范围为 0 - 15

描述

SRW (右移字) 只逐位向右移动 ACCU 1-L 的内容。由零填充移位指令空出的位。将最后一个移出的位装载到状态字的 CC 1 位中。地址<数目>或 ACCU 2-L-L 中的数值指定要移位的位数目。

SRW <数目>：地址<数目>指定移位数目。允许的数值范围为 0 - 15。当<数目>大于 0 时，将状态字的位 CC 0 和 OV 复位为 0。当<数目>等于 0 时，则将此移位指令视为 **NOP** 操作。

SRW：移位数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 -255。移位数目大于 16 时，始终产生相同的结果：ACCU 1-L = 0、CC 1 = 0、CC 0 = 0 和 OV = 0。当 0 < 移位数目 <= 16 时，复位状态字的位 CC 0 和 OV 为 0。当移位数目为 0 时，则将移位指令视为 **NOP** 操作。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **SRW 6** 前	0101	1111	0110	0100	0101	1101	0011	1011
执行 **SRW 6** 后	0101	1111	0110	0100	0000	0001	0111	0100

实例 1

STL		解释
L	MW4	//将值载入 ACCU 1 中。
SRW	6	//将 ACCU 1-L 中的位向右移动 6 位。
T	MW8	//将结果传送到 MW8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MW20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MW20 的数值装载到 ACCU 1 中。
SRW		//移位数目是 ACCU 2-L-L 的数值 => 将 ACCU 1-L 中的位向右移动 3 位。
SPP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.1.6 SLD 左移双字 (32 位)

格式

SLD
SLD <数目>

地址	数据类型	描述
<数字>	整型、无符号	要移位的位数目，范围为 0 - 32

描述

SLD (左移双字) 逐位向左移动 ACCU 1 的整个内容。由零填充移位指令空出的位。将最后一个移出的位装载到状态字的 CC 1 位中。地址<数目>或 ACCU 2-L-L 中的数值指定要移位的位数目。

SLD <数目>：地址<数目>指定移位数目。允许的数值范围为 0 - 32。当<数目>大于 0 时，复位状态字的位 CC 0 和 OV 为 0。当<数目>等于 0 时，则将此移位指令视为 **NOP** 操作。

SLD：移位数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 -255。移位数目大于 32 时，始终产生相同的结果：ACCU 1 = 0、CC 1 = 0、CC 0 = 0 和 OV = 0。当 0 < 移位数目 <= 32 时，复位状态字的位 CC 0 和 OV 为 0。当移位数目为 0 时，则将移位指令视为 **NOP** 操作。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **SLD 5** 前	0101	1111	0110	0100	0101	1101	0011	1011
执行 **SLD 5** 后	1110	1100	1000	1011	1010	0111	0110	0000

实例 1

STL		解释
L	MD4	//将值载入 ACCU 1 中。
SLD	5	//将 ACCU 1 中的位向左移动 5 位。
T	MD8	//将结果传送到 MD8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MD20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MD20 的数值装载到 ACCU 1 中。
SLD		//移位数目是 ACCU 2-L-L 的数值 => 将 ACCU 1 中的位向左移动 3 位。
JP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.1.7 SRD 右移双字 (32 位)

格式

SRD
SRD <数目>

地址	数据类型	描述
<数字>	整型、无符号	要移位的位数目，范围为 0 - 32

描述

SRD (右移双字) 逐位向右移动 ACCU 1 的整个内容。由零填充移位指令空出的位。将最后一个移出的位装载到状态字的 CC 1 位中。地址<数目>或 ACCU 2-L-L 中的数值指定要移位的位数目。

SRD <数目>：地址<数目>指定移位数目。允许的数值范围为 0 - 32。当<数目>大于 0 时，将状态字的位 CC 0 和 OV 复位为 0。当<数目>等于 0 时，则将此移位指令视为 **NOP** 操作。

SRD：移位数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 -255。移位数目大于 32 时，始终产生相同的结果：ACCU 1 = 0、CC 1 = 0、CC 0 = 0 和 OV = 0。当 0 < 移位数目 <= 32 时，复位状态字的位 CC 0 和 OV 为 0。当移位数目为 0 时，则将移位指令视为 **NOP** 操作。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **SRD 7** 前	0101	1111	0110	0100	0101	1101	0011	1011
执行 **SRD 7** 后	0000	0000	1011	1110	1100	1000	1011	1010

实例 1

STL		解释
L	MD4	//将值载入 ACCU 1 中。
SRD	7	//将 ACCU 1 中的位向右移动 7 位。
T	MD8	//将结果传送到 MD8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MD20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MD20 的数值装载到 ACCU 1 中。
SRD		//移位数目是 ACCU 2-L-L 的数值 => 将 ACCU 1 中的位向右移动 3 位。
JP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.2 循环指令

11.2.1 循环移位指令概述

描述

可使用循环移位指令逐位左移或右移累加器 1 的整个内容 (参见 CPU 寄存器)。循环移位指令触发与 14.1 节所描述的移位指令相似的功能。然而，它使用累加器中移出的位的信号状态填充空出的位。

循环移位指令后的数字或在累加器 2 的低字低字节中的数值表示要循环移位的数目。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：

- RLD 循环左移双字 (32 位)
- RRD 循环右移双字 (32 位)
- RLDA 通过 CC 1 循环左移 ACCU 1 (32 位)
- RRDA 通过 CC 1 循环右移 ACCU 1 (32 位)

11.2.2 RLD 循环左移双字 (32 位)

格式

RLD
RLD <数目>

地址	数据类型	描述
<数字>	整型、无符号	要循环移位的位数目，范围为 0 - 32

描述

RLD (循环左移双字) 逐位向左循环移动 ACCU 1 的整个内容。循环移位指令空出的位由 ACCU 1 中移出位的信号状态填充。最后移出的位被装载到状态位 CC 1 中。而地址<数目>或 ACCU 2-L-L 中的数值则指定要循环移位的位数目。

RLD <数目>：地址<数目>指定循环移位的数目。允许的数值范围为 0 - 32。当<数目>大于 0 时，将状态字的位 CC 0 和 OV 复位为 0。当<数目>等于 0 时，则将此循环移位指令视为 **NOP** 操作。

RLD：循环移位的数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 - 255。当 ACCU 2-L-L 内的数值大于 0 时，将状态字的位 CC 0 和 OV 复位为 0。如果循环移位的数目为零，则将此循环移位指令视为 **NOP** 操作。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **RLD 4** 前	0101	1111	0110	0100	0101	1101	0011	1011
执行 **RLD 4** 后	1111	0110	0100	0101	1101	0011	1011	0101

实例 1

STL		解释
L	MD2	//将值载入 ACCU 1 中。
RLD	4	//将 ACCU 1 中的位向左循环移动 4 位。
T	MD8	//将结果传送到 MD8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MD20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MD20 的数值装载到 ACCU 1 中。
RLD		//循环移位数目是 ACCU 2-L-L 的值 => 将 ACCU 1 中的位向左循环移动 3 位。
JP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.2.3 RRD 循环右移双字 (32 位)

格式

RRD
RRD <数目>

地址	数据类型	描述
<数字>	整型、无符号	要循环移位的位数目，范围为 0 - 32

描述

RRD (循环右移双字) 逐位向右循环移动 ACCU 1 的整个内容。循环移位指令空出的位由 ACCU 1 中移出位的信号状态填充。最后移出的位被装载到状态位 CC 1 中。而地址<数目>或 ACCU 2-L-L 中的数值则指定要循环移位的位数目。

RRD <数目>：地址<数目>指定循环移位的数目。允许的数值范围为 0 - 32。当<数目>大于 0 时，将状态字的位 CC 0 和 OV 复位为 0。当<数目>等于 0 时，则将此循环移位指令视为 **NOP** 操作。

RRD：循环移位的数目由 ACCU 2-L-L 中的数值指定。可能的数值范围为 0 - 255。当 ACCU 2-L-L 的数值大于 0 时，复位状态字的位为 0。

状态字

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

实例

目录	ACCU1-H				ACCU1-L			
位	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **RRD 4** 前	0101	1111	0110	0100	0101	1101	0011	1011
执行 **RRD 4** 后	1011	0101	1111	0110	0100	0101	1101	0011

实例 1

STL		解释
L	MD2	//将值载入 ACCU 1 中。
RRD	4	//将 ACCU 1 中的位向右循环移动 4 位。
T	MD8	//将结果传送到 MD8。

实例 2

STL		解释
L	+3	//将数值+3 装载到 ACCU 1 中。
L	MD20	//将 ACCU 1 的内容装载到 ACCU 2 中。将 MD20 的数值装载到 ACCU 1 中。
RRD		//循环移位数目是 ACCU 2-L-L 的数值 => 将 ACCU 1 中的位向右循环移动 3 位。
JP	NEXT	//当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。

11.2.4 RLDA 通过 CC 1 循环左移 ACCU 1 (32 位)

格式

RLDA

描述

RLDA (通过 CC 1 循环左移双字) 通过 CC 1 将 ACCU 1 的整个内容循环左移 1 位。复位状态字的位 CC 0 和 OV 为 0。

状态字

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

实例

目录	CC 1	ACCU1-H				ACCU1-L			
位		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **RLDA** 前	**X**	**0**101	1111	0110	0100	0101	1101	0011	1011
执行 **RLDA** 后	**0**	1011	1110	1100	1000	1011	1010	0111	011**X**
	(X = 0 或 1，CC 1 之前的信号状态)								

```
STL          解释
L     MD2    //将 MD2 的数值装载到 ACCU 1 中。
RLDA         //通过 CC 1，将 ACCU 1 中的位循环左移 1 位。
JP    NEXT   //当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。
```

11.2.5 RRDA 通过 CC 1 循环右移 ACCU 1 (32 位)

格式

RRDA

描述

RRDA (通过 CC 1 循环右移双字) 将 ACCU 1 的整个内容循环向右移动 1 位。复位状态字的位 CC 0 和 OV 为 0。

状态字

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

实例

目录	CC 1	ACCU1-H				ACCU1-L			
位		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
执行 **RRDA** 前	**X**	0101	1111	0110	0100	0101	1101	0011	101**1**
执行 **RRDA** 后	**1**	**X**010	1111	1011	0010	0010	1110	1001	1101
	(X = 0 或 1，CC 1 之前的信号状态)								

```
STL          解释
L    MD2     //将 MD2 的数值装载到 ACCU 1 中。
RRDA         //通过 CC 1，将 ACCU 1 中的位循环右移 1 位。
JP   NEXT    //当最后一个移出的位 (CC 1) = 1 时，跳转到 NEXT 跳转标签。
```

12 定时器指令

12.1 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：

- FR 启用定时器 (自由)
- L 将当前定时器值作为整数载入 ACCU 1
- LC 将当前定时器值作为 BCD 载入 ACCU 1
- R 复位定时器
- SD 接通延迟定时器
- SE 扩展脉冲定时器
- SF 断开延时定时器
- SP 脉冲定时器
- SS 掉电保护接通延时定时器

12.2 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#txyz
  - 其中 t = 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#a**H**_b**M**_c**S**_d**MS**
  - 其中 H = 小时、M = 分钟、S = 秒、MS = 毫秒；
    用户变量为：a、b、c、d
  - 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。

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
1s	10
10s	11

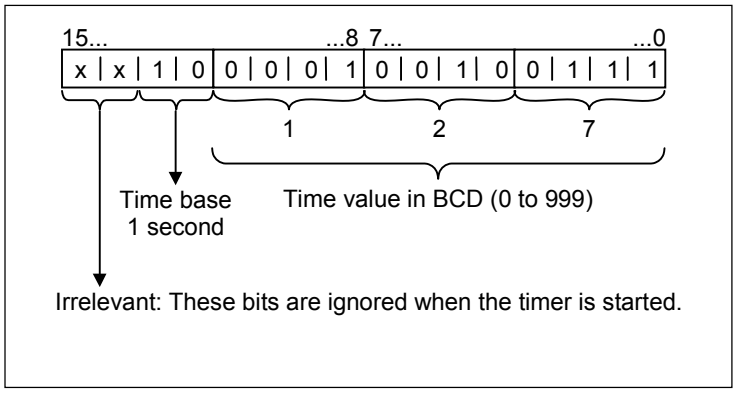
不接受超过 2 小时 46 分 30 秒的数值。对于范围限制 (例如，2h10ms) 而言，过高的分辨率将被截尾为有效分辨率。S5TIME 的通用格式对范围和分辨率有如下限制：

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

ACCU 1 中的位组态

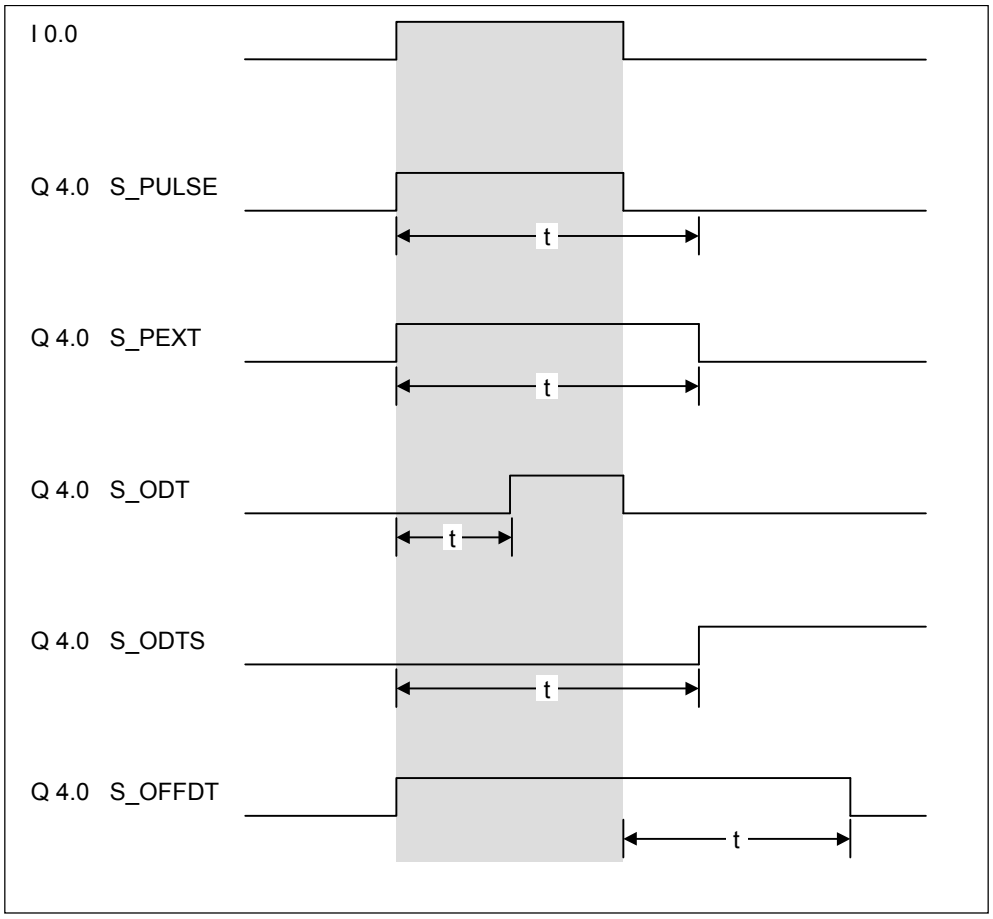
当启动定时器时，ACCU1 的内容将被用作时间值。ACCU1-L 的 0 到 11 位保留二进制编码的十进制格式时间值 (BCD 格式：由四位组成的每一组都包含一个十进制值的二进制代码)。第 12 和 13 位存放二进制编码的时间基准。

下图显示了装载定时器值 127 和 1 秒时间基准的 ACCU1-L 的内容：



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
S_PULSE 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。
S_PEXT 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。
S_ODT 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。
S_ODTS 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。
S_OFFDT 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。

12.3 FR 启用定时器 (自由)

格式

FR <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

当 RLO 从“0”跳转到为“1”时，**FR <定时器>**清除用于启动寻址定时器的边沿检测标记。启用指令 (FR) 前，RLO 位由 0 跳转到 1 即可启用定时器。

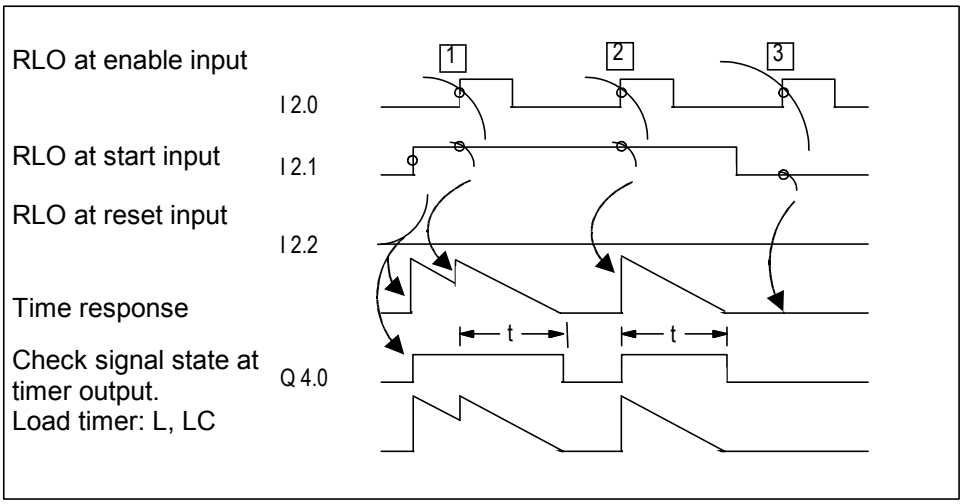
无论是启动定时器还是正常的定时器指令，都不需要定时器的启用。启用只适用于重触发一个正在运行的定时器，即重新启动定时器。只有在 RLO = 1 的情况下继续处理启动指令时，才可进行重新启动。

状态字

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

实例

```
STL                 解释
A     I 2.0
FR    T1            //启用定时器 T1。
A     I 2.1
L     S5T#10s       //在 ACCU 1 中预置 10s。
SI    T1            //启动定时器 T1 以作为脉冲定时器。
A     I 2.2
R     T1            //复位定时器 T1。
A     T1            //检查定时器 T1 的信号状态。
=     Q 4.0
L     T1            //以二进制的格式装载定时器 T1 的当前时间值。
T     MW10
```



t = programmed time interval

(1) 在定时器运行的同时 RLO 在启用输入处从 0 变为 1，会完全重新启动定时器。程序时间将用作重新启动的当前时间。则 RLO 在启用输入处从 1 变为 0 将不会有任何作用。

(2) 如果在定时器未运行时 RLO 在启用输入处从 0 变为 1，且在使能输入处仍有一个值为 1 的 RLO，则定时器也会作为脉冲以已编程的时间启动。

(3) 当使能输入处仍有值为 1 的 RLO 时，则 RLO 在启用输入处从 0 变为 1 对定时器毫无影响。

12.4 L 将当前定时器值作为整数载入 ACCU 1

格式

L <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

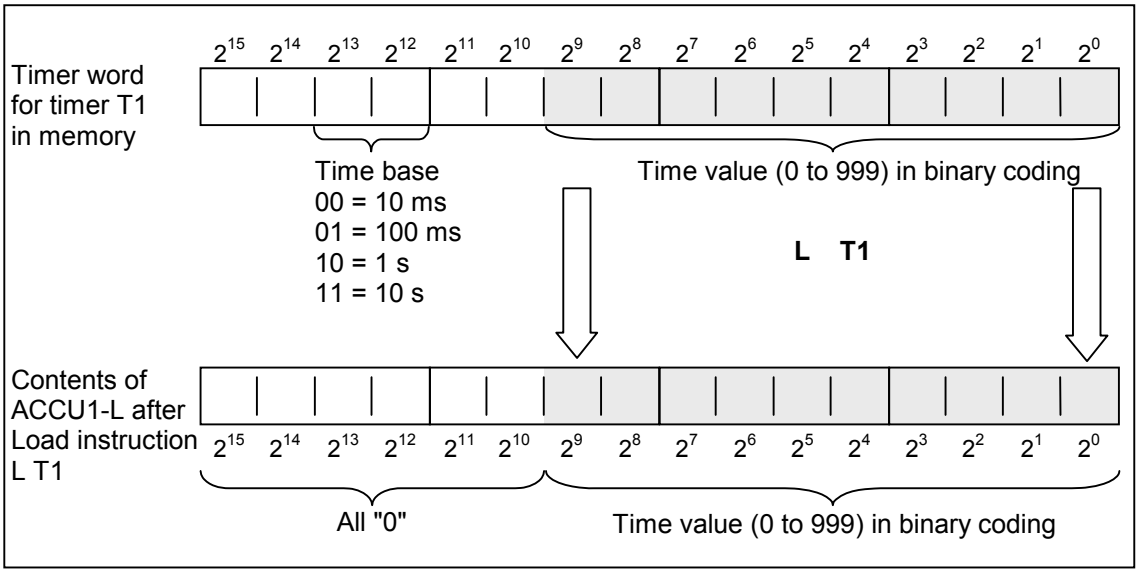
在将 ACCU 1 的内容存入 ACCU 2 中后，**L <定时器>** 会从没有时间基准的寻址定时器字中以二进制整数的形式将当前定时器值装入 ACCU 1-L。

状态字

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

实例

STL		解释
L	T1	//以二进制整数的形式为 ACCU 1 装载定时器 T1 的当前定时器值。



注意

L <定时器> 只将当前定时器值的二进制代码装入 ACCU1-L，而不装载时间基准。装载的时间为初始值减去自定时器启动后所消耗的时间。

12.5 LC 将当前定时器值作为 BCD 载入 ACCU 1

格式

LC <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

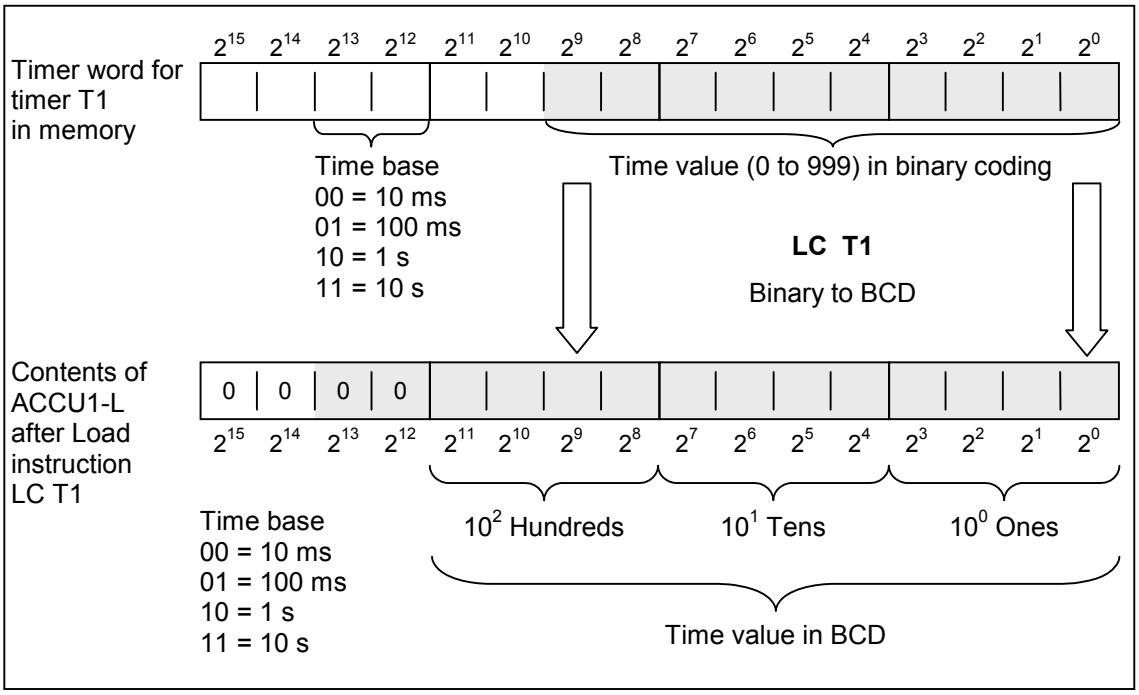
在将 ACCU 1 的内容存入 ACCU 2 中后，**LC <定时器>** 会从寻址定时器字中以二进制编码的十进制 (BCD) 数的形式将当前定时器值和时间基准装入 ACCU 1 中。

状态字

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

实例

STL		解释
LC	T1	//以二进制编码十进制 (BCD) 的格式为 ACCU 1-L 装载定时器 T1 的时间基准和当前定时器值。



12.6 R 复位定时器

格式

R <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

如果在 RLO 从 0 跳转到 1，**R <定时器>** 会停止当前计时功能并清除寻址定时器字的定时器值和时间基准。

状态字

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
写:	-	-	-	-	-	0	-	-	0

实例

```
STL                 解释
A     I 2.1
R     T1            //检查输入 I 2.1 的信号状态。如果 RLO 从 0 跳转到 1，则复位定时器 T1。
```

12.7 SP 脉冲定时器

格式

SP <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

SP <定时器> 在 RLO 从“0”跳转到“1”时启动寻址的定时器。只要 RLO = 1，程序时间间隔就会过去。如果在程序时间间隔截止之前 RLO 跳转到“0”，则停止计时器。此定时器启动指令要求将时间值和时间基准作为 BCD 数存储在 ACCU 1-L 中。

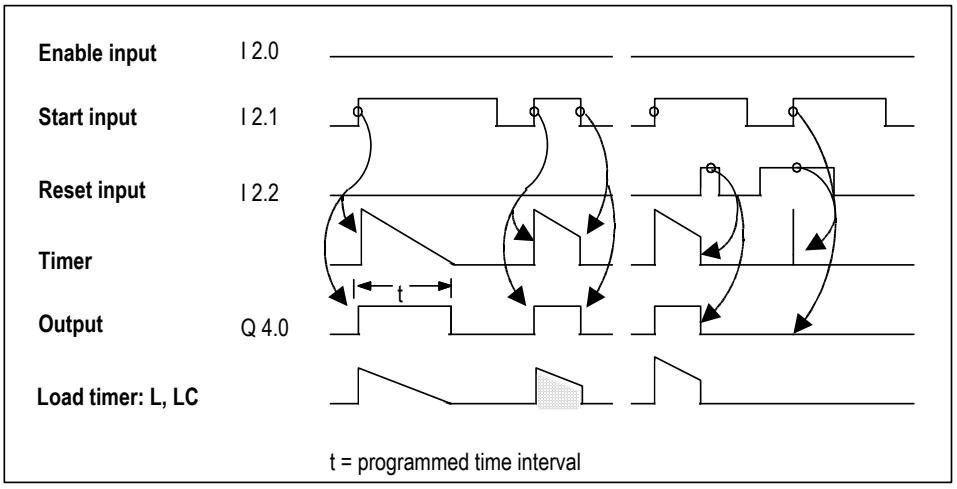
参见定时器在存储器中的位置与定时器组件。

状态字

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
写：	-	-	-	-	-	0	-	-	0

实例

```
STL                 解释
A     I 2.0
FR    T1            //启用定时器 T1。
A     I 2.1
L     S5T#10s       //在 ACCU 1 中预置 10s。
SP    T1            //启动定时器 T1 以作为脉冲定时器。
A     I 2.2
R     T1            //复位定时器 T1。
A     T1            //检查定时器 T1 的信号状态。
=     Q 4.0
L     T1            //以二进制形式装载定时器 T1 的当前时间值。
T     MW10
LC    T1            //以 BCD 形式装载定时器 T1 的当前时间值。
T     MW12
```



12.8 SE 扩展脉冲定时器

格式

SE <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

SE <定时器> 在 RLO 从“0”跳转到“1”时启动寻址的定时器。程序时间间隔会流逝，即使 RLO 在这段时间内跳转到“0”。如果在程序时间间隔截止之前 RLO 从“0”跳转到“1”，则重新开始程序时间间隔。此定时器启动指令要求将时间值和时间基准作为 BCD 数存储在 ACCU 1-L 中。

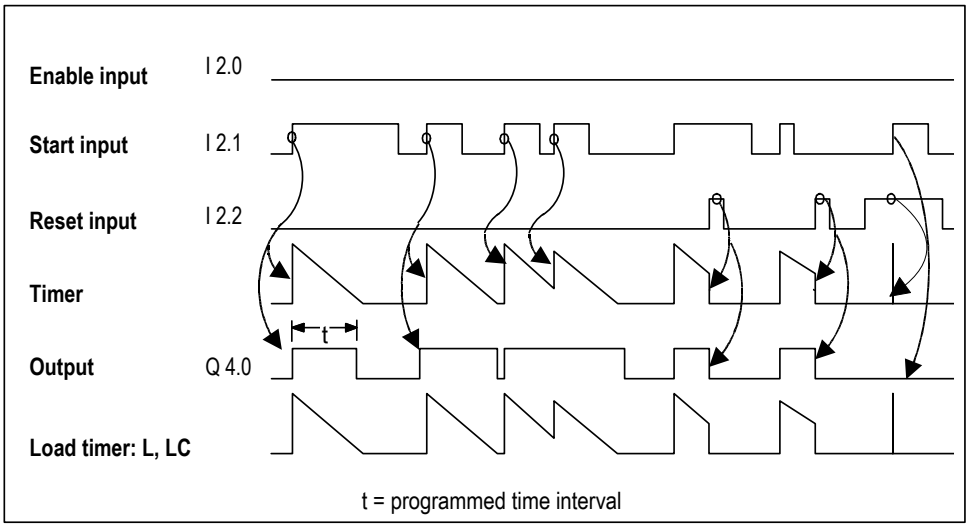
参见定时器在存储器中的位置与定时器组件。

状态字

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
写：	-	-	-	-	-	0	-	-	0

实例

STL		解释
A	I 2.0	
FR	T1	//启用定时器 T1。
A	I 2.1	
L	S5T#10s	//在 ACCU 1 中预置 10s。
SE	T1	//将定时器 T1 启动为扩展脉冲定时器。
A	I 2.2	
R	T1	//复位定时器 T1。
A	T1	//检查定时器 T1 的信号状态。
=	Q 4.0	
L	T1	//以二进制形式装载定时器 T1 的当前定时器值。
T	MW10	
LC	T1	//以 BCD 形式装载定时器 T1 的当前定时器值。
T	MW12	



12.9 SD 接通延迟定时器

格式

SD <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

在 RLO 从“0”跳转到“1”时，**SD <定时器>** 启动寻址的定时器。只要 RLO = 1，程序时间间隔就会流逝。如果在程序时间间隔截止之前 RLO 跳转到“0”，则停止计时。此定时器启动指令要求将时间值和时间基准作为 BCD 数存储在 ACCU 1-L 中。

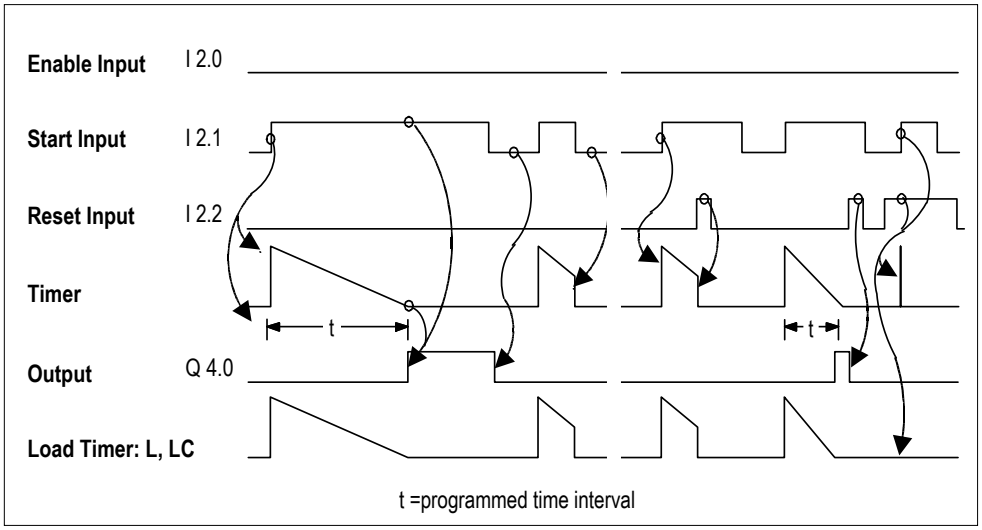
参见定时器在存储器中的位置与定时器组件。

状态字

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
写:	-	-	-	-	-	0	-	-	0

实例

STL		解释
A	I 2.0	
FR	T1	//启用定时器 T1。
A	I 2.1	
L	S5T#10s	//在 ACCU 1 中预置 10s。
SD	T1	//启动定时器 T1 作为接通延时定时器。
A	I 2.2	
R	T1	//复位定时器 T1。
A	T1	//检查定时器 T1 的信号状态。
=	Q 4.0	
L	T1	//以二进制形式装载定时器 T1 的当前定时器值。
T	MW10	
LC	T1	//以 BCD 形式装载定时器 T1 的当前定时器值。
T	MW12	



12.10 SS 掉电保护接通延时定时器

格式

SS <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

SS <定时器> (将定时器启动为带保持的接通定时器) 在 RLO 从“0”跳转到“1”时启动寻址的定时器。完整的程序时间间隔会流逝，即使 RLO 在这段时间内跳转到“0”。如果在程序时间间隔截止之前 RLO 从“0”跳转到“1”，则重新触发程序时间间隔 (重新启动)。此定时器启动指令要求将时间值和时间基准作为 BCD 数存储在 ACCU 1-L 中。

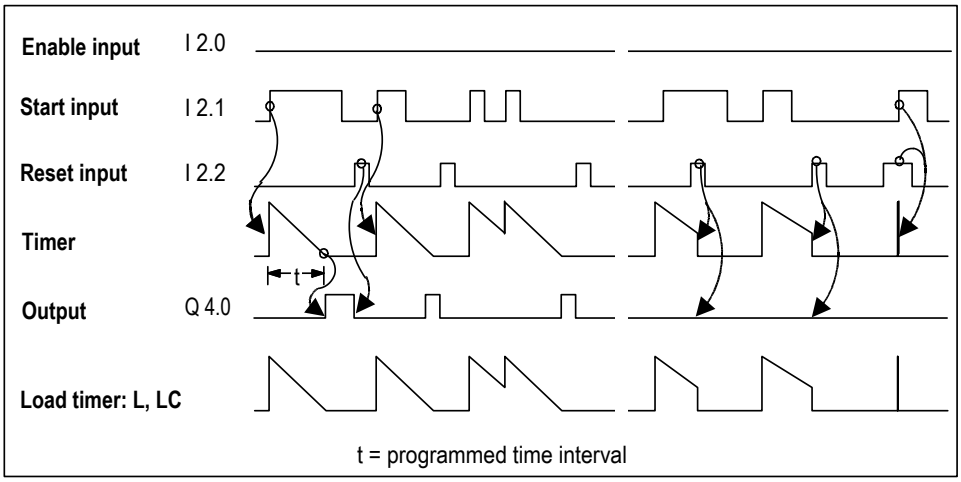
参见定时器在存储器中的位置与定时器组件。

状态字

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
写：	-	-	-	-	-	0	-	-	0

实例

STL		解释
A	I 2.0	
FR	T1	//启用定时器 T1。
A	I 2.1	
L	S5T#10s	//在 ACCU 1 中预置 10s。
SS	T1	//将定时器 T1 启动为带保持的接通延时定时器。
A	I 2.2	
R	T1	//复位定时器 T1。
A	T1	//检查定时器 T1 的信号状态。
=	Q 4.0	
L	T1	//以二进制形式装载定时器 T1 的当前时间值。
T	MW10	
LC	T1	//以 BCD 形式装载定时器 T1 的当前时间值。
T	MW12	



12.11 SF 断开延时定时器

格式

SF <定时器>

地址	数据类型	存储区域	描述
<定时器>	TIMER	T	定时器编号、范围取决于 CPU

指令描述

SF <定时器> 在 RLO 从“1”跳转到“0”时启动寻址的定时器。只要 RLO = 1，程序时间间隔就会流逝。如果在程序时间间隔截止之前 RLO 跳转到“1”，则停止计时。此定时器启动指令要求将时间值和时间基准作为 BCD 数存储在 ACCU 1-L 中。

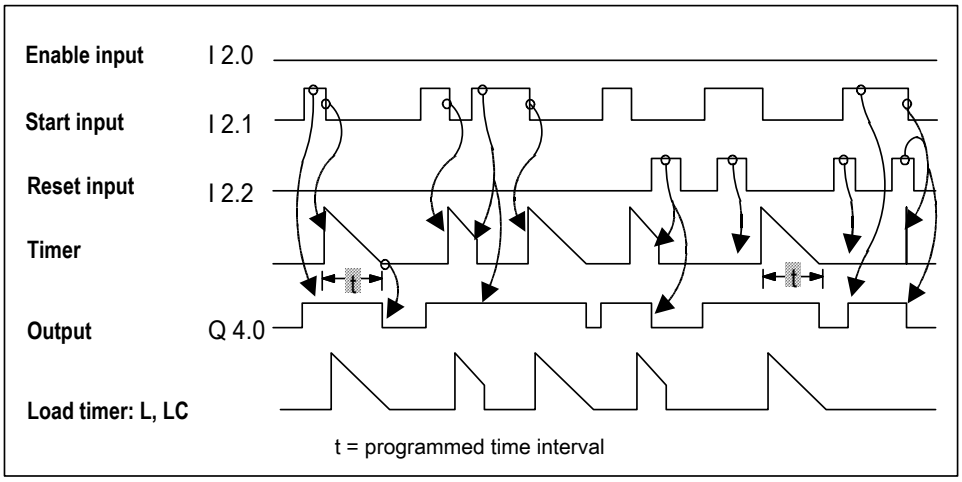
参见定时器在存储器中的位置与定时器组件。

状态字

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
写：	-	-	-	-	-	0	-	-	0

实例

```
STL                 解释
A     I 2.0
FR    T1            //启用定时器 T1。
A     I 2.1
L     S5T#10s       //在 ACCU 1 中预置 10s。
SF    T1            //将定时器 T1 启动为断开延时定时器。
A     I 2.2
R     T1            //复位定时器 T1。
A     T1            //检查定时器 T1 的信号状态。
=     Q 4.0
L     T1            //以二进制形式装载定时器 T1 的当前定时器值。
T     MW10
LC    T1            //以 BCD 形式装载定时器 T1 的当前定时器值。
T     MW12
```



13 字逻辑指令

13.1 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。每个字或双字必须位于两个累加器其中一个之内。

对于字而言，累加器 2 的低字中的内容会与累加器 1 的低字中的内容组合。组合结果存储在累加器 1 的低字中，同时覆盖原有的内容。

对于双字而言，累加器 2 的内容与累加器 1 的内容相组合。组合结果存储在累加器 1 中，同时覆盖原有的内容。

如结果不等于 0，则将状态字的位 CC 1 置为“1”。 如结果等于 0，则将状态字的位 CC 1 置为“0”。

要执行字逻辑运算，可使用下列指令：

- AW 单字与运算 (16 位)
- OW 单字或运算 (16 位)
- XOW 单字异或运算 (16 位)
- AD 双字与运算 (32 位)
- OD 双字或运算 (32 位)
- XOD 双字异或运算 (32 位)

13.2 AW 单字与运算 (16 位)

格式

AW
AW <常数>

地址	数据类型	描述
<常数>	WORD， 16 位常数	使用与运算 (AND) 同 ACCU 1-L 组合的位模式

指令描述

AW (单字与运算) 根据布尔逻辑与运算，将 ACCU 1-L 的内容与 ACCU 2-L 的内容或与 16 位常数逐位进行组合。只有当两个在逻辑运算中组合的两个字的相应位都为“1”时，结果字中的位才为“1”。结果存储在 ACCU 1-L 中。ACCU 1-H 和 ACCU 2 (以及 ACCU 3 和 ACCU 4，对于具有四个 ACCU 的 CPU) 保持不变。状态位 CC 1 被置为运算结果 (如结果不等于零，则 CC 1 = 1)。复位状态字的位 CC 0 和 OV 为 0。

AW：组合 ACCU 1-L 和 ACCU 2-L。

AW <常数>：将 ACCU 1 和 16 位常数组合。

状态字

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

实例

位	15 . . .	. .	. .	. . . 0
执行 **AW** 前的 ACCU 1-L	0101	1001	0011	1011
ACCU 2-L 或 16 位常数：	1111	0110	1011	0101
执行 **AW** 后的结果 (ACCU 1-L)	0101	0000	0011	0001

实例 1

```
STL                 解释
L     IW20          //将 IW20 的内容装入 ACCU 1-L。
L     IW22          //将 ACCU 1 的内容装载到 ACCU 2 中。将 IW22 的内容装入 ACCU 1-L。
AW                  //使用与运算将 ACCU 1-L 中的位与 ACCU 2-L 的位相组合，将结果存储在 ACCU 1-L 中。
T     MW 8          //将结果传送到 MW8。
```

实例 2

```
STL                 解释
L     IW20          //将 IW20 的内容装入 ACCU 1-L。
AW    W#16#0FFF     //使用与运算，将 ACCU 1-L 的位与 16 位常数 (0000_1111_1111_1111) 的位模式组合，将结
                    //果存储在 ACCU 1-L 中。
JP    NEXT          //如结果不等于零，则跳转到下一个跳转标签 (CC 1 = 1)。
```

13.3 OW 单字或运算 (16 位)

格式

OW
OW <常数>

地址	数据类型	描述
<常数>	WORD， 16 位常数	使用或运算 (OR) 与 ACCU 1-L 组合的位模式

指令描述

OW (单字或运算) 根据布尔逻辑或运算，将 ACCU 1-L 的内容与 ACCU 2-L 的内容或与 16 位常数逐位组合。只有当在逻辑运算中组合的两个字的相应位中至少有一位是“1”时，结果字中的位才为“1”。结果存储在 ACCU 1-L 中。ACCU 1-H 和 ACCU 2 (以及 ACCU 3 和 ACCU 4，对于具有四个 ACCU 的 CPU) 保持不变。执行指令时既不考虑也不会影响 RLO。状态位 CC 1 被置为运算结果 (如结果不等于零，则 CC 1 = 1)。复位状态字的位 CC 0 和 OV 为 0。

OW：组合 ACCU 1-L 和 ACCU 2-L。

OW <常数>：将 ACCU 1-L 与 16 位常数组合。

状态字

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

实例

位	15 . . .	. .	. .	. . . 0
执行 **OW** 前的 ACCU 1-L	0101	0101	0011	1011
ACCU 2-L 或 16 位常数：	1111	0110	1011	0101
执行 **OW** 后的结果 (ACCU 1-L)	1111	0111	1011	1111

实例 1

```
STL                 解释
L     IW20          //将 IW20 的内容装入 ACCU 1-L。
L     IW22          //将 ACCU 1 的内容装载到 ACCU 2 中。将 IW22 的内容装入 ACCU 1-L。
OW                  //使用或运算，将 ACCU 1-L 中的位与 ACCU 2-L 的位组合，将结果存储在 ACCU 1-L 中。
T     MW8           //将结果传送到 MW8。
```

实例 2

```
STL                 解释
L     IW20          //将 IW 20 的内容装入 ACCU 1-L。
OW    W#16#0FFF     //使用或运算，将 ACCU 1-L 的位与 16 位常数 (0000_1111_1111_1111) 的位模式相组合，
                    //将结果存储在 ACCU 1-L 中。
JP    NEXT          //如结果不等于零，则跳转到下一个跳转标签 (CC 1 = 1)。
```

13.4 XOW 单字异或运算 (16 位)

格式

XOW
XOW <常数>

地址	数据类型	描述
<常数>	WORD， 16 位常数	使用异或 (XOR) 运算与 ACCU 1-L 组合的位模式

指令描述

XOW (单字异或运算) 根据布尔逻辑异或运算，将 ACCU 1-L 的内容与 ACCU 2-L 的内容或与 16 位的常数逐位组合。只有当两个在逻辑运算中组合的字的相应位中有一位是“1”时，结果字中的位才为“1”。结果存储在 ACCU 1-L 中。ACCU 1-H 和 ACCU 2 保持不变。状态位 CC 1 被置为运算结果 (如结果不等于零，则 CC 1 = 1)。复位状态字的位 CC 0 和 OV 为 0。

可多次使用异或运算函数。如果所选中地址有奇数个“1”，则逻辑运算结果为“1”。

XOW：组合 ACCU 1-L 和 ACCU 2-L。

XOW <常数>：将 ACCU 1-L 与 16 位常数组合。

状态字

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

实例

位	15 . . .	. .	. .	. . . 0
执行 **XOW** 前的 ACCU 1	0101	0101	0011	1011
ACCU 2-L 或 16 位常数：	1111	0110	1011	0101
执行 **XOW** 后的结果 (ACCU 1)	1010	0011	1000	1110

实例 1

STL		解释
L	IW20	//将 IW20 的内容装入 ACCU 1-L。
L	IW22	//将 ACCU 1 的内容装载到 ACCU 2 中。将 ID24 的内容装入 ACCU 1-L。
XOW		//使用异或运算，将 ACCU 1-L 的位与 ACCU 2-L 的位组合，将结果存储在 ACCU 1-L 中。
T	MW8	//将结果传送到 MW8。

实例 2

STL		解释
L	IW20	//将 IW20 的内容装入 ACCU 1-L。
XOW	16#0FFF	//使用异或运算，将 ACCU 1-L 的位与 16 位常数 (0000_1111_1111_1111) 的位模式相组合， //将结果存储在 ACCU 1-L 中。
JP	NEXT	//如结果不等于零，则跳转到下一个跳转标签 (CC 1 = 1)。

13.5 AD 双字与运算 (32 位)

格式

AD
AD <常数>

地址	数据类型	描述
<常数>	DWORD， 32 位常数	使用与运算 (AND) 与 ACCU 1 组合的位模式

指令描述

AD (双字与运算) 根据布尔逻辑与运算，将 ACCU 1 的内容与 ACCU 2 的内容或与 32 位的常数逐位进行组合。只有当在逻辑运算中组合的两个双字的相应位都为“1”时，结果双字中的位才为“1”。结果存储在 ACCU 1-L 中。ACCU 2 (以及 ACCU 3 和 ACCU 4，对于具有四个 ACCU 的 CPU) 保持不变。状态位 CC 1 被置为运算结果 (如结果不等于零，则 CC 1 = 1)。复位状态字的位 CC 0 和 OV 为 0。

AD：将 ACCU 1 与 ACCU 2 组合。

AD <常数>：将 ACCU 1 和 32 位常数组合。

状态字

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

实例

位	31 . .	. .	. .	. .	. .	. .	. .	. . 0
执行 **UD** 前的 ACCU 1	0101	0000	1111	1100	1000	1001	0011	1011
ACCU 2 或 32 位常数	1111	0011	1000	0101	0111	0110	1011	0101
执行 **UD** 后的结果 (ACCU 1)	0101	0000	1000	0100	0000	0000	0011	0001

实例 1

```
STL                     解释
L     ID20              //将 ID20 的内容装入 ACCU 1。
L     ID24              //将 ACCU 1 的内容装载到 ACCU 2 中。将 ID24 的内容装入 ACCU 1。
AD                      //使用与运算将 ACCU 1 中的位与 ACCU 2 中的位组合，将结果存储在 ACCU 1 中。
T     MD8               //将结果传送到 MD8。
```

实例 2

```
STL                     解释
L     ID 20             //将 ID20 的内容装入 ACCU 1。
AD    DW#16#0FFF_EF21   //使用与运算将 ACCU 1 的位与 32 位常数
                        //(0000_1111_1111_1111_1110_1111_0010_0001) 的位模式组合，
                        //将结果存储在 ACCU 1 中。
JP    NEXT              //如结果不等于零，则跳转到下一个跳转标签 (CC 1 = 1)。
```

13.6 OD 双字或运算 (32 位)

格式

OD
OD <常数>

地址	数据类型	描述
<常数>	DWORD， 32 位常数	使用或运算 (OR) 与 ACCU 1 组合的位模式

指令描述

OD (双字或运算) 根据布尔逻辑或运算，将 ACCU 1 的内容与 ACCU 2 的内容或与 32 位常数逐位组合。只有当在逻辑运算中组合的两个双字的相应位中至少有一位是“1”时，结果双字中的位才为“1”。结果存储在 ACCU 1-L 中。ACCU 2 (以及 ACCU 3 和 ACCU 4，对于具有四个 ACCU 的 CPU) 保持不变。状态位 CC 1 被置为运算结果的函数 (如结果不等于零，则 CC 1 = 1)。复位状态字的位 CC 0 和 OV 为 0。

OD：将 ACCU 1 与 ACCU 2 组合。

OD <常数>：将 ACCU 1 和 32 位常数组合。

状态字

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

实例

位	31 . .	. .	. .	. .	. .	. .	. .	. . . 0
执行 **OD** 前的 ACCU 1	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 或 32 位常数：	1111	0011	1000	0101	0111	0110	1011	0101
执行 **OD** 后的结果 (ACCU 1)	1111	0011	1111	1101	1111	0111	1011	1111

实例 1

STL		解释
L	ID20	//将 ID20 的内容装入 ACCU 1。
L	ID24	//将 ACCU 1 的内容装载到 ACCU 2 中。将 ID24 的内容装入 ACCU 1。
OD		//用或运算，将 ACCU 1 中的位与 ACCU 2 中的位组合，将结果存储在 ACCU 1 中。
T	MD8	//将结果传送到 MD8。

实例 2

STL		解释
L	ID20	//将 ID20 的内容装入 ACCU 1。
OD	DW#16#0FFF_EF21	//使用或运算，将 ACCU 1 中的位与 32 位常数
		//(0000_1111_1111_1111_1110_1111_0010_0001) 的位模式组合，将结果存储在
		//ACCU 1 中。
JP	NEXT	//如结果不等于零，则跳转到下一个跳转标签 (CC 1 = 1)。

13.7 XOD 双字异或运算 (32 位)

格式

XOD
XOD <常数>

地址	数据类型	描述
<常数>	DWORD， 32 位常数	使用异或 (XOR) 运算与 ACCU 1 组合的位模式。

指令描述

XOD (双字异或运算) 根据布尔逻辑 XOR (异或) 运算，将 ACCU 1 的内容与 ACCU 2 的内容或与 32 位常数逐位组合。当在逻辑运算中组合的两个双字的相应位中仅有一位是“1”时，结果双字中的位才为“1”。结果存储在 ACCU 1-L 中。ACCU 2 保持不变。状态位 CC 1 被置为运算的结果 (如结果不等于零，则 CC 1 = 1)。复位状态字的位 CC 0 和 OV 为 0。

可多次使用异或运算函数。如果所选中地址有奇数个“1”，则逻辑运算结果为“1”。

XOD：将 ACCU 1 与 ACCU 2 组合。

XOD <常数>：将 ACCU 1 和 32 位常数组合。

状态字

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

实例

位	31 . .	. .	. .	. .	. .	. .	. .	. . . 0
执行 **XOD** 前的 ACCU 1	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 或 32 位常数：	1111	0011	1000	0101	0111	0110	1011	0101
执行 **XOD** 后的结果 (ACCU 1)	1010	0011	0111	1001	1111	0011	1000	1110

实例 1

STL		解释
L	ID20	//将 ID20 的内容装入 ACCU 1。
L	ID24	//将 ACCU 1 的内容装载到 ACCU 2 中。将 ID24 的内容装入 ACCU 1。
XOD		//使用异或运算，将 ACCU 1 中的位与 ACCU 2 的位组合，将结果存储在 ACCU 1 中。
T	MD8	//将结果传送到 MD8。

实例 2

STL		解释
L	ID20	//将 ID20 的内容装入 ACCU 1。

XOD	DW#16#0FFF_EF21	//使用异或运算，将 ACCU 1 中的位与 32 位常数
		//(0000_1111_1111_1111_1111_1110_0010_0001) 的位模式组合，将结果存储在
		//ACCU 1 中。

JP	NEXT	//如结果不等于零，则跳转到下一个跳转标签 (CC 1 = 1)。

14 累加器指令

14.1 累加器和地址寄存器指令概述

描述

下列指令可用于处理一个或两个累加器的内容：

- TAK 将 ACCU 1 与 ACCU 2 互换
- PUSH 具有两个 ACCU 的 CPU
- PUSH 具有四个 ACCU 的 CPU
- POP 具有两个 ACCU 的 CPU
- POP 具有四个 ACCU 的 CPU

- ENT 进入 ACCU 堆栈
- LEAVE 离开 ACCU 堆栈
- INC 增加 ACCU 1-L-L
- DEC 减少 ACCU 1-L-L

- +AR1 将 ACCU 1 加到地址寄存器 1
- +AR2 将 ACCU 1 加到地址寄存器 2

- BLD 程序显示指令 (空)
- NOP 0 空指令
- NOP 1 空指令

14.2 TAK 将 ACCU 1 与 ACCU 2 互换

格式

TAK

描述

TAK (将 ACCU 1 与 ACCU 2 互换) 将把 ACCU 1 的内容与 ACCU 2 的内容交换。该指令的执行与状态位无关，对状态位也没有影响。对具有四个 ACCU 的 CPU，ACCU 3 和 ACCU 4 的内容保持不变。

状态字

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

实例：从较大值中减去较小值

```
STL                     解释
        L     MW10      //将 MW10 的内容载入 ACCU 1-L。
        L     MW12      //将 ACCU 1-L 的内容装载到 ACCU 2-L 中。将 MW12 的内容载入 ACCU 1-L。
        >I              //检查 ACCU 2-L (MW10) 是否大于 ACCU 1-L (MW12)。
        SPB   NEXT      //如果 ACCU 2 (MW10) 大于 ACCU 1 (MW12)，则跳转到 NEXT 跳转标签。
        TAK             //将 ACCU 1 的内容与 ACCU 2 的内容交换
NEXT:   -I              //从 ACCU 1-L 的内容中减去 ACCU 2-L 的内容。
        T     MW14      //将结果 (= 较大值减较小值) 传送到 MW14。
```

目录	ACCU 1	ACCU 2
执行 **TAK** 指令之前	<MW12>	<MW10>
执行 **TAK** 指令之后	<MW**10**>	<MW**12**>

14.3 POP 具有两个 ACCU 的 CPU

格式

POP

描述

POP (具有两个 ACCU 的 CPU) 将 ACCU 2 的全部内容复制到 ACCU 1。ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL             解释
T    MD10       //将 ACCU 1 的内容 (= 值 A) 传送到 MD10
POP             //将 ACCU 2 的整个内容复制到 ACCU 1
T    MD14       //将 ACCU 1 的内容 (= 值 B) 传送到 MD14
```

目录	ACCU 1	ACCU 2
执行 **POP** 指令之前	值 A	值 B
执行 **POP** 指令之后	值 B	值 B

14.4 POP 具有四个 ACCU 的 CPU

格式

POP

描述

POP (具有四个 ACCU 的 CPU) 将 ACCU 2 的全部内容复制到 ACCU 1，将 ACCU 3 的内容复制到 ACCU 2，并将 ACCU 4 的内容复制到 ACCU 3。ACCU 4 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL             解释
T     MD10      //将 ACCU 1 的内容 (= 值 A) 传送到 MD10
POP             //将 ACCU 2 的整个内容复制到 ACCU 1
T     MD14      //将 ACCU 1 的内容 (= 值 B) 传送到 MD14
```

目录	ACCU 1	ACCU 2	ACCU 3	ACCU 4
执行 **POP** 指令之前	值 A	值 B	值 C	值 D
执行 **POP** 指令之后	值 B	值 C	值 D	值 D

14.5 PUSH 具有两个 ACCU 的 CPU

格式

PUSH

描述

PUSH (ACCU 1 到 ACCU 2) 将 ACCU 1 的整个内容复制到 ACCU 2。ACCU 1 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL             解释
L      MW10     //将 MW10 的内容载入 ACCU 1。
PUSH            //将 ACCU 1 的整个内容复制到 ACCU 2
```

目录	ACCU 1	ACCU 2
执行 **PUSH** 指令之前	<MW10>	<X>
执行 **PUSH** 指令之后	<MW10>	<MW10>

14.6 PUSH 具有四个 ACCU 的 CPU

格式

PUSH

描述

PUSH (具有四个 ACCU 的 CPU) 将 ACCU 3 的内容复制到 ACCU 4，将 ACCU 2 的内容复制到 ACCU 3，并将 ACCU 1 的内容复制到 ACCU 2。ACCU 1 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

状态字

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

实例

```
STL             解释
L     MW10      //将 MW10 的内容载入 ACCU 1。
PUSH            //将 ACCU 1 的整个内容复制到 ACCU 2，将 ACCU 2 的内容复制到 ACCU 3，
                //并将 ACCU 3 的内容复制到 ACCU 4。
```

目录	ACCU 1	ACCU 2	ACCU 3	ACCU 4
执行 **PUSH** 指令之前	值 A	值 B	值 C	值 D
执行 **PUSH** 指令之后	值 A	值 A	值 B	值 C

14.7 ENT 进入 ACCU 堆栈

格式

ENT

描述

ENT (进入累加器堆栈) 将把 ACCU 3 的内容复制到 ACCU 4，并将 ACCU 2 的内容复制到 ACCU 3。如果在装载指令前面直接编程 ENT 指令，则可将中间结果保存到 ACCU 3 中。

实例

```
STL             解释
L     DBD0      //从数据双字 DBD0 中将值载入 ACCU 1。//(该值必须为浮点数格式。)
L     DBD4      //将值从 ACCU 1 复制到 ACCU2。从数据双字 DBD4 中将值载入 ACCU 1。 (该值必须以浮点格式表示)。
+R              //将 ACCU 1 和 ACCU 2 的内容作为浮点数 (32 位，IEEE 754) 相加，并将结果保存到 ACCU 1 中。
L     DBD8      //将值从 ACCU 1 复制到 ACCU 2，并从数据双字 DBD8 中将值载入 ACCU 1。
ENT             //将 ACCU 3 的内容复制到 ACCU 4。将 ACCU 2 的内容 (中间结果) 复制到 ACCU 3。
L     DBD12     //从数据双字 DBD12 中将值载入 ACCU 1。
-R              //从 ACCU 2 的内容中减去 ACCU 1 的内容，并将结果保存在 ACCU 1 中。将 ACCU 3 的内容复制到 ACCU 2。
                //将 ACCU 4 的内容复制到 ACCU 3。
/R              //将 ACCU 2 (DBD0 + DBD4) 的内容除以 ACCU 1 (DBD8 - DBD12) 的内容。将结果保存
                //在 ACCU 1 中。
T     DBD16     //将结果 (ACCU 1) 传送到数据双字 DBD16。
```

14.8 LEAVE 离开 ACCU 堆栈

格式

LEAVE

描述

LEAVE (离开累加器堆栈) 将 ACCU 3 的内容复制到 ACCU 2，并将 ACCU 4 的内容复制到 ACCU 3。如果在移位或循环指令前面直接编程 LEAVE 指令，并将各累加器组合，则 leave 指令就可起到数学运算指令的作用。ACCU 1 的内容和 ACCU 4 的内容保持不变。

14.9 INC 增加 ACCU 1-L-L

格式

INC <8 位整数>

参数	数据类型	描述
<8 位整数>	8 位整型常数	加到 ACCU 1-L-L 上的常数，范围从 0 至 255

描述

INC <8 位整数> (增加 ACCU 1-L-L) 将 ACCU 1-L-L 的内容加上 8 位整数，并将结果存储在 ACCU 1-L-L 中。ACCU 1-L-H、ACCU 1-H 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

注意

这些指令不适用于 16 位或 32 位算术运算，因为没有从累加器 1 的低字的低字节到低字的高字节的进位。对 16 位或 32 位算术运算，分别使用+I 或+D 指令。

状态字

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

实例

```
STL             解释
L     MB22      //载入 MB22 的值
INC   1         //指令“ACCU 1 (MB22) 加 1”；将结果存储在 ACCU 1-L-L 中
T     MB22      //将 ACCU 1-L-L 的内容 (结果) 传送回 MB22。
```

14.10 DEC 减少 ACCU 1-L-L

格式

DEC <8 位整数>

地址	数据类型	描述
<8 位整数>	8 位整型常数	ACCU 1-L-L 减去常数，范围从 0 至 255

描述

DEC <8 位整数> (减少 ACCU 1-L-L) 从 ACCU 1-L-L 的内容中减去 8 位整数，并将结果存储在 ACCU 1-L-L 中。ACCU 1-L-H、ACCU 1-H 和 ACCU 2 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

注意

这些指令不适用于 16 位或 32 位算术运算，因为没有从累加器 1 的低字的低字节到低字的高字节的进位。对 16 位或 32 位算术运算，分别使用+I 或+D 指令。

状态字

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

实例

```
STL             解释
L     MB250     //载入 MB250 的值
DEC   1         //指令“ACCU 1-L-L 减 1”，将结果存储在 ACCU 1-L-L 中。
T     MB250     //将 ACCU 1-L-L 的内容（结果）传送回 MB250。
```

14.11 +AR1 将 ACCU 1 加到地址寄存器 1

格式

+AR1
+AR1 <P#Byte.Bit>

参数	数据类型	描述
<P#Byte.Bitr>	指针常量	被加到 AR1 上的地址

描述

+AR1 (加到 AR1) 将在语句或 ACCU 1-L 中指定的偏移量加到 AR1 的内容上。首先将整数 (16 位) 扩展为符号正确的 24 位，然后将其加到 AR1 的最低有效的 24 位 (AR1 中的相对地址的一部分)。在 AR1 中，区域 ID 的部分 (位 24、25 和 26) 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

+AR1：要加到 AR1 的内容中的整数 (16 位) 由 ACCU 1-L 中的值指定。允许的值范围为 -32768 至 +32767。

+AR1 <P#Byte.Bit>：要加的偏移量由 <P#Byte.Bit> 地址指定。

状态字

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

实例 1

```
STL             解释
L     +300      //将值载入 ACCU 1-L
+AR1            //将 ACCU 1-L (整数，16 位) 加到 AR1。
```

实例 2

```
STL             解释
+AR1  P#300.0   //将偏移量 300.0 加到 AR1。
```

14.12 +AR2 将 ACCU 1 加到地址寄存器 2

格式

+AR2
+AR2 <P#Byte.Bit>

参数	数据类型	描述
<P#Byte.Bitr>	指针常量	被加到 AR2 上的地址

描述

+AR2 (加到 AR2) 将在指令或 ACCU 1-L 中指定的偏移量加到 AR2 的内容中。首先将整数 (16 位) 扩展为符号正确的 24 位，然后将其加到 AR2 的最低有效的 24 位 (AR2 中的相对地址的一部分)。在 AR2 中，区域 ID 的部分 (位 24、25 和 26) 保持不变。该指令的执行与状态位无关，对状态位也没有影响。

+AR2：要加到 AR2 的内容中的整数 (16 位) 由 ACCU 1-L 中的值指定。允许的值范围为 -32768 至 +32767。

+AR2 <P#Byte.Bit>：要加的偏移量由 <P#Byte.Bit> 地址指定。

状态字

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

实例 1

```
STL             解释
L     +300      //载入 ACCU 1-L 中的值。
+AR1            //将 ACCU 1-L (整数，16 位) 加到 AR2。
```

实例 2

```
STL             解释
+AR1  P#300.0   //将偏移量 30.0 加到 AR2。
```

14.13 BLD 程序显示指令 (空)

格式

BLD <数字>

地址	描述
<数字>	数字指定 BLD 指令，范围从 0 至 255

描述

BLD <数字> (程序显示指令；空指令) 不执行任何功能，并且不影响状态位。该指令用于编程设备 (PG) 的图形显示。在 STL 中显示梯形图或 FBD 程序时将自动创建它。地址 <数字> 指定 BLD 指令并由编程设备生成。

状态字

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

14.14 NOP 0 空指令

格式

NOP 0

描述

NOP 0 (地址为“0”的指令 NOP) 不执行任何功能，并且不影响状态位。该指令代码包含具有 16 个零的位模式。当显示某个程序时，该指令仅对编程设备 (PG) 有用。

状态字

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

14.15 NOP 1 空指令

格式

NOP 1

描述

NOP 1 (地址为“1”的指令 NOP) 不执行任何功能，并且不影响状态位。该指令代码包含具有 16 个“1”的位模式。当显示某个程序时，该指令仅对编程设备 (PG) 有用。

状态字

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

A 所有 STL 指令概述

A.1 按德语助记符排序的 STL 指令 (SIMATIC)

德语助记符	英语助记符	程序单元目录	描述
+	+	整型数学运算指令	加整型常数 (16、32 位)
=	=	位逻辑指令	赋值
)	)	位逻辑指令	嵌套结束
+AR1	+AR1	累加器	AR1 将 ACCU 1 加到地址寄存器 1
+AR2	+AR2	累加器	AR2 将 ACCU 1 加到地址寄存器 2
+D	+D	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为长整型 (32 位) 相加
-D	-D	整型数学运算指令	以长整型 (32 位) 的形式从 ACCU 2 中减去 ACCU 1
*D	*D	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为长整型 (32 位) 相乘
/D	/D	整型数学运算指令	以长整型 (32 位) 的形式用 ACCU 1 除 ACCU 2
? D	? D	比较	比较长整型数 (32 位) ==、<>、>、<、>=、<=
+I	+I	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为整型 (16 位) 相加
–I	–I	整型数学运算指令	以整型 (16 位) 的形式从 ACCU 2 中减去 ACCU 1
*I	*I	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为整型 (16 位) 相乘
/I	/I	整型数学运算指令	以整型数 (16 位) 的形式用 ACCU 1 除 ACCU 2
? I	? I	比较	比较整型数 (16 位) ==、<>、>、<、>=、<=
+R	+R	浮点型指令	将 ACCU 1 和 ACCU 2 作为浮点数 (32 位 IEEE 754) 相加
–R	–R	浮点型指令	以浮点数 (32 位 IEEE 754) 的形式从 ACCU 2 中减去 ACCU 1
*R	*R	浮点型指令	将 ACCU 1 和 ACCU 2 作为浮点数 (32 位 IEEE 754) 相乘
/R	/R	浮点型指令	以浮点数 (32 位 IEEE 754) 的形式用 ACCU 1 除 ACCU 2
? R	? R	比较	比较浮点数 (32 位) ==、<>、>、<、>=、<=
ABS	ABS	浮点型指令	浮点数 (32 位 IEEE 754) 的绝对值
ACOS	ACOS	浮点型指令	生成浮点数 (32 位) 的反余弦
ASIN	ASIN	浮点型指令	生成浮点数 (32 位) 的反正弦
ATAN	ATAN	浮点型指令	生成浮点数 (32 位) 的反正切
AUF	OPN	DB 调用	打开数据块
BE	BE	程序控制	块结束
BEA	BEU	程序控制	无条件的块结束
BEB	BEC	程序控制	有条件的块结束
BLD	BLD	程序控制	程序显示指令 (空)
BTD	BTD	转换	BCD 码转换为整型 (32 位)
BTI	BTI	转换	BCD 码转换为整型 (16 位)

德语助记符	英语助记符	程序单元目录	描述
CALL	CALL	程序控制	块调用
CALL	CALL	程序控制	调用多重情景
CALL	CALL	程序控制	调用来自库的块
CC	CC	程序控制	条件调用
CLR	CLR	位逻辑指令	清除 RLO (=0)
COS	COS	浮点型指令	以浮点数 (32 位) 形式生成角的余弦
DEC	DEC	累加器	减量 ACCU 1-L-L
DTB	DTB	转换	长整型 (32 位) 转换为 BCD 码
DTR	DTR	转换	将长整型 (32 位) 转换为浮点数 (32 位，IEEE 754)
ENT	ENT	累加器	进入 ACCU 堆栈
EXP	EXP	浮点型指令	生成浮点数 (32 位) 的指数值
FN	FN	位逻辑指令	下降沿
FP	FP	位逻辑指令	上升沿
FR	FR	计数器	启用计数器 (空闲) (空闲，FR C 0 到 C 255)
FR	FR	定时器	启用定时器 (空闲)
INC	INC	累加器	增量 ACCU 1-L-L
INVD	INVD	转换	长整数按位取反 (32 位)
INVI	INVI	转换	整数按位取反 (16 位)
ITB	ITB	转换	整型 (16 位) 转换为 BCD 码
ITD	ITD	转换	整型 (16 位) 转换为长整型 (32 位)
L	L	装载/传送	装载
L DBLG	L DBLG	装载/传送	在 ACCU 1 中装载共享 DB 的长度
L DBNO	L DBNO	装载/传送	在 ACCU 1 中装载共享 DB 的编号
L DILG	L DILG	装载/传送	在 ACCU 1 中装载实例 DB 的长度
L DINO	L DINO	装载/传送	在 ACCU 1 中装载实例 DB 的编号
L STW	L STW	装载/传送	将状态字装载到 ACCU 1 中
L	L	装载/传送	以整数的形式将当前定时器值装载到 ACCU 1 中 (当前定时器值可以是一个位于 0-255 之间的数值，例如，L T 32)
L	L	装载/传送	将当前计数器值装载到 ACCU 1 (当前计数器值可以是一个位于 0-255 之间的数值，例如，L C 15)
LAR1	LAR1	装载/传送	从 ACCU 1 中装载地址寄存器
LAR1	LAR1	装载/传送	用长整型 (32 位指针) 装载地址寄存器 1
LAR1	LAR1	装载/传送	从地址寄存器 2 装载地址寄存器 1
LAR2	LAR2	装载/传送	从 ACCU 1 中装载地址寄存器 2
LAR2	LAR2	装载/传送	用长整型 (32 位指针) 装载地址寄存器 2
LC	LC	计数器	以 BCD 码的形式将当前计数器值装载到 ACCU 1 中 (当前定时器值可以是一个位于 0-255 之间的数值，例如 LC C 15)
LC	LC	定时器	以 BCD 码的形式将当前定时器值装载到 ACCU 1 (当前计数器可以是一个位于 0-255 之间的数值，例如，LC T 32)
LEAVE	LEAVE	累加器	离开 ACCU 堆栈

德语助记符	英语助记符	程序单元目录	描述
LN	LN	浮点型指令	生成浮点数 (32 位) 的自然对数
LOOP	LOOP	跳转	循环
MCR(	MCR(	程序控制	将 RLO 保存在 MCR 堆栈中，开始 MCR
)MCR	)MCR	程序控制	结束 MCR
MCRA	MCRA	程序控制	激活 MCR 区域
MCRD	MCRD	程序控制	取消激活 MCR 区域
MOD	MOD	整型数学运算指令	除法余数为长整型 (32 位)
NEGD	NEGD	转换	长整数取负 (32 位)
NEGI	NEGI	转换	对整数 (16 位) 求补码
NEGR	NEGR	转换	浮点数 (32 位，IEEE 754) 取反
NOP 0	NOP 0	累加器	空指令
NOP 1	NOP 1	累加器	空指令
NOT	NOT	位逻辑指令	取反 RLO
O	O	位逻辑指令	或
O(	O(	位逻辑指令	或运算嵌套开始
OD	OD	字逻辑指令	双字或运算 (32 位)
ON	ON	位逻辑指令	或非运算
ON(	ON(	位逻辑指令	或非运算嵌套开始
OW	OW	字逻辑指令	字或运算 (16 位)
POP	POP	累加器	具有两个 ACCU 的 CPU
POP	POP	累加器	具有四个 ACCU 的 CPU
PUSH	PUSH	累加器	具有两个 ACCU 的 CPU
PUSH	PUSH	累加器	具有四个 ACCU 的 CPU
R	R	位逻辑指令	复位
R	R	计数器	复位计数器 (当前计数器可以是一个位于 0-255 之间的数值，例如，R C 15)
R	R	定时器	复位定时器 (当前定时器可以是一个位于 0-255 之间的数值，例如，R T 32)
RLD	RLD	移位/循环	双字循环左移 (32 位)
RLDA	RLDA	移位/循环	通过 CC 1 (32 位) 左循环 ACCU 1
RND	RND	转换	取整
RND+	RND+	转换	向上取整长整型
RND–	RND–	转换	向下取整长整型
RRD	RRD	移位/循环	双字循环右移 (32 位)
RRDA	RRDA	移位/循环	经过 CC 1 (32 位) 右循环 ACCU 1
S	S	位逻辑指令	置位
S	S	计数器	置位计数器预设定值 (当前计数器可以是一个位于 0-255 之间的数值，例如，S C 15)
SA	SF	定时器	断开延时定时器
SAVE	SAVE	位逻辑指令	将 RLO 保存在 BR 寄存器中
SE	SD	定时器	接通延时定时器
SET	SET	位逻辑指令	置位
SI	SP	定时器	脉冲定时器

德语助记符	英语助记符	程序单元目录	描述
SIN	SIN	浮点型指令	以浮点数 (32 位) 形式生成角的正弦
SLD	SLD	移位/循环	双字左移 (32 位)
SLW	SLW	移位/循环	字左移 (16 位)
SPA	JU	跳转	无条件跳转
SPB	JC	跳转	如果 RLO = 1，则跳转
SPBB	JCB	跳转	如果具有 BR 的 RLO = 1，则跳转
SPBI	JBI	跳转	如果 BR = 1，则跳转
SPBIN	JNBI	跳转	如果 BR = 0，则跳转
SPBN	JCN	跳转	如果 RLO = 0，则跳转
SPBNB	JNB	跳转	如果具有 BR 的 RLO = 0，则跳转
SPL	JL	跳转	跳转到标签
SPM	JM	跳转	如果为负，则跳转
SPMZ	JMZ	跳转	如果为负或零，则跳转
SPN	JN	跳转	如果非零，则跳转
SPO	JO	跳转	如果 OV = 1，则跳转
SPP	JP	跳转	如果为正，则跳转
SPPZ	JPZ	跳转	如果为正或零，则跳转
SPS	JOS	跳转	如果 OS = 1，则跳转
SPU	JUO	跳转	如果无序，则跳转
SPZ	JZ	跳转	如果为零，则跳转
SQR	SQR	浮点型指令	生成浮点数 (32 位) 的平方
SQRT	SQRT	浮点型指令	生成浮点数 (32 位) 的平方根
SRD	SRD	移位/循环	双字右移 (32 位)
SRW	SRW	移位/循环	单字右移 (16 位)
SS	SS	定时器	带保持的接通延时定时器
SSD	SSD	移位/循环	移位有符号长整数 (32 位)
SSI	SSI	移位/循环	移位有符号整型数 (16 位)
SV	SE	定时器	扩展的脉冲定时器
T	T	装载/传送	传送
T STW	T STW	装载/传送	将 ACCU 1 传送到状态字
TAD	CAD	转换	改变 ACCU 1 (32 位) 中的字节顺序
TAK	TAK	累加器	将 ACCU 1 与 ACCU 2 进行交换
TAN	TAN	浮点型指令	以浮点数 (32 位) 形式生成角的正切
TAR	CAR	装载/传送	将地址寄存器 1 与地址寄存器 2 进行交换
TAR1	TAR1	装载/传送	将地址寄存器 1 传送到 ACCU 1
TAR1	TAR1	装载/传送	将地址寄存器 1 传送到目标地址 (32 位指针)
TAR1	TAR1	装载/传送	将地址寄存器 1 传送到地址寄存器 2
TAR2	TAR2	装载/传送	将地址寄存器 2 传送到 ACCU 1
TAR2	TAR2	装载/传送	将地址寄存器 2 传送到目标地址 (32 位指针)
TAW	CAW	转换	改变 ACCU 1-L (16 位) 中的字节顺序
TDB	CDB	转换	交换共享 DB 和背景数据块
TRUNC	TRUNC	转换	截尾
U	A	位逻辑指令	与运算
U(	A(	位逻辑指令	与运算嵌套开始

德语助记符	英语助记符	程序单元目录	描述
UC	UC	程序控制	无条件调用
UD	AD	字逻辑指令	双字与运算 (32 位)
UN	AN	位逻辑指令	与非运算
UN(	AN(	位逻辑指令	与非运算嵌套开始
UW	AW	字逻辑指令	单字与运算 (16 位)
X	X	位逻辑指令	异或运算
X(	X(	位逻辑指令	异或运算嵌套开始
XN	XN	位逻辑指令	同或运算
XN(	XN(	位逻辑指令	同或运算嵌套开始
XOD	XOD	字逻辑指令	双字异或运算 (32 位)
XOW	XOW	字逻辑指令	单字异或运算 (16 位)
ZR	CD	计数器	向下计数器
ZV	CU	计数器	向上计数器

A.2 按英语助记符 (国际) 排序的 STL 指令

英语助记符	德语助记符	程序单元目录	描述
+	+	整型数学运算指令	加整型常数 (16、32 位)
=	=	位逻辑指令	赋值
)	)	位逻辑指令	嵌套结束
+AR1	+AR1	累加器	AR1 将 ACCU 1 加到地址寄存器 1
+AR2	+AR2	累加器	AR2 将 ACCU 1 加到地址寄存器 2
+D	+D	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为长整型 (32 位) 相加
-D	-D	整型数学运算指令	以长整型 (32 位) 的形式从 ACCU 2 中减去 ACCU 1
*D	*D	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为长整型 (32 位) 相乘
/D	/D	整型数学运算指令	以长整型 (32 位) 的形式用 ACCU 1 除 ACCU 2
? D	? D	比较	比较长整型数 (32 位) ==、<>、>、<、>=、<=
+I	+I	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为整型 (16 位) 相加
–I	–I	整型数学运算指令	以整型 (16 位) 的形式从 ACCU 2 中减去 ACCU 1
*I	*I	整型数学运算指令	将 ACCU 1 和 ACCU 2 作为整型 (16 位) 相乘
/I	/I	整型数学运算指令	以整型数 (16 位) 的形式用 ACCU 1 除 ACCU 2
? I	? I	比较	比较整型数 (16 位) ==、<>、>、<、>=、<=
+R	+R	浮点型指令	将 ACCU 1 和 ACCU 2 作为浮点数 (32 位 IEEE 754) 相加
–R	–R	浮点型指令	以浮点数 (32 位 IEEE 754) 的形式从 ACCU 2 中减去 ACCU 1
*R	*R	浮点型指令	将 ACCU 1 和 ACCU 2 作为浮点数 (32 位 IEEE 754) 相乘
/R	/R	浮点型指令	以浮点数 (32 位 IEEE 754) 的形式用 ACCU 1 除 ACCU 2
? R	? R	比较	比较浮点数 (32 位) ==、<>、>、<、>=、<=
A	U	位逻辑指令	与运算
A(	U(	位逻辑指令	与运算嵌套开始
ABS	ABS	浮点型指令	浮点数 (32 位 IEEE 754) 的绝对值
ACOS	ACOS	浮点型指令	生成浮点数 (32 位) 的反余弦
AD	UD	字逻辑指令	双字与运算 (32 位)
AN	UN	位逻辑指令	与非运算
AN(	UN(	位逻辑指令	与非运算嵌套开始
ASIN	ASIN	浮点型指令	生成浮点数 (32 位) 的反正弦
ATAN	ATAN	浮点型指令	生成浮点数 (32 位) 的反正切

英语助记符	德语助记符	程序单元目录	描述
AW	UW	字逻辑指令	单字与运算 (16 位)
BE	BE	程序控制	块结束
BEC	BEB	程序控制	有条件的块结束
BEU	BEA	程序控制	无条件的块结束
BLD	BLD	程序控制	程序显示指令 (空)
BTD	BTD	转换	BCD 码转换为整型 (32 位)
BTI	BTI	转换	BCD 码转换为整型 (16 位)
CAD	TAD	转换	改变 ACCU 1 (32 位) 中的字节顺序
CALL	CALL	程序控制	块调用
CALL	CALL	程序控制	调用多重情景
CALL	CALL	程序控制	调用来自库的块
CAR	TAR	装载/传送	将地址寄存器 1 与地址寄存器 2 进行交换
CAW	TAW	转换	改变 ACCU 1-L (16 位) 中的字节顺序
CC	CC	程序控制	条件调用
CD	ZR	计数器	向下计数器
CDB	TDB	转换	交换共享 DB 和背景数据块
CLR	CLR	位逻辑指令	清除 RLO (=0)
COS	COS	浮点型指令	以浮点数 (32 位) 形式生成角的余弦
CU	ZV	计数器	向上计数器
DEC	DEC	累加器	减量 ACCU 1-L-L
DTB	DTB	转换	长整型 (32 位) 转换为 BCD 码
DTR	DTR	转换	将长整型 (32 位) 转换为浮点数 (32 位，IEEE 754)
ENT	ENT	累加器	进入 ACCU 堆栈
EXP	EXP	浮点型指令	生成浮点数 (32 位) 的指数值
FN	FN	位逻辑指令	下降沿
FP	FP	位逻辑指令	上升沿
FR	FR	计数器	启用计数器 (空闲) (空闲，FR C 0 到 C 255)
FR	FR	定时器	启用定时器 (空闲)
INC	INC	累加器	增量 ACCU 1-L-L
INVD	INVD	转换	长整数按位取反 (32 位)
INVI	INVI	转换	整数按位取反 (16 位)
ITB	ITB	转换	整型 (16 位) 转换为 BCD 码
ITD	ITD	转换	整型 (16 位) 转换为长整型 (32 位)
JBI	SPBI	跳转	如果 BR = 1，则跳转
JC	SPB	跳转	如果 RLO = 1，则跳转
JCB	SPBB	跳转	如果具有 BR 的 RLO = 1，则跳转
JCN	SPBN	跳转	如果 RLO = 0，则跳转
JL	SPL	跳转	跳转到标签
JM	SPM	跳转	如果为负，则跳转
JMZ	SPMZ	跳转	如果为负或零，则跳转
JN	SPN	跳转	如果非零，则跳转
JNB	SPBNB	跳转	如果具有 BR 的 RLO = 0，则跳转
JNBI	SPBIN	跳转	如果 BR = 0，则跳转
JO	SPO	跳转	如果 OV = 1，则跳转
JOS	SPS	跳转	如果 OS = 1，则跳转
JP	SPP	跳转	如果为正，则跳转
JPZ	SPPZ	跳转	如果为正或零，则跳转

英语助记符	德语助记符	程序单元目录	描述
JU	SPA	跳转	无条件跳转
JUO	SPU	跳转	如果无序，则跳转
JZ	SPZ	跳转	如果为零，则跳转
L	L	装载/传送	装载
L DBLG	L DBLG	装载/传送	在 ACCU 1 中装载共享 DB 的长度
L DBNO	L DBNO	装载/传送	在 ACCU 1 中装载共享 DB 的编号
L DILG	L DILG	装载/传送	在 ACCU 1 中装载实例 DB 的长度
L DINO	L DINO	装载/传送	在 ACCU 1 中装载实例 DB 的编号
L STW	L STW	装载/传送	将状态字装载到 ACCU 1 中
L	L	定时器	以整数的形式将当前定时器值装载到 ACCU 1 中 (当前定时器值可以是一个位于 0-255 之间的数值，例如，L T 32)
L	L	计数器	将当前计数器值装载到 ACCU 1 (当前计数器值可以是一个位于 0-255 之间的数值，例如，L C 15)
LAR1	LAR1	装载/传送	从 ACCU 1 中装载地址寄存器
LAR1<D>	LAR1<D>	装载/传送	用长整型 (32 位指针) 装载地址寄存器 1
LAR1 AR2	LAR1 AR2	装载/传送	从地址寄存器 2 装载地址寄存器 1
LAR2	LAR2	装载/传送	从 ACCU 1 中装载地址寄存器 2
LAR2 <D>	LAR2 <D>	装载/传送	用长整型 (32 位指针) 装载地址寄存器 2
LC	LC	计数器	以 BCD 码的形式将当前计数器值装载到 ACCU 1 中 (当前定时器值可以是一个位于 0-255 之间的数值，例如 LC C 15)
LC	LC	定时器	以 BCD 码的形式将当前定时器值装载到 ACCU 1 (当前计数器可以是一个位于 0-255 之间的数值，例如，LC T 32)
LEAVE	LEAVE	累加器	离开 ACCU 堆栈
LN	LN	浮点型指令	生成浮点数 (32 位) 的自然对数
LOOP	LOOP	跳转	循环
MCR(	MCR(	程序控制	将 RLO 保存在 MCR 堆栈中，开始 MCR
)MCR	)MCR	程序控制	结束 MCR
MCRA	MCRA	程序控制	激活 MCR 区域
MCRD	MCRD	程序控制	取消激活 MCR 区域
MOD	MOD	整型数学运算指令	除法余数为长整型 (32 位)
NEGD	NEGD	转换	长整数取负 (32 位)
NEGI	NEGI	转换	对整数 (16 位) 求补码
NEGR	NEGR	转换	浮点数 (32 位，IEEE 754) 取反
NOP 0	NOP 0	累加器	空指令
NOP 1	NOP 1	累加器	空指令
NOT	NOT	位逻辑指令	取反 RLO
O	O	位逻辑指令	或
O(	O(	位逻辑指令	或运算嵌套开始
OD	OD	字逻辑指令	双字或运算 (32 位)
ON	ON	位逻辑指令	或非运算
ON(	ON(	位逻辑指令	或非运算嵌套开始
OPN	AUF	DB 调用	打开数据块
OW	OW	字逻辑指令	字或运算 (16 位)
POP	POP	累加器	具有两个 ACCU 的 CPU

英语助记符	德语助记符	程序单元目录	描述
POP	POP	累加器	具有四个 ACCU 的 CPU
PUSH	PUSH	累加器	具有两个 ACCU 的 CPU
PUSH	PUSH	累加器	具有四个 ACCU 的 CPU
R	R	位逻辑指令	复位
R	R	计数器	复位计数器 (当前计数器可以是一个位于 0-255 之间的数值，例如，R C 15)
R	R	定时器	复位定时器 (当前定时器可以是一个位于 0-255 之间的数值，例如，R T 32)
RLD	RLD	移位/循环	双字循环左移 (32 位)
RLDA	RLDA	移位/循环	通过 CC 1 (32 位) 左循环 ACCU 1
RND	RND	转换	取整
RND -	RND -	转换	向下取整长整型
RND+	RND+	转换	向上取整长整型
RRD	RRD	移位/循环	双字循环右移 (32 位)
RRDA	RRDA	移位/循环	经过 CC 1 (32 位) 右循环 ACCU 1
S	S	位逻辑指令	置位
S	S	计数器	置位计数器预设定值 (当前计数器可以是一个位于 0-255 之间的数值，例如，S C 15)
SAVE	SAVE	位逻辑指令	将 RLO 保存在 BR 寄存器中
SD	SE	定时器	接通延时定时器
SE	SV	定时器	扩展的脉冲定时器
SET	SET	位逻辑指令	置位
SF	SA	定时器	断开延时定时器
SIN	SIN	浮点型指令	以浮点数 (32 位) 形式生成角的正弦
SLD	SLD	移位/循环	双字左移 (32 位)
SLW	SLW	移位/循环	字左移 (16 位)
SP	SI	定时器	脉冲定时器
SQR	SQR	浮点型指令	生成浮点数 (32 位) 的平方
SQRT	SQRT	浮点型指令	生成浮点数 (32 位) 的平方根
SRD	SRD	移位/循环	双字右移 (32 位)
SRW	SRW	移位/循环	单字右移 (16 位)
SS	SS	定时器	带保持的接通延时定时器
SSD	SSD	移位/循环	移位有符号长整数 (32 位)
SSI	SSI	移位/循环	移位有符号整型数 (16 位)
T	T	装载/传送	传送
T STW	T STW	装载/传送	将 ACCU 1 传送到状态字
TAK	TAK	累加器	将 ACCU 1 与 ACCU 2 进行交换
TAN	TAN	浮点型指令	以浮点数 (32 位) 形式生成角的正切
TAR1	TAR1	装载/传送	将地址寄存器 1 传送到 ACCU 1
TAR1	TAR1	装载/传送	将地址寄存器 1 传送到目标地址 (32 位指针)
TAR1	TAR1	装载/传送	将地址寄存器 1 传送到地址寄存器 2
TAR2	TAR2	装载/传送	将地址寄存器 2 传送到 ACCU 1
TAR2	TAR2	装载/传送	将地址寄存器 2 传送到目标地址 (32 位指针)
TRUNC	TRUNC	转换	截尾
UC	UC	程序控制	无条件调用

英语助记符	德语助记符	程序单元目录	描述
X	X	位逻辑指令	异或运算
X(	X(	位逻辑指令	异或运算嵌套开始
XN	XN	位逻辑指令	同或运算
XN(	XN(	位逻辑指令	同或运算嵌套开始
XOD	XOD	字逻辑指令	双字异或运算 (32 位)
XOW	XOW	字逻辑指令	单字异或运算 (16 位)

B 编程实例

B.1 编程实例总览

实际应用

每个语句表指令触发一项指定操作。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。本章提供了语句表指令实际应用的下列实例：

- 使用位逻辑指令控制传送带
- 使用位逻辑指令检测传送带上的移动方向
- 使用定时器指令生成一个时钟脉冲
- 使用计数器和比较指令跟踪存储空间
- 使用整数运算指令解决问题
- 设置加热烘炉的时间长度

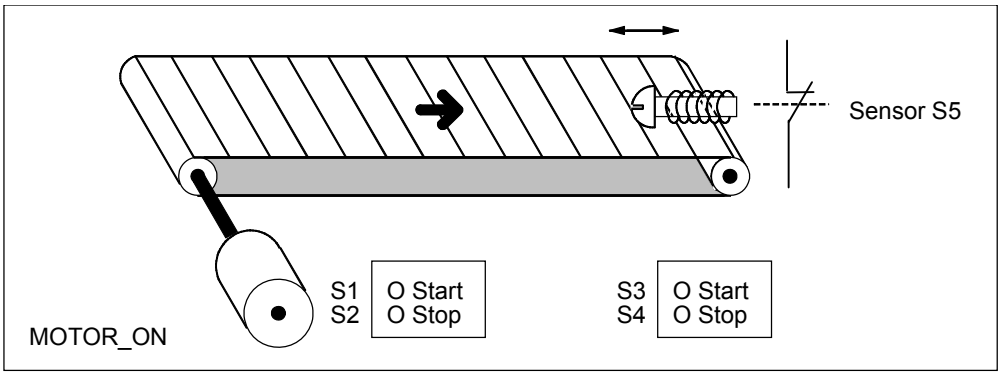
使用的指令

助记符	程序单元目录	描述
AW	字逻辑指令	单字与运算
OW	字逻辑指令	单字或运算
CD，CU	计数器	向下计数，向上计数
S，R	位逻辑指令	置位，复位
NOT	位逻辑指令	取反 RLO
FP	位逻辑指令	上升沿
+I	浮点指令	将累加器 1 和 2 作为整数相加
/I	浮点指令	以整数形式用累加器 1 除累加器 2
*I	浮点指令	将累加器 1 和 2 作为整数相乘
>=I，<=I	比较	比较整数
A，AN	位逻辑指令	与运算，与非运算
O，ON	位逻辑指令	或运算，或非运算
=	位逻辑指令	赋值
INC	累加器	累加器加 1
BE，BEC	程序控制	块结束和块条件结束
L，T	装载/传送	装载和传送
SE	定时器	扩展的脉冲定时器

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
Motor	Q 4.0	MOTOR_ON	Q 4.0 MOTOR_ON

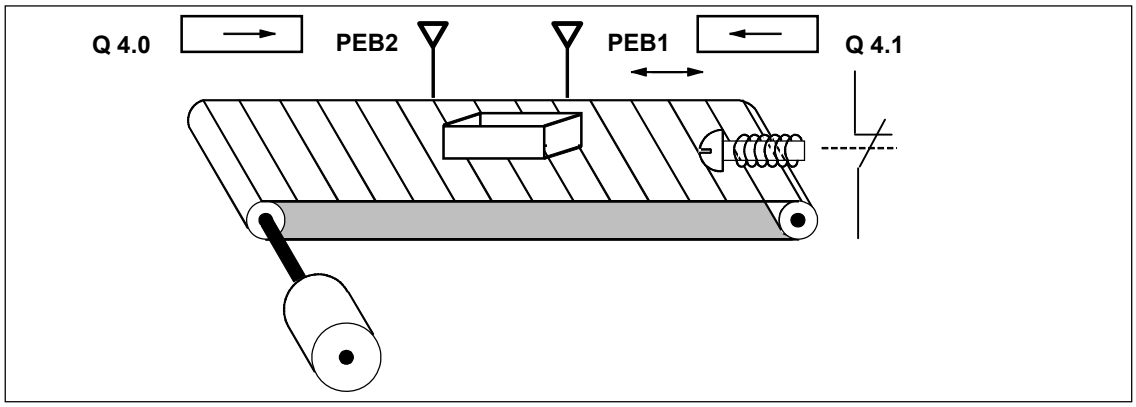
绝对地址编写的程序	符号编写的程序
O I 1.1	O S1
O I 1.3	O S3
S Q 4.0	S MOTOR_ON
O I 1.2	O S2
O I 1.4	O S4
ON I 1.5	ON S5
R Q 4.0	R MOTOR_ON

控制传送带的语句表

```
STL                 解释
O     I 1.1         //按任一启动开关可接通电机。
O     I 1.3
S     Q 4.0
O     I 1.2         //按任一停止开关或打开传送带末端的常闭触点均可关闭电机。
O     I 1.4
ON    I 1.5
R     Q 4.0
```

实例 2：检测传送带方向

下图显示配备有两个光电屏障 (PEB1 和 PEB2) 的传送带，这两个光电屏障用于检测包裹在传送带上的移动方向。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

绝对地址编写的程序		符号编写的程序	
A	I 0.0	A	PEB1
FP	M 0.0	FP	PMB1
AN	I 0.1	AN	PEB 2
S	Q 4.1	S	LEFT
A	I 0.1	A	PEB 2
FP	M 0.1	FP	PMB 2
AN	I 0.0	AN	PEB 1
S	Q 4.0	S	RIGHT
AN	I 0.0	AN	PEB 1
AN	I 0.1	AN	PEB 2
R	Q 4.0	R	RIGHT
R	Q 4.1	R	LEFT

语句表

```
STL                 解释
A     I 0.0   //如果输入 I 0.0 有从 0 到 1 (上升沿) 的转变，同时输入 I 0.1 处的信号状态为 0，
              //则传送带上的包裹会向左移动。
FP    M 0.0
AN    I 0.1
S     Q 4.1
A     I 0.1   //如果输入 I 0.1 处有从 0 到 1 (上升沿) 的转变，同时输入 I 0.0 处的信号状态为 0，则传送带上的包
              //裹会向右移动。如果其中一个光电屏障断开，
              //则说明光电屏障之间有包裹。
FP    M 0.1
AN    I 0.0
S     Q 4.0
AN    I 0.0   //如果两个光电屏障都未断开，则说明光电屏障之间无包裹。方向指针关闭。
AN    I 0.1
R     Q 4.0
R     Q 4.1
```

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。

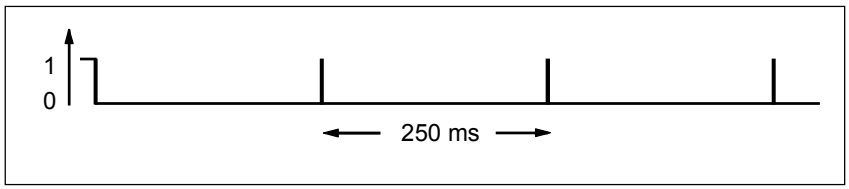
当使用 S7-300 时，您可用特殊组织块中的时间处理功能来执行时钟脉冲发生器功能。而下列语句表中显示的实例说明了使用定时器函数生成时钟脉冲的相关内容。实例程序显示如何通过使用定时器实现任意的时钟脉冲发生器。

生成时钟脉冲的语句表 (脉冲占空比 1:1)

STL	解释
AN T1	//如果定时器 T 1 的定时时间到，
L S5T#250ms	//将时间值 250 毫秒装载到 T 1 并
SV T1	//启动 T 1 作为扩展脉冲定时器。
NOT	//对逻辑运算结果求反 (取反)。
BEB	//如果定时器正在运行，请结束当前块。
L MB100	//如果定时器的定时时间到，请装载存储器字节 MB100 的内容，
INC 1	//并加 1，
T MB100	//然后将结果传送到存储器字节 MB100。

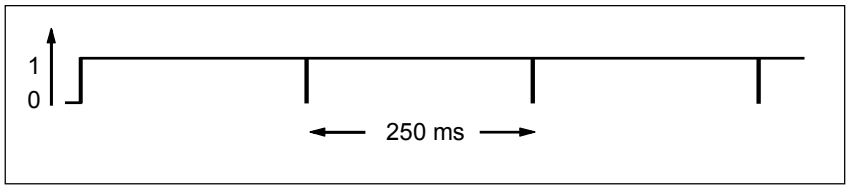
信号检测

检测定时器 T1 的信号，逻辑运算结果 (RLO) 如下。



一旦定时时间到，就会重新启动定时器。因此，信号检测使语句 **AN T1** 只会瞬间产生信号状态 1。

RLO 取反 (反向)：



每隔 250 毫秒 RLO 位为 0。因而 BEC 语句不会结束块处理。相反，会将存储器字节 MB100 的内容加 1。

存储器字节 MB100 的内容每 250 毫秒会以如下方式发生改变：

0 -> 1 -> 2 -> 3 -> ... -> 254 -> 255 -> 0 -> 1 ...

获得指定频率

可从存储器字节 MB100 的各个位中获得下列频率：

MB100 的位	**频率 (单位：赫兹)**	**持续时间**
M 100.0	2.0	0.5s (250 毫秒开/ 250 毫秒关)
M 100.1	1.0	1s (0.5 秒开/ 0.5 秒关)
M 100.2	0.5	2s (1 秒开/ 1 秒关)
M 100.3	0.25	4s (2 秒开/ 2 秒关)
M 100.4	0.125	8s (4 秒开/ 4 秒关)
M 100.5	0.0625	16s (8 秒开/ 8 秒关)
M 100.6	0.03125	32s (16 秒开/ 16 秒关)
M 100.7	0.015625	64s (32 秒开/ 32 秒关)

语句表

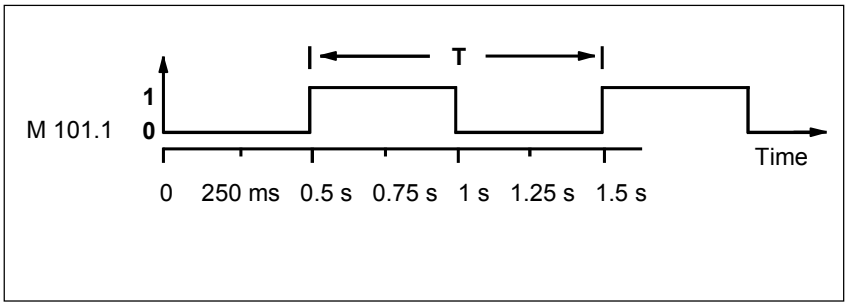
```
STL                 解释
A     M10.0         //当发生错误时，M 10.0 = 1。当发生错误时，错误指示灯以 1 赫兹的频率闪烁。
A     M100.1
=     Q 4.0
```

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	**0**	0	250
1	0	0	0	0	0	0	**0**	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	**0**	0	250
5	0	0	0	0	0	1	**0**	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	**0**	0	250
9	0	0	0	0	1	0	**0**	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	**0**	0	250

MB 101 (M 101.1) 第 1 位的信号状态

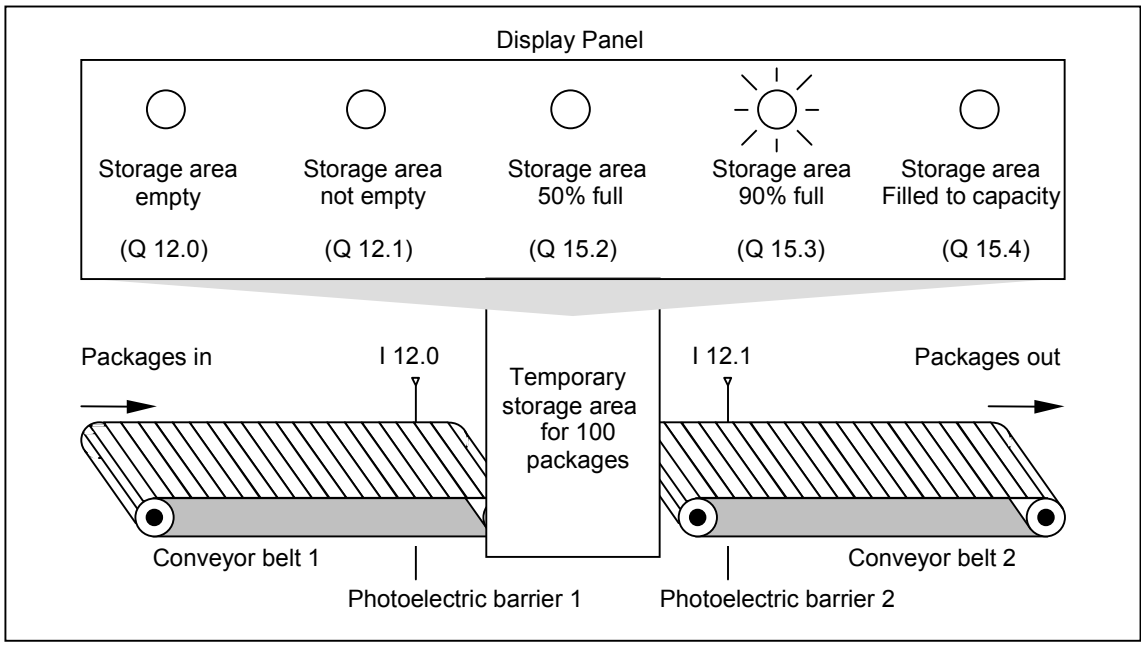
频率 = 1/T = 1/1 s = 1 赫兹



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。存储区域附近的传送带 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。



激活显示面板上指示灯的语句表

```
STL             解释
A       I 0.0   //由光电屏障 1 生成的各个脉冲
CU      C1      //将计数器 C 1 的计数值增加 1，借此计算进入存储区域的包裹数量。
                //
A       I 0.1   //由光电屏障 2 生成的各个脉冲
CD      C1      //将计数器 C 1 的计数值减少 1，借此计算离开存储区域的包裹数量。
                //
AN      C1      //如果计数值为 0，
=       Q 4.0   //则“存储区域已空”的指示灯亮起。
                //
A       C1      //如果计数值不为零，
=       A 4.1   //则“存储区域非空”的指示灯亮起。
                //
L       50
L       C1
<=I             //如果计数值大余等于 50，
=       Q 4.2   //则“50%存储区域已满”的指示灯亮起。
                //
L       90
>=I             //如果计数值大于等于 90，
=       Q 4.3   //则“90%存储区域已满”的指示灯亮起。
                //
L       Z1
L       100
>=I             //如果计数值大于等于 100，
=       Q 4.4   //则“存储区域已填满”的指示灯亮起。
                //(您也可使用输出 Q 4.4 来锁定传送带 1。)
```

B.5 实例：整数运算指令

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：

MD4 = ((IW0 + DBW3) x 15) / MW2

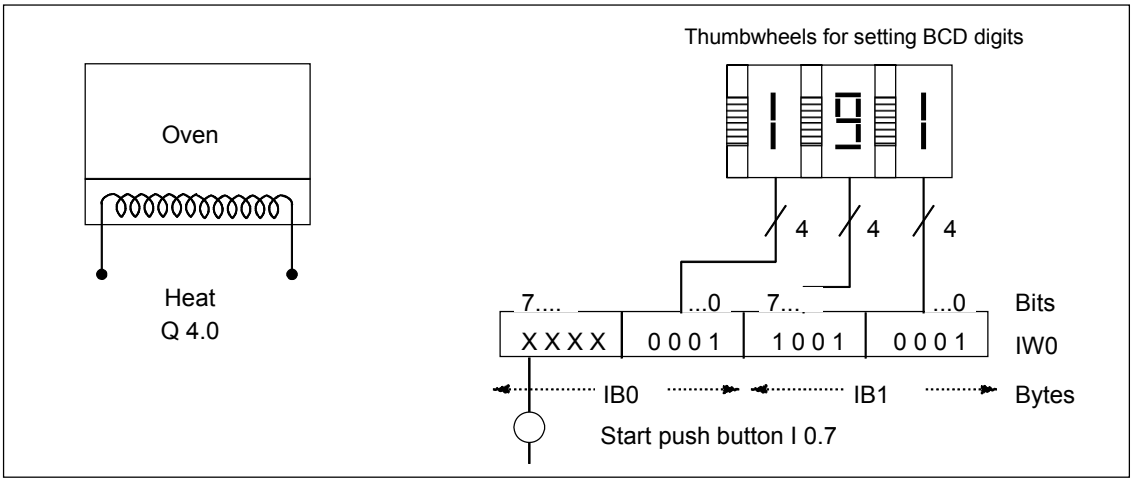
语句表

```
STL                 解释
L       EW0         //将输入字 IWO 的值装载到累加器 1。
L       DB5.DBW3    //将 DB5 的共享数据字 DBW3 中的值装载到累加器 1。
                    //累加器 1 的旧内容被传送到累加器 2 中。
+I      I 0.1       //将累加器 1 和 2 的低位字内容相加。
                    //其结果存储在累加器 1 的低位字中。
                    //累加器 2 的内容和累加器 1 的高位字保持不变。
L       +15         //将常数值 +15 装载到累加器 1。
                    //累加器 1 的旧内容被传送到累加器 2 中。
*I                  //用累加器 1 低位字的内容乘以累加器 2 低位字的内容。
                    //结果存储在累加器 1 中。
                    //累加器 2 的内容保持不变。
L       MW2         //将存储器字 MW2 的值装载到累加器 1。
                    //累加器 1 的旧内容被传送到累加器 2 中。
/I                  //用累加器 1 低位字的内容除累加器 2 低位字的内容。
                    //结果存储在累加器 1 中。
                    //累加器 2 的内容保持不变。
T       MD4         //将最终结果传送到存储器双字 MD4。
                    //两个累加器的内容都保持不变。
```

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
十位码盘	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

语句表

```
STL                 解释
A     T1            //如果定时器正在运行，
=     Q 4.0         //则打开加热装置。
BEC                 //如果定时器正在运行，则在此结束进程。
                    //这可防止在按下按钮时重新启动定时器 T1。
L     IW0
AW    W#16#0FFF     //屏蔽输入位 I 0.4 到 I 0.7 (即，将它们复位到 0)。
                    //以秒为单位的时间值位于累加器 1 的低位字中，并以二进制编码的十进制格式 (BCD 码) 显示。
OW    W#16#2000     //在累加器 1 低位字中的第 12 位和 13 位中将时间基准指定为秒。
A     I 0.7
SE    T1            //如果按下按钮，则启动定时器 T1 作为扩展脉冲定时器。
```

C 参数传送

块参数以值的形式传送。对于功能块，在被调用块中使用背景数据块中的实际参数值的副本。对于功能，实际值的副本存在于本地数据堆栈中。将不复制指针。在调用之前，将 INPUT 值复制到背景数据块或 L 堆栈中。调用以后，将 OUTPUT 值往回复制给变量。在被调用的块中，仅可使用副本。所需的 STL 指令位于调用块中，并且对于用户来说是不可见。

注意

如果将存储位、输入、输出或外设 I/O 存储区用作功能的实际地址，则对它们的处理方法将不同于对其它地址的处理方法。此处，直接执行更新，不通过 L 堆栈。

当心

编写被调用块时，请确保同时还编写了声明为 OUTPUT 的参数。否则，将输出随机值！ 对于功能块，此值将是上一次调用记录的来自背景数据块的值；对于功能，此值将恰巧是 L 堆栈中的值。

请注意以下几点：

- 尽可能初始化所有OUTPUT参数。
- 尽量不要使用置位和复位指令。这些指令与RLO相关。如果RLO具有0值，则将保留随机值。
- 如果在块内跳转，请确保不要跳过任何编写了OUTPUT参数的位置。请勿忘记BEC和MCR指令的作用。

索引

字母

A

B

C

D

F

G

H

J

K

L

M

Q

R

S

T

W

X

Y

Z