

SIEMENS

SIMATIC

S7-300 和 S7-400

语句表(STL)编程

参考手册

2006 年 3 月版
A5E00706960-01

前言，目录	
位逻辑指令	1
比较指令	2
转换指令	3
计数器指令	4
数据块指令	5
逻辑控制指令	6
整数运算指令	7
浮点数运算指令	8
装载和传送指令	9
程序控制指令	10
移位和循环移位指令	11
定时器指令	12
字逻辑指令	13
累加器指令	14
附录	
所有语句表指令一览	A
编程举例	B
参数传递	C
索引	

安全指南

本手册包括应该遵守的注意事项，以保证人身安全及财产损失。在本手册中，与人身安全有关的注意事项通过安全警告符号突出显示，而只与财产损失有关的注意事项则没有安全警告符号。这些注意事项根据危险等级显示如下：



危险

表示若不采取适当的预防措施，将导致死亡或严重的人身伤害。



警告

表示若不采取适当的预防措施，将可能导致死亡或严重的人身伤害。



小心

带安全警告符号时，表示若不采取适当的预防措施，将导致轻微的人身伤害。

小心

不带安全警告符号时，表示若不采取适当的预防措施，将造成财产损失。

注意

如果不引起相应的重视，将会导致意外的结果或状态。

当出现多个安全等级时，应该采用最高危险等级的安全提示。带安全警告符号的人员伤害警告也可能会导致财产损失。

合格人员

只有合格人员才允许对设备/系统进行调试和操作。合格人员规定为根据既定的安全惯例和标准，被授权对设备、系统和电路进行调试、接地和加装标签的人员。

正确使用

请注意如下事项：



警告

该装置及其组件只能用于产品目录或技术说明书中所阐述的应用，并且只能与由西门子公司认可或推荐的第三方厂商提供的设备或组件一起使用。

本产品只有在正确的运输、贮存、设置和安装以及仔细地运行和维护的情况下，才能正确而安全地运行。

商标

标有®的所有名称均为西门子的注册商标。

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

郑重声明

我们已核对过本手册的内容与所述硬件和软件相符。由于差错在所难免，我们不能保证完全的一致。我们会定期审查本手册中的内容，并在后续版本中进行必要的修正。欢迎提出改进意见。

西门子公司
自动化与驱动集团

西门子公司版权所有©2006
若有改动，恕不另行通知。

前言

目的

本使用手册旨在指导您如何用语句表编程语言（STL）编制用户程序。
本手册中还包含对描述 STL 语言元素的语法和功能的参考部分。

所需基本知识

本手册旨在用于编程人员、操作人员以及维护和维修人员。
为了很好理解本手册，需要具有自动化技术的一般知识。
除此之外，还需要具备计算机知识以及熟悉操作系统MS Windows 2000 Professional、MS Windows Server 2003 或 XP Professional。

本手册的应用范围

本手册适用于 STEP 7 编程软件包的 5.4 版。

符合标准

STL 符合国际电工委员会标准 IEC 1131-3 中定义的“指令表”编程语言，但在操作方面有很大的不同。关于详细信息，请参考 STEP 7 文件 NORM_TBL.WRI 中的标准列表。

要求

为了有效地使用这本语句表手册，你要预先熟悉 STEP 7 在线帮助中 S7 编程理论。该编程软件包也使用 STEP 7 标准软件，所以你要熟练使用这个软件并阅读相关的资料。

本手册是“STEP 7 参考资料”整套资料的一部分。

下表所示为 STEP 7 的整套资料：

资 料	用 途	订 货 号
STEP 7 基本信息 <ul style="list-style-type: none">STEP 7 V5.3, 《快速入门手册》STEP 7 V5.3 编程配置硬件和通讯连接, STEP 7 V5.3《从 S5 到 S7 转换手册》	向技术人员解释关于使用 STEP 7 以及 S7-300/400 可编程控制器实现控制任务的方法的基本信息。	6ES7810-4CA08-8BW0
STEP 7 参考资料 <ul style="list-style-type: none">《S7-300/400 梯形逻辑 (LAD) / 功能块图 (FBD) / 语句表 (STL) 使用手册》S7-300/400 标准和系统功能手册	介绍一些参考信息以及编程语言 LAD、FBD 和 STL 以及 STEP 7 基本信息的扩展标准功能和系统功能。	6ES7810-4CA08-8BW1

在线帮助	用 途	订 货 号
STEP 7 帮助	以在线帮助的形式提供关于使用 STEP 7 编程和组态硬件的基本信息。	为 STEP 7 标准软件包的一部分
STL/LAD/FBD 参考帮助 系统功能块 / 系统功能 (SFB / SFC) 参考帮助 组织块参考帮助	上下文相关信息	为 STEP 7 标准软件包的一部分

在线帮助

集成在软件中的在线帮助是本手册的补充。

在线帮助的目的是为你提供详细的软件使用帮助。

帮助系统通过多个界面集成在软件中：

- 上下文相关帮助可以提供关于当前的文本信息，例如，一个打开的对话框或一个激活的窗口。你可以按动 F1 或使用工具栏中的“？”，通过菜单命令 **Help > Context-Sensitive Help**，打开文本相关的帮助。
- 你可以使用菜单命令 **Help > Contents** 或文本相关帮助窗口中的“Help on STEP 7”按钮，调用 STEP 7 中的一般帮助信息。
- 你也可以通过“Glossary（术语）”按钮，调用所有 STEP 7 应用的术语。

本手册是“语句表中的帮助信息”摘选。由于手册和在线帮助的结构一样，所以能够很容易地在手册和在线帮助之间进行转换。

其它支持

如果你有任何技术问题，你可以与当地的西门子代表处或代理商联系。

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

培训中心

西门子公司还提供有许多培训课程，介绍 SIMATIC S7 自动化系统。详情请与您所在地区的培训中心联系，或与德国纽伦堡（邮编 D90327）的总部培训中心联系：

电话： +49 (911) 895-3200.

网址： <http://www.sitrain.com>

<http://www.ad.siemens.com.cn/training>

北 京： (010) 8459 7518

上 海： (021) 6281 5933 - 305

广 州： (020) 3761 9458

武 汉： (027) 8548 6688 - 6400

沈 阳： (024) 2294 9886

重 庆： (023) 6382 8919 - 3002

A&D 技术支持

总部（纽伦堡） 技术支持 一天 24 小时，一年 365 天全天候服务 电话：+49 (0) 180 5050-222 传真：+49 (0) 180 5050-223 E-Mail: adsupport@siemens.com GMT: +1:00		
欧洲/非洲（纽伦堡） 授权 当地时间：星期一到星期五 08:00:00 - 17:00 电话：+49 (0) 180 5050-222 传真：+49 (0) 180 5050-223 E-Mail: adsupport@siemens.com GMT: +1:00	美国（约翰森城） 技术支持和授权 当地时间：星期一到星期五 08:00:00 - 17:00 电话：+1 (0) 770 740 3505 传真：+1 (0) 770 740 3699 E-Mail:isd-callcenter@sea. siemens.com GMT: -5:00	亚洲/澳大利亚（北京） 技术支持和授权 当地时间：星期一到星期五 08:30 - 17:30 电话：+86 10 64 75 75 75 传真：+86 10 64 74 74 74 E-Mail: adsupport.asia@siemens.com GMT: +8:00
SIMATIC 热线和授权热线的使用语言一般为德语和英语。		

网上服务和技术支持

除了纸文件资料以外，我们在网上还提供有在线资料：

<http://www.siemens.com/automation/service&support> (英文网站)

<http://www.ad.siemens.com.cn/service> (中文网站)

在网上你可以找到：

- 新闻列表可以向你提供不断更新的最新产品信息。
- 通过网上服务和技术支持部分的搜索功能，可以找到所需文件。
- 在论坛部分，全世界的用户和专家都可交流其经验。
- 通过我们在网上的代表处数据库，你可以找到当地的自动化与驱动集团代表处。
- 有关现场服务、修理、备件等更多信息，可参见“服务”。

北 京：(010) 6471 9990

大 连：(0411) 369 9760 - 40

上 海：(021) 5879 5255

广 州：(020) 8732 3967

成 都：(028) 6820 0939

目录

前言.....	iii
目录.....	vii
1 位逻辑指令.....	1-1
1.1 位逻辑指令概述.....	1-1
1.2 A “与”.....	1-3
1.3 AN “与非”.....	1-4
1.4 O “或”.....	1-5
1.5 ON “或非”.....	1-6
1.6 X “异或”.....	1-7
1.7 XN “异或非”.....	1-8
1.8 O 先“与”后“或”.....	1-9
1.9 A(“与”运算嵌套开始.....	1-10
1.10 AN(“与非”运算嵌套开始.....	1-11
1.11 O(“或”运算嵌套开始.....	1-11
1.12 ON(“或非”运算嵌套开始.....	1-12
1.13 X(“异或”运算嵌套开始.....	1-12
1.14 XN(“异或非”运算嵌套开始.....	1-13
1.15) 嵌套结束.....	1-14
1.16 = 赋值.....	1-15
1.17 R 复位.....	1-16
1.18 S 置位.....	1-17
1.19 NOT RLO 取反.....	1-18
1.20 SET RLO 置位 (=1).....	1-18
1.21 CLR RLO 清零 (=0).....	1-19
1.22 SAVE 把 RLO 存入 BR 寄存器.....	1-20
1.23 FN 下降沿.....	1-21
1.24 FP 上升沿.....	1-23
2 比较指令.....	2-1
2.1 比较指令概述.....	2-1
2.2 ?I 整数比较 (16 位).....	2-2
2.3 ?D 双整数比较 (32 位).....	2-3
2.4 ?R 浮点数比较 (32 位).....	2-4
3 转换指令.....	3-1
3.1 转换指令概述.....	3-1
3.2 BTI BCD 转成整数 (16 位).....	3-2
3.3 ITB 整数 (16 位) 转成 BCD.....	3-3
3.4 BTD BCD 转成整数 (32 位).....	3-4

3.5	ITD 整数 (16 位) 转成双整数 (32 位)	3-5
3.6	DTB 双整数 (32 位) 转成 BCD	3-6
3.7	DTR 双整数 (32 位) 转成浮点数 (32 位, IEEE-FP)	3-7
3.8	INVI 对整数 (16 位) 求反码	3-8
3.9	INVD 对双整数 (32 位) 求反码	3-9
3.10	NEGI 对整数 (16 位) 求补码	3-10
3.11	NEGD 对双整数 (32 位) 求补码	3-11
3.12	NEGR 对浮点数 (32 位, IEEE-FP) 求反	3-12
3.13	CAW 交换累加器 1 低字 (16 位) 中的字节顺序	3-13
3.14	CAD 交换累加器 1 (32 位) 中的字节顺序	3-14
3.15	RND 取整	3-15
3.16	TRUNC 截尾取整	3-16
3.17	RND+ 向上取整为双整数	3-17
3.18	RND- 向下取整为双整数	3-18
4	计数器指令	4-1
4.1	计数器指令概述	4-1
4.2	FR 使能计数器 (任意)	4-2
4.3	L 将当前计数器值装入累加器 1	4-3
4.4	LC 将当前计数器值以 BCD 码形式装入累加器 1	4-4
4.5	R 计数器复位	4-5
4.6	S 设置计数器预设值	4-6
4.7	CU 加计数器	4-7
4.8	CD 减计数器	4-8
5	数据块指令	5-1
5.1	数据块指令概述	5-1
5.2	OPN 打开数据块	5-2
5.3	CDB 交换共享数据块和背景数据块	5-3
5.4	L DBLG 将共享数据块的长度装入累加器 1 中	5-3
5.5	L DBNO 将共享数据块的编号装入累加器 1 中	5-4
5.6	L DILG 将背景数据块的长度装入累加器 1 中	5-4
5.7	L DINO 将背景数据块的编号装入累加器 1 中	5-5
6	逻辑控制指令	6-1
6.1	逻辑控制指令概述	6-1
6.2	JU 无条件跳转	6-3
6.3	JL 跳转到标号	6-4
6.4	JC 若 RLO = 1, 则跳转	6-5
6.5	JCN 若 RLO = 0, 则跳转	6-6
6.6	JCB 若 RLO = 1 则跳转, 同时对 BR 置位	6-7
6.7	JNB 若 RLO = 0 则跳转, 同时对 BR 复位	6-8
6.8	JB 若 BR = 1, 则跳转	6-9
6.9	JNBI 若 BR = 0, 则跳转	6-10
6.10	JO 若 OV = 1, 则跳转	6-11

6.11	JOS 若 OS = 1, 则跳转	6-12
6.12	JZ 若零, 则跳转	6-13
6.13	JN 若非零, 则跳转	6-14
6.14	JP 若正, 则跳转	6-15
6.15	JM 若负, 则跳转	6-16
6.16	JPZ 若大于或等于零, 则跳转	6-17
6.17	JMZ 若小于或等于零, 则跳转	6-18
6.18	JUO 若运算无效, 则跳转	6-19
6.19	LOOP 循环控制	6-20
7	整数算术运算指令	7-1
7.1	整数算术运算指令概述	7-1
7.2	判断整数算术运算指令后状态字的位	7-2
7.3	+I 将累加器 1 和累加器 2 中的内容相加(16 位整数运算)	7-3
7.4	-I 将累加器 2 的内容减累加器 1 的内容(16 位整数运算)	7-4
7.5	*I 将累加器 1 和累加器 2 中的内容相乘(16 位整数运算)	7-5
7.6	/I 将累加器 2 的内容除以累加器 1 的内容(16 位整数运算)	7-6
7.7	+ 加整型常数 (16 位, 32 位)	7-7
7.8	+D 将累加器 1 和累加器 2 的内容相加(32 位双整数运算)	7-9
7.9	-D 累加器 2 的内容减累加器 1 的内容(32 位双整数运算)	7-10
7.10	*D 将累加器 1 和累加器 2 的内容相乘(32 位双整数运算)	7-11
7.11	/D 累加器 2 的内容除以累加器 1 的内容(32 位双整数运算)	7-12
7.12	MOD 双整数除法的余数 (32 位)	7-13
8	浮点数运算指令	8-1
8.1	浮点数运算指令概述	8-1
8.2	判断浮点数运算指令后状态字的位	8-2
8.3	浮点数运算指令: 基本指令	8-3
8.3.1	+R 将累加器 1 和累加器 2 中的内容相加(32 位浮点数, IEEE-FP)	8-3
8.3.2	-R 将累加器 2 中的内容减去累加器 1 中的内容(32 位浮点数, IEEE-FP)	8-4
8.3.3	*R 将累加器 1 和累加器 2 中的内容相乘(32 位浮点数, IEEE-FP)	8-5
8.3.4	/R 累加器 2 的内容除以累加器 1 的内容(32 位浮点数, IEEE-FP)	8-6
8.3.5	ABS 取绝对值 (32 位浮点数, IEEE-FP)	8-7
8.4	浮点数运算指令: 扩展指令	8-8
8.4.1	SQR 平方运算 (32 位)	8-8
8.4.2	SQRT 开方运算 (32 位)	8-9
8.4.3	EXP 指数运算 (32 位)	8-10
8.4.4	LN 自然对数运算 (32 位)	8-11
8.4.5	SIN 正弦运算 (32 位)	8-12
8.4.6	COS 余弦运算 (32 位)	8-13
8.4.7	TAN 正切运算 (32 位)	8-14
8.4.8	ASIN 反正弦运算 (32 位)	8-15
8.4.9	ACOS 反余弦运算 (32 位)	8-16
8.4.10	ATAN 反正切运算 (32 位)	8-17

9	装载和传送指令	9-1
9.1	装载和传送指令概述	9-1
9.2	L 装载	9-2
9.3	L STW 将状态字装入累加器 1	9-4
9.4	LAR1 将累加器 1 中的内容装入地址寄存器 1	9-5
9.5	LAR1 <D> 将双整数(32 位指针)装入地址寄存器 1	9-6
9.6	LAR1 AR2 将地址寄存器 2 的内容装入地址寄存器 1	9-7
9.7	LAR2 将累加器 1 中的内容装入地址寄存器 2	9-7
9.8	LAR2 <D> 将双整数(32 位指针)装入地址寄存器 2	9-8
9.9	T 传送	9-9
9.10	T STW 将累加器 1 中的内容传送到状态字	9-10
9.11	CAR 交换地址寄存器 1 和地址寄存器 2 的内容	9-11
9.12	TAR1 将地址寄存器 1 中的内容传送到累加器 1	9-11
9.13	TAR1 <D>将地址寄存器 1 的内容传送到目的地(32 位指针)	9-12
9.14	TAR1 AR2 将地址寄存器 1 的内容传送到地址寄存器 2	9-13
9.15	TAR2 将地址寄存器 2 中的内容传送到累加器 1	9-13
9.16	TAR2 <D>将地址寄存器 2 的内容传送到目的地(32 位指针)	9-14
10	程序控制指令	10-1
10.1	程序控制指令概述	10-1
10.2	BE 块结束	10-2
10.3	BEC 块有条件结束	10-3
10.4	BEU 块无条件结束	10-4
10.5	CALL 块调用	10-5
10.6	调用功能块	10-8
10.7	调用功能	10-10
10.8	调用系统功能块	10-11
10.9	调用系统功能	10-13
10.10	调用多背景块	10-14
10.11	从库中调用块	10-14
10.12	CC 条件调用	10-15
10.13	UC 无条件调用	10-16
10.14	MCR (主控继电器)	10-17
10.15	使用 MCR 功能的重要注意事项	10-19
10.16	MCR(将 RLO 存入 MCR 堆栈, 开始 MCR	10-20
10.17)MCR 结束 MCR	10-22
10.18	MCRA 激活 MCR 程序段	10-23
10.19	MCRD 取消激活 MCR 程序段	10-24
11	移位和循环移位指令	11-1
11.1	移位指令	11-1
11.1.1	移位指令概述	11-1
11.1.2	SSI 移位有符号整数 (16 位)	11-2
11.1.3	SSD 移位有符号双整数 (32 位)	11-3

11.1.4	SLW 字 (16 位) 左移	11-5
11.1.5	SRW 字 (16 位) 右移	11-6
11.1.6	SLD 双字 (32 位) 左移	11-7
11.1.7	SRD 双字 (32 位) 右移	11-8
11.2	循环移位指令	11-10
11.2.1	循环移位指令概述	11-10
11.2.2	RLD 双字 (32 位) 循环左移	11-10
11.2.3	RRD 双字 (32 位) 循环右移	11-12
11.2.4	RLDA 通过 CC 1 (32 位) 对累加器 1 循环左移	11-13
11.2.5	RRDA 通过 CC 1 (32 位) 对累加器 1 循环右移	11-14
12	定时器指令	12-1
12.1	定时器指令概述	12-1
12.2	存储器中定时器的位置和定时器的组件	12-2
12.3	FR 使能定时器 (任意)	12-5
12.4	L 将当前定时值作为整数装入累加器 1	12-7
12.5	LC 将当前定时器值作为 BCD 码装入累加器 1	12-8
12.6	R 复位定时器	12-9
12.7	SP 脉冲定时器	12-10
12.8	SE 扩展脉冲定时器	12-11
12.9	SD 延时接通定时器	12-13
12.10	SS 保持型延时接通定时器	12-14
12.11	SF 延时断开定时器	12-16
13	字逻辑指令	13-1
13.1	字逻辑指令概述	13-1
13.2	AW 字“与” (16 位)	13-2
13.3	OW 字“或” (16 位)	13-3
13.4	XOW 字“异或” (16 位)	13-4
13.5	AD 双字“与” (32 位)	13-6
13.6	OD 双字“或” (32 位)	13-7
13.7	XOD 双字“异或” (32 位)	13-8
14	累加器操作指令	14-1
14.1	累加器和地址寄存器操作指令概述	14-1
14.2	TAK 累加器 1 与累加器 2 进行互换	14-2
14.3	POP 带有两个累加器的 CPU	14-3
14.4	POP 带有四个累加器的 CPU	14-4
14.5	PUSH 带有两个累加器的 CPU	14-5
14.6	PUSH 带有四个累加器的 CPU	14-6
14.7	ENT 进入累加器堆栈	14-7
14.8	LEAVE 离开累加器堆栈	14-7
14.9	INC 增加累加器 1 低字的低字节内容	14-8
14.10	DEC 减少累加器 1 低字的低字节内容	14-9
14.11	+AR1 将累加器 1 加到地址寄存器 1	14-10

14.12	+AR2 将累加器 1 加到地址寄存器 2	14-11
14.13	BLD 程序显示指令（空）	14-12
14.14	NOP 0 空操作指令	14-13
14.15	NOP 1 空操作指令	14-13
A	所有语句表指令一览	A-1
A.1	按德文助记符分类的语句表指令	A-1
A.2	按英文助记符分类的语句表指令（国际）	A-6
B	编程举例	B-1
B.1	编程举例概述	B-1
B.2	例如：位逻辑指令	B-2
B.3	例如：定时器指令	B-5
B.4	例如：计数器和比较指令	B-8
B.5	例如：整数算术运算指令	B-10
B.6	例如：字逻辑指令	B-11
C	参数传递	C-1

1 位逻辑指令

1.1 位逻辑指令概述

说明

位逻辑指令使用两个数字：“1”和“0”。这两个数字构成二进制数字系统的基础。“1”和“0”称为二进制数字或二进制位。对触点与线圈而言，“1”表示动作或通电，“0”表示未动作或未通电。

位逻辑指令解释信号状态 1 和 0，并根据布尔逻辑对它们进行组合。这些组合产生结果 1 或 0，称为“逻辑运算结果（RLO）”。

布尔位逻辑使用下列基本指令：

- A “与”
- AN “与非”
- O “或”
- ON “或非”
- X “异或”
- XN “异或非”
- O “先与后或”

可使用以下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(“异或非” 运算嵌套开始
-) 嵌套结束

使用以下指令，可以结束一个布尔位逻辑串：

- = 赋值
- R 复位
- S 置位

可使用下述指令之一，更改逻辑运算的结果（RLO）：

- NOT RLO 取反
- SET RLO 置位（=1）
- CLR RLO 清零（=0）
- SAVE 把 RLO 存入 BR 寄存器

其它指令对上升沿和下降沿有反应：

- FN 下降沿
- FP 上升沿

1.2 A “与”

格式

A <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D、T、C

说明

使用“与”指令可以检查被寻址位的信号状态是否为“1”，并将测试结果与逻辑运算结果（RLO）进行“与”运算。

状态字位检查：

“与”指令也可通过使用以下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

举例

语句表程序	继电器逻辑图	
A I 1.0	I 1.0 信号状态 “1”	常开触点
A I 1.1	I 1.1 信号状态 “1”	常开触点
= Q 4.0	Q 4.0 信号状态 “1”	线圈
▼ 显示为开关闭合		

1.3AN“与非”

格式

N <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D、T、C

说明

使用“与非”指令可以检查被寻址位的信号状态是否为“0”，并将测试结果与逻辑运算结果（RLO）进行“与”运算。

“与非”指令也可通过使用以下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

举例

语句表程序	继电器逻辑图	
A I 1.0	I 1.0 信号状态“0”	常开触点
AN I 1.1	I 1.1 信号状态“1”	常闭触点
= Q 4.0	Q 4.0 信号状态“0”	线圈

1.4 O “或”

格式

O <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D、T、C

说明

使用“或”指令可以检查被寻址位的信号状态是否为“1”，并将测试结果与逻辑运算结果（RLO）进行“或”运算。

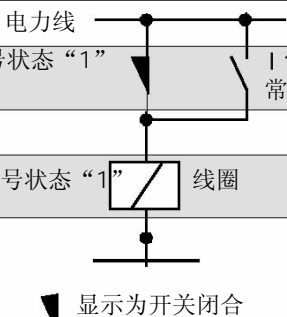
状态字位检查：

“或”指令也可通过使用以下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

举例

语句表程序	继电器逻辑图
	
O I 1.0	
O I 1.1	
= Q 4.0	

1.5 ON “或非”

格式

ON <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D、T、C

说明

使用“或非”指令可以检查被寻址位的信号状态是否为“0”，并将测试结果与逻辑运算结果（RLO）进行“或”运算。

状态字位检查：

“或非”指令也可通过使用以下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

举例

语句表程序	继电器逻辑图
O I 1.0	I 1.0 信号状态 0
ON I 1.1	I 1.1 信号状态 1
= Q 4.0	Q 4.0 信号状态 “1”

1.6 X “异或”

格式

X <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D、T、C

说明

使用“异或”指令可以检查被寻址位的信号状态是否为“1”，并将测试结果与逻辑运算结果（RLO）进行“异或”运算。

也可以多次使用“异或”指令。因此，如果有奇数个被检查地址为“1”，则逻辑运算的交互结果为“1”。

状态字位检查：

“异或”指令也可通过使用以下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

举例

语句表程序		继电器逻辑图	
		电力线	
X	I 1.0	触点 I 1.0	
X	I 1.1	触点 I 1.1	
=	Q 4.0	Q 4.0 线圈	

1.7 XN “异或非”

格式

XN <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D、T、C

说明

使用“异或非”指令可以检查被寻址位的信号状态是否为“0”，并将测试结果与逻辑运算结果（RLO）进行“异或”运算。

状态字位检查：

“异或非”指令也可通过使用以下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

举例

语句表程序		继电器逻辑图	
		电力线	
X	I 1.0	触点 I 1.0	
XN	I 1.1	触点 I 1.1	
=	Q 4.0	Q 4.0 线圈	

1.8 O 先“与”后“或”

格式

O

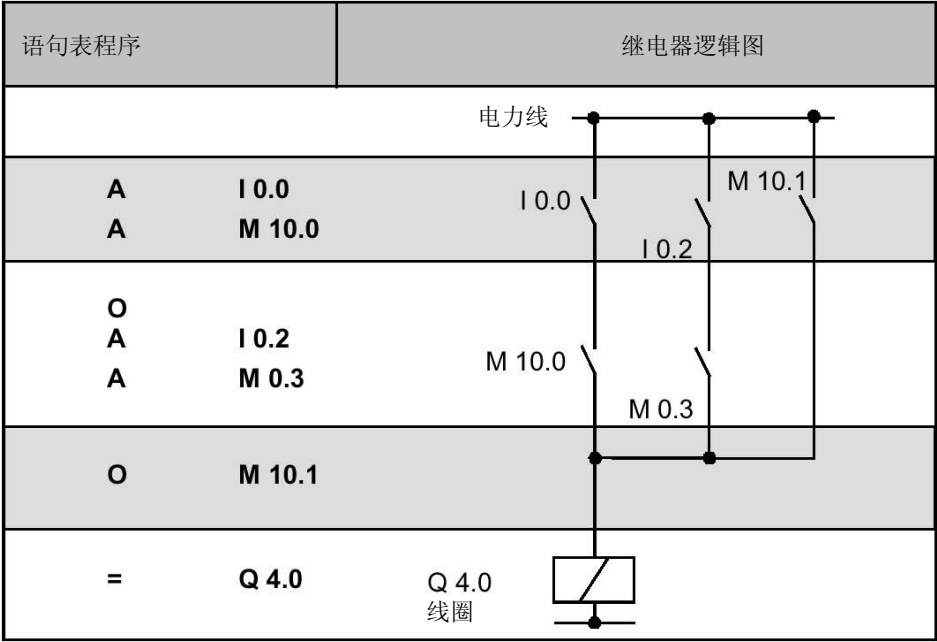
说明

“先与后或（O）”指令根据以下规则对“与”运算结果执行“或”运算：先“与”后“或”。

状态字

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

举例



1.9 A(“与” 运算嵌套开始

格式

A(

说明

A(（“与”运算嵌套开始）可以将 RLO 和 OR 位以及一个函数代码保存到嵌套堆栈中。最多可有 7 个嵌套堆栈输入项。

状态字

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

举例

语句表程序	继电器逻辑图
	电力线
A(O O) I 0.0 M 10.0	I 0.0 M 10.0
A(O O) I 0.2 M 10.3	I 0.2 M10.3
A M 10.1	M 10.1
= Q 4.0	Q 4.0 线圈

1.10 AN(“与非”运算嵌套开始

格式

AN(

说明

AN((“与非”运算嵌套开始) 可以将 RLO 和 OR 位以及一个函数代码保存到嵌套堆栈中。最多可有 7 个嵌套堆栈输入项。

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(

说明

O((“或”操作嵌套开始) 可以将 RLO 和 OR 位以及一个函数代码保存到嵌套堆栈中。最多可有 7 个嵌套堆栈输入项。

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(

说明

ON(（“或非”运算嵌套开始）可以将 RLO 和 OR 位以及一个函数代码保存到嵌套堆栈中。最多可有 7 个嵌套堆栈输入项。

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(

说明

X(（“异或”运算嵌套开始）可以将 RLO 和 OR 位以及一个函数代码保存到嵌套堆栈中。最多可有 7 个嵌套堆栈输入项。

状态字

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

1.14 XN(“异或非”运算嵌套开始

格式

XN(

说明

XN((“异或非”运算嵌套开始) 可以将 RLO 和 OR 位以及一个函数代码保存到嵌套堆栈中。最多可有 7 个嵌套堆栈输入项。

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

)

说明

使用)（嵌套结束）指令，可以从嵌套堆栈中删除一个输入项，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（与）”或“AND NOT（与非）”，则也包括 OR 位。

打开括号组的语句：

- 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(“异或非”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

举例

语句表程序		继电器逻辑图	
		电力线	
A(O O)	I 0.0 M 10.0		
A(O O)	I 0.2 M 10.3		
A	M 10.1		
=	Q 4.0	Q 4.0 线圈	

1.17 R 复位

格式

R <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D

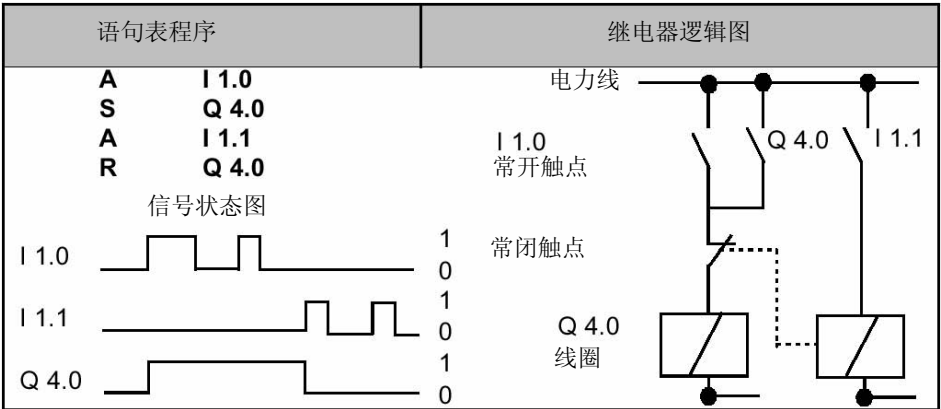
说明

如果 RLO = 1，并且主控继电器 MCR = 1，则复位指令（R）可以将寻址位复位为“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

举例



1.18 S 置位

格式

S <位>

地 址	数据类型	存储区
<位>	BOOL	I、Q、M、L、D

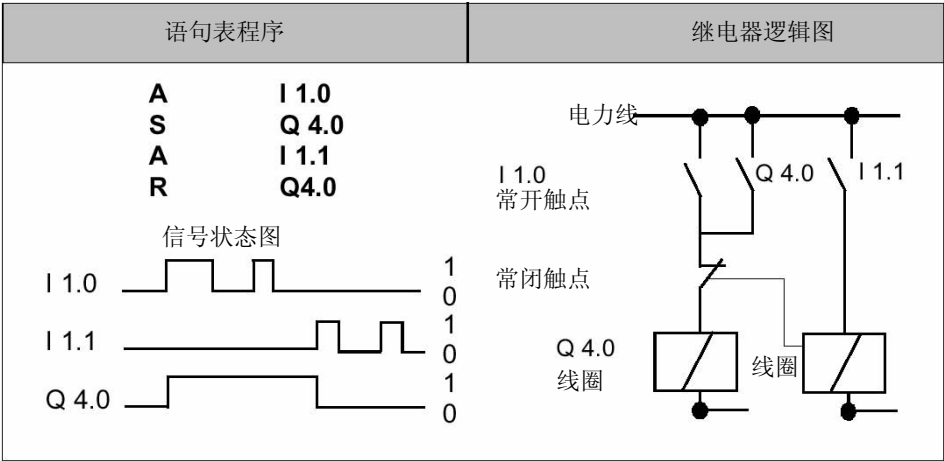
指令说明

如果 RLO = 1，并且主控继电器 MCR = 1，则置位指令（S）可以将寻址位置位为“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

举例



1.19 NOT RLO 取反

格式

NOT

说明

使用取反（NOT）指令，可以对 RLO 取反。

状态字

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

1.20 SET RLO置位 (=1)

格式

SET

说明

使用 RLO 置位（SET）指令，可以将 RLO 的信号状态置为“1”。

状态字

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

举例

语句表程序	信号状态	逻辑运算结果（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 RLO 清零 (=0)

格式

CLR

说明

使用 RLO 清零 (CLR) 指令，可以将 RLO 的信号状态置为“0”。

状态字

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

举例

语句表程序	信号状态	逻辑运算结果 (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	

1.22 **SAVE 把 RLO 存入 BR 寄存器**

格式

SAVE

指令说明

使用 SAVE 指令，可以将 RLO 存入 BR 位。首个检查位 /FC 不被复位。为此，BR 位的状态包括在下一程序段的“与”（AND）逻辑运算中。

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	说 明
<位>	BOOL	I、Q、M、L、D	沿标志，存储 RLO 的上一信号状态。

说明

使用 RLO 下降沿检测指令（FN <位>）可以在 RLO 从“1”变为“0”时检测到下降沿，并以 RLO = 1 显示。

在每一个程序扫描周期内，RLO 位的信号状态将与上一个周期中获得的RLO位信号状态进行比较，看是否有变化。上一个周期的 RLO 信号状态必须保存在沿标志位地址（<位>）中，以便进行比较。如果在当前和先前的 RLO “1” 状态之间发生变化（检测到下降沿），则在该指令执行后，RLO 位将为“1”。

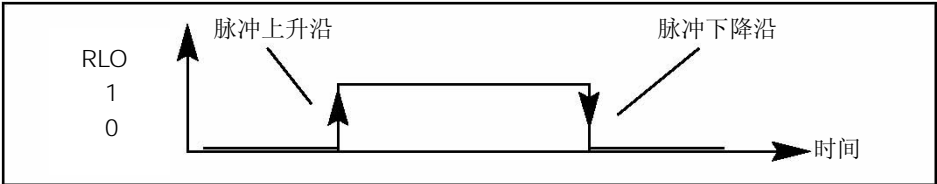
注意

由于一个块的本地数据只在块运行期间有效，如果想要监视的位在过程映像中，则该指令就不起作用。

状态字

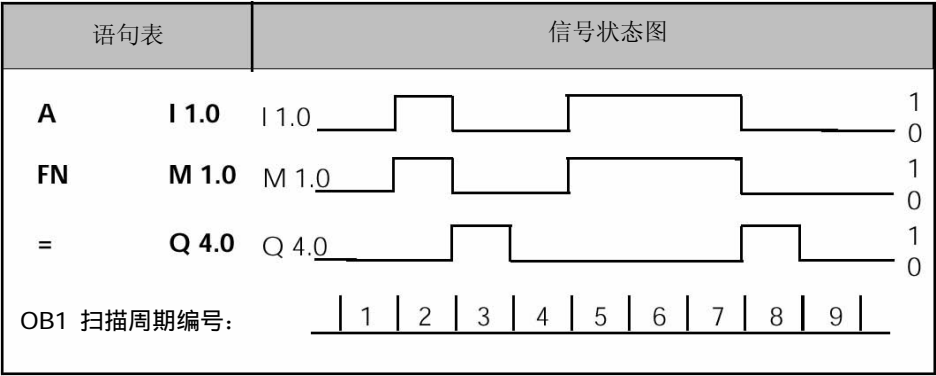
	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	x	x	1

定义



举例

如果可编程控制器在触点 I 1.0 检测到一个下降沿，则它会在一个 OB1 扫描周期内使 Q 4.0 线圈得电。



1.24 FP 上升沿

格式

FP <位>

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

说明

使用 RLO 上升沿检测指令（FP <位>）可以在 RLO 从“0”变为“1”时检测到一个上升沿，并以 RLO = 1 显示。

在每一个程序扫描周期内，RLO 位的信号状态将与上一个周期中获得的RLO位信号状态进行比较，看是否有变化。上一个周期的 RLO 信号状态必须保存在沿标志位地址（<位>）中，以便进行比较。如果在当前和先前的 RLO “0” 状态之间发生变化（检测到上升沿），则在该指令执行后，RLO 位将为“1”。

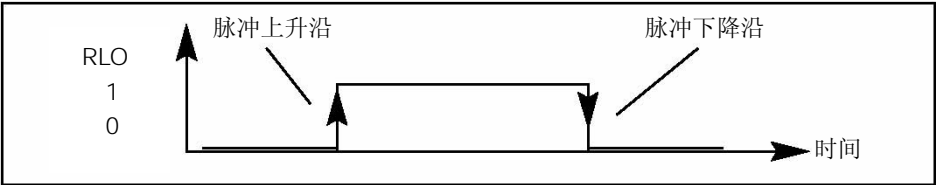
注意

由于一个块的本地数据只在块运行期间有效，如果想要监视的位在过程映像中，则该指令就不起作用。

状态字

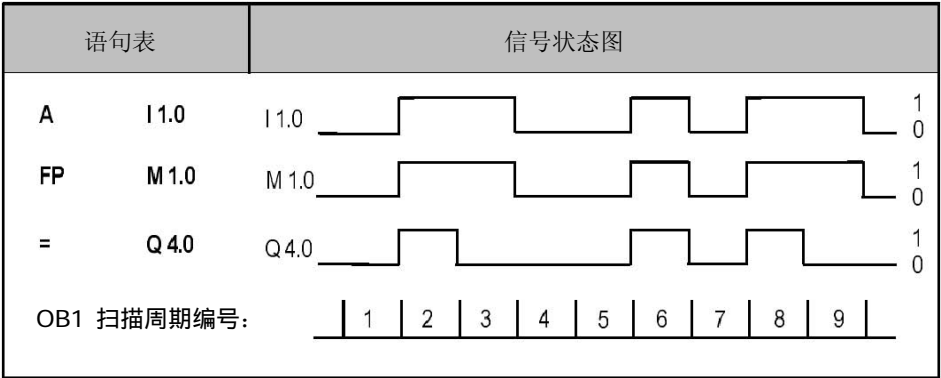
	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
写:	-	-	-	-	-	0	x	x	1

定义



举例

如果可编程控制器在触点 I 1.0 检测到一个下降沿，则它会在一个 OB1 扫描周期内使 Q 4.0 线圈得电。



2 比较指令

2.1 比较指令概述

说明

根据所选择的比较类型，对累加器 1（ACCU1）和累加器 2（ACCU2）进行比较：

==	累加器 1 等于累加器 2
<>	累加器 1 不等于累加器 2
>	累加器 1 大于累加器 2
<	累加器 1 小于累加器 2
>=	累加器 1 大于等于累加器 2
<=	累加器 1 小于等于累加器 2

如果比较结果为真，则指令的 RLO 为“1”。状态字位 CC 1 和 CC 0 表示“小于”、“等于”或“大于”关系。

可执行下列比较指令：

- ?I 整数比较（16位）
- ?D 双整数比较（32位）
- ?R 浮点数比较（32位）

2.2 ?I 整数比较（16位）

格式

=I , <>I , >I , <I , >=I , <=I

指令说明

使用整数（16 位）比较指令，可以将累加器 2 中低字的内容与累加器 1 中低字的内容进行比较。累加器 2 和累加器 1 低字的内容都作为 16 位整数。比较的结果以 RLO 以及相关状态字位的设置来表示。RLO = 1 表示比较的结果为“真”；RLO = 0 表示比较的结果为“假”。状态字位 CC 1 和 CC 0 表示“小于”、“等于”或“大于”关系。

状态字

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

RLO 值

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

举例

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	// 比较累加器 2 低字中的内容(MW10)是否大于累加器 1 低字中的内容(IW24)。
= M 2.0	// 如果 MW10 > IW24，则 RLO = 1。

2.3 ? D 双整数比较（32位）

格式

==D , <>D , >D , <D , >=D , <=D

指令说明

使用双整数（32 位）比较指令，可以将累加器 2 中的内容与累加器 1 中的内容进行比较。累加器 2 和累加器 1 的内容都作为 32 位整数。比较的结果以 RLO 以及相关状态字位的设置来表示。RLO = 1 表示比较的结果为“真”；RLO = 0 表示比较的结果为“假”。状态字位 CC 1 和 CC 0 表示“小于”、“等于”或“大于”关系。

状态字

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

RLO 值

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

举例

STL	解 释
L MD10	// 装载 MD10 的内容（32 位双整数）。
L ID24	// 装载 ID24 的内容（32 位双整数）。
>D	// 比较累加器 2 中的内容（MD10）是否大于累加器 1 中的内容（ID24）。
= M 2.0	// 如果 MD10 > ID24，则 RLO = 1。

2.4 ? R 浮点数比较（32位）

格式

==R , <>R , >R , <R , >=R , <=R

指令说明

使用浮点数（32 位，IEEE-FP）比较指令，可以将累加器 2 中的内容与累加器 1 中的内容进行比较。累加器 1 和累加器 2 的内容都作为 32 位浮点数（IEEE-FP）。比较的结果以 RLO 以及相关状态字位的设置来表示。RLO = 1 表示比较的结果为“真”；RLO = 0 表示比较的结果为“假”。状态字位 CC 1 和 CC 0 表示“小于”、“等于”或“大于”关系。

状态字

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

RLO 值

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

举例

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	// 比较累加器 2 中的内容（MD10）是否大于累加器 1 中的内容（1.359-E+02）。
= M 2.0	// 如果 MD10 > 1.359E+02，则 RLO = 1。

3 转换指令

3.1 转换指令概述

说明

你可以使用以下指令将二进制编码的十进制数（BCD）和整数转换为其它类型的数字：

- BTI BCD 转成整数（16位）
- ITB 整数（16位）转成 BCD
- BTD BCD 转成双整数（32位）
- ITD 整数（16 位）转成双整数（32 位）
- DTB 双整数（32位）转成 BCD
- DTR 双整数（32 位）转成浮点数（32 位，IEEE-FP）

你可以使用下述指令之一，求一个整数的补码，或改变一个浮点数的符号：

- INVI 对整数求反码（16 位）
- INV D 对双整数求反码（32 位）
- NEGI 对整数求补码（16 位）
- NEG D 对双整数求补码（32 位）
- NEGR 对浮点数求反（32 位，IEEE-FP）

你可以使用以下“改变累加器 1 中的位顺序”指令，交换累加器 1 低字中或整个累加器中的字节顺序：

- CAW 改变累加器 1 低字中的字节顺序（16 位）
- CAD 改变累加器 1 中的字节顺序（32 位）

你可以使用以下任一指令，将累加器 1 中的 32 位 IEEE 浮点数转换成 32 位整数（双整数）。各条指令的取整方法略有不同：

- RND 取整
- TRUNC 截尾取整
- RND+ 向上取整为双整数
- RND- 向下取整为双整数

3.2 BTI BCD 转成整数（16位）

格式

BTI

说明

使用 BTI 指令（3 位 BCD 数十进制 – 二进制转换），可以将累加器 1 低字中的内容解释为一个 3 位二进制编码的十进制数（BCD），并将其转换为一个 16 位整数。转换结果保存在累加器 1 的低字中。累加器 1 的高字和累加器 2 的内容保持不变。

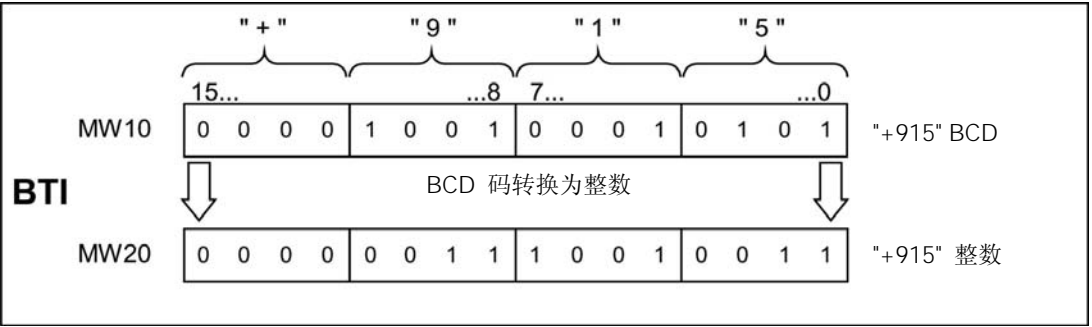
累加器 1 低字中的 BCD 数：BCD 数的允许范围为 -999 至 +999。位 0 到位 11 为 BCD 数的数值部分，位 15 为 BCD 数的符号位（0 = 正数，1 = 负数）。位 12 到位 14 在转换时不使用。如果是转换一个 4 位 BCD 数，并占用了无效的位 10 至位 15，则转换时会出现 BCDF 错误。通常，CPU 会进入“STOP（停机）”状态。但是，可以通过编写 OB121 出错响应程序，来处理该同步编程错误。

状态字

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

举例

STL	解 释
L MW10	// 将 BCD 数装入累加器 1 低字中。
BTI	// 将 BCD 数转换为整数；结果保存到累加器 1 低字中。
T MW20	// 将结果（整数）传送到 MW20。



3.3 ITB 整数（16位）转成 BCD

格式

ITB

说明

使用 ITB 指令（16 位整数的二进制 – 十进制转换），可以将累加器 1 低字中的内容解释为一个 16 位整数，并将其转换为一个 3 位二进制编码的十进制数（BCD）。转换结果保存在累加器 1 的低字中。位 0 – 11 为 BCD 数的数值部分。位 12 – 15 为 BCD 数的符号状态（0000 = 正数，1111= 负数）。累加器 1 的高字和累加器 2 的内容保持不变。

BCD 数的范围在 -999 和 +999 之间。如果超出允许范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置位为“1”。

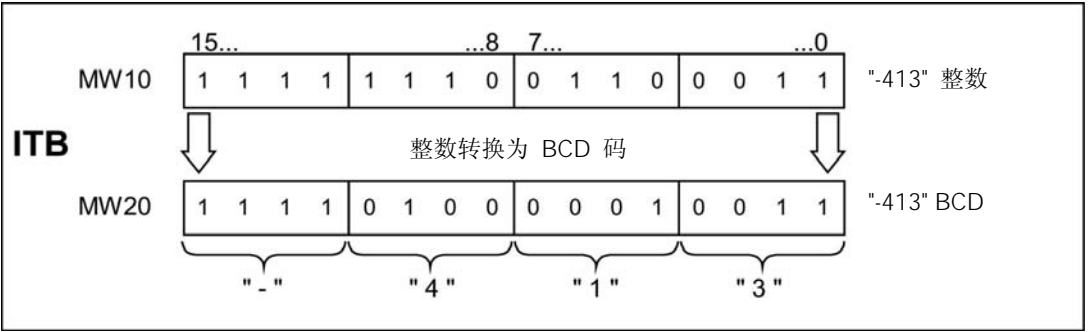
指令的执行与 RLO 无关，也不会影响 RLO。

状态字

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

举例

STL	解 释
L MW10	// 将整数装入累加器 1 低字中。
ITB	// 将整数转换为 BCD 数（16 位）；结果保存到累加器 1 低字中。
T MW20	// 将结果（BCD 数）传送到 MW20。



3.4 BTD BCD 转成整数（32位）

格式

BTD

说明

使用 BTD 指令（7 位 BCD 数十进制 – 二进制转换），可以将累加器 1 中的内容解释为一个 7 位二进制编码的十进制数（BCD），并将其转换为一个 32 位双整数。转换结果保存在累加器 1 中。累加器 2 保持不变。

累加器 1 中的 BCD 数：BCD 数的允许范围为 -9,999,999 至 +9,999,999。位 0 到位 27 为 BCD 数的数值部分，位 31 为 BCD 数的符号位（0 = 正数，1= 负数）。位 28 到位 30 在转换时不使用。

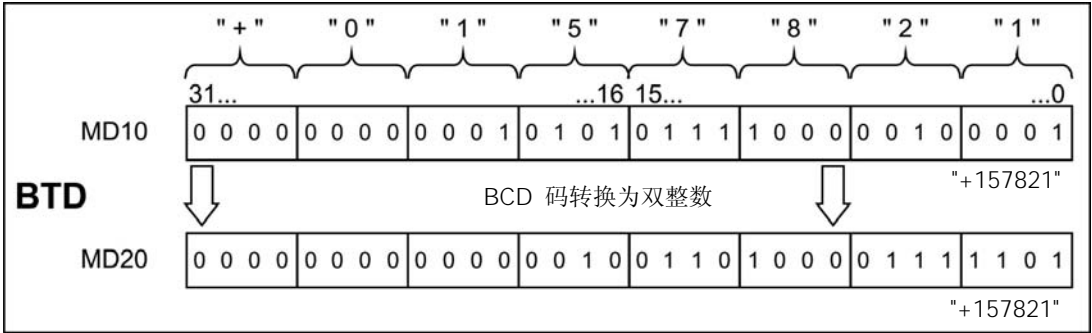
如果需转换的 BCD 数的任何一位十进制数（BCD 编码，4 位一组）在无效的 10 – 15 范围之间，则转换时会出现 BCDF 错误。通常，CPU 会进入“STOP（停机）”状态。但是，可以通过编写 OB121 出错响应程序，来处理该同步编程错误。

状态字

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

举例

STL	解 释
L MD10	// 将 BCD 数装入累加器 1。
BTD	// 将 BCD 数转换为双整数；结果保存到累加器 1 中。
T MD20	// 将结果（双整数）传送到 MD20。



3.5 ITD 整数（16 位）转成双整数（32 位）

格式

ITD

说明

使用 ITD 指令（16 位整数转换成为 32 位整数），可以将累加器 1 低字中的内容解释为一个 16 位整数，并将其转换为一个 32 位双整数。转换结果保存在累加器 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:	-	-	-	-	-	-	-	-	-

举例

STL	解 释
L MW12	// 将整数装入累加器 1。
ITD	// 将整数（16 位）转换为双整数（32 位）；结果保存到累加器 1 中。
T MD20	// 将结果（双整数）传送到 MD20。

例如：MW12 = “-10”（整数，16 位）

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
ITD 执行前	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
ITD 执行后	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 或 1, 这些位不用于转换)							

3.6 DTB 双整数 (32位) 转成 BCD

格式

DTB

说明

使用 DTB 指令（32 位整数的二进制 – 十进制转换），可以将累加器 1 中的内容解释为一个 32 位双整数，并将其转换为一个 7 位二进制编码的十进制数（BCD）。转换结果保存在累加器 1 中。位 0 – 27 为 BCD 数的数值部分。位 28 – 31 为 BCD 数的符号状态（0000 = 正数，1111= 负数）。累加器 2 保持不变。

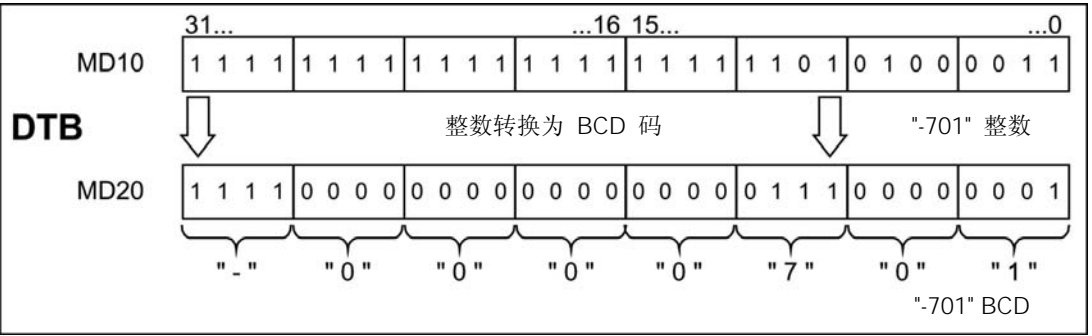
BCD 数的范围在 -9,999,999 和 +9,999,999 之间。如果超出允许范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置位为“1”。

状态字

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

举例

STL	解 释
L MD10	// 将 32 位整数装入累加器 1。
DTB	// 将整数（32 位）转换为 BCD 数；结果保存到累加器 1 中。
T MD20	// 将结果（BCD 数）传送到 MD20。



3.7 DTR 双整数（32 位）转成浮点数（32 位，IEEE-FP）

格式

DTR

说明

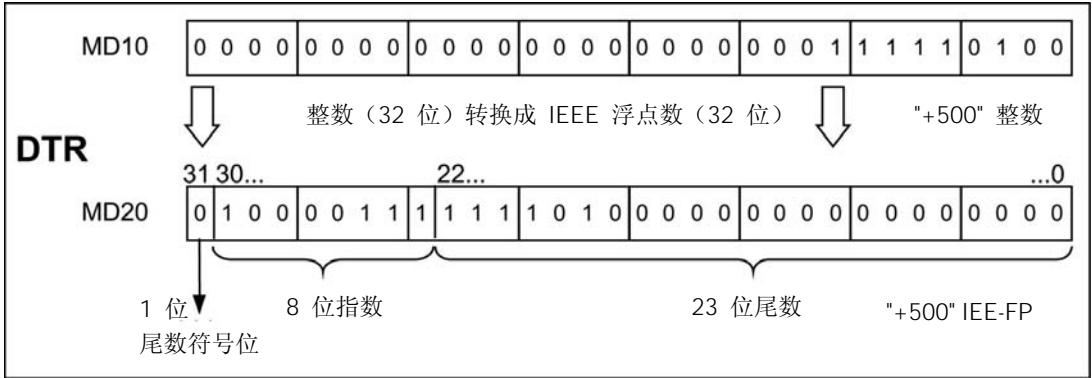
使用 DTR 指令(32 位整数转换成为 32 位 IEEE 浮点数),可以将累加器 1 中的内容解释为一个 32 位整数,并将其转换为一个 32 位 IEEE 浮点数。如果必要的话,该指令会对结果进行取整。(一个 32 位整数要比一个 32 位浮点数精度高)。结果保存在累加器 1 中。

状态字

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

举例

STL	解 释
L MD10	// 将 32 位整数装入累加器 1。
DTR	// 将双整数转换为浮点数（32 位，IEEE FP）；结果保存到累加器 1 中。
T MD20	// 将结果（BCD 数）传送到 MD20。



3.8 INVI 对整数（16 位）求反码

格式

INVI

说明

使用对整数求反码指令（INVI），可以对累加器 1 低字中的 16 位数值求反码。求反码指令为逐位转换，即“0”变为“1”，“1”变为“0”。其结果保存在累加器 1 的低字中。

状态字

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

举例

STL	解 释
L IW8	// 将数值装入累加器 1 低字中。
INVI	// 对 16 位数求反码。
T MW10	// 将结果传送到 MW10。

内 容	累加器 1 低字			
位	15...0
INVI 执行前	0110	0011	1010	1110
INVI 执行后	1001	1100	0101	0001

3.9 INVD 对双整数（32 位）求反码

格式

INVD

说明

使用对双整数求反码指令（INVD），可以对累加器 1 中的 32 位数值求反码。求反码指令为逐位转换，即“0”变为“1”，“1”变为“0”。其结果保存在累加器 1 中。

状态字

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

举例

STL	解 释
L ID8	// 将数值装入累加器 1 中。
INVD	// 对 32 位数求反码。
T MD10	// 将结果传送到 MD10。

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
INVD 执行前	0110	1111	1000	1100	0110	0011	1010	1110
INVD 执行后	1001	0000	0111	0011	1001	1100	0101	0001

3.10 NEGI 对整数（16 位）求补码

格式

NEGI

说明

使用对整数求补码指令（NEGI），可以对累加器 1 低字中的 16 位数值求补码。求补码指令为逐位转换，即“0”变为“1”，“1”变为“0”；然后对累加器中的内容加“1”。转换结果保存在累加器 1 的低字中。求补码指令相当于该数乘以“-1”。状态位 CC 1、CC 0、OS 和 OV 将根据运算的结果进行设置。

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	-
$-32768 \leq \text{结果} \leq -1$	0	1	0	-
$32767 \geq \text{结果} \geq 1$	1	0	0	-
结果 = 2768	0	1	1	1

举例

STL	解 释
L IW8	// 将数值装入累加器 1 低字中。
NEGI	// 对 16 位数求补码。
T MW10	// 将结果传送到 MW10。

内 容	累加器 1 低字			
位	15...0
NEGI 执行前	0101	1101	0011	1000
NEGI 执行后	1010	0010	1100	1000

3.11 NEGD 对双整数（32 位）求补码

格式

NEGD

说明

使用对双整数（NEGD）求补码指令，可以对累加器 1 中的 32 位数值求补码。求补码指令为逐位转换，即“0”变为“1”，“1”变为“0”；然后对累加器中的内容加“1”。转换结果保存在累加器 1 中。求补码指令相当于该数乘以“-1”。指令的执行与 RLO 无关，也不影响 RLO。状态位 CC 1、CC 0、OS 和 OV 将根据运算的结果进行设置。

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	-
$-2,147,483,648 \leq \text{结果} \leq -1$	0	1	0	-
$2,147,483,647 \geq \text{结果} \geq 1$	1	0	0	-
结果 = 2,147,483,648	0	1	1	1

举例

STL	解 释
L ID8	// 将数值装入累加器 1 中。
NEGD	// 对 32 位数求补码。
T MD10	// 将结果传送到 MD10。

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
NEGD 执行前	0101	1111	0110	0100	0101	1101	0011	1000
NEGD 执行后	1010	0000	1001	1011	1010	0010	1100	1000

3.12 NEGR 对浮点数（32 位，IEEE-FP）求反

格式

NEGR

指令说明

使用 NEGR（对 32 位 IEEE 浮点数求反）指令，可以对累加器 1 中的浮点数（32 位，IEEE-FP）求反。该指令将对累加器 1 中位 31（尾数的符号位）取反。结果保存在累加器 1 中。

状态字

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

举例

STL	解 释
L ID8	// 将数值装入累加器 1 中（例如：ID 8 = 1.5E+02）。
NEGR	// 将浮点数（32 位，IEEE FP）取反；结果保存到累加器 1 中。
T MD10	// 将结果传送到 MD10（例如：结果 = -1.5E+02）。

3.13 CAW 交换累加器 1 低字 (16 位) 中的字节顺序

格式

CAW

说明

使用 CAW 指令，可以反转累加器 1 低字中的字节顺序。结果保存在累加器 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:	-	-	-	-	-	-	-	-	-

举例

STL	解 释
L MW10	// 将 MW10 的数值装入累加器 1。
CAW	// 反转累加器 1 低字中的字节顺序。
T MW20	// 将结果传送到 MW20。

内 容	累加器 1 高字中的高字节	累加器 1 高字中的低字节	累加器 1 低字中的高字节	累加器 1 低字中的低字节
CAW 执行前	数值 A	数值 B	数值 C	数值 D
CAW 执行后	数值 A	数值 B	数值 D	数值 C

3.14 CAD 交换累加器 1（32 位）中的字节顺序

格式

CAD

说明

使用 CAD 指令，可以反转累加器 1 中的字节顺序。结果保存在累加器 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:	-	-	-	-	-	-	-	-	-

举例

STL	解 释
L MD10	// 将 MD10 的数值装入累加器 1。
CAD	// 反转累加器 1 中的字节顺序。
T MD20	// 将结果传送到 MD20。

内 容	累加器 1 高字中的高字节	累加器 1 高字中的低字节	累加器 1 低字中的高字节	累加器 1 低字中的低字节
CAD 执行前	数值 A	数值 B	数值 C	数值 D
CAD 执行后	数值 D	数值 C	数值 B	数值 A

3.15 RND 取整

格式

RND

说明

RND 指令（32 位 IEEE 浮点数转换为 32 位整数）将累加器 1 中的内容解释为一个 32 位 IEEE 浮点数（32 位，IEEE-FP）。该指令将 32 位 IEEE 浮点数转换成为一个 32 位整数（双整数），并将结果取整为最近的整数。如果被转换数字的小数部分位于奇数和偶数结果中间，则该指令选择偶数结果。如果超出允许范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置位为“1”。结果保存在累加器 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	-	-	-	-

举例

STL	解 释
L MD10	// 将浮点数装入累加器 1 中。
RND	// 将浮点数（32 位，IEEE FP）转换为整数，并将结果取整。
T MD20	// 将结果（双整数）传送到 MD20。

转换前的数值		转换后的数值
MD10 = "100.5"	=> RND =>	MD20 = "+100"
MD10 = "-100.5"	=> RND =>	MD20 = "-100"

3.16 TRUNC 截尾取整

格式

TRUNC

说明

TRUNC 指令（32 位 IEEE 浮点数转换为 32 位整数）将累加器 1 中的内容解释为一个 32 位 IEEE 浮点数。该指令将 32 位 IEEE 浮点数转换成为一个 32 位整数（双整数）。结果为所转换浮点数的整数部分（IEEE 取整方式“截尾取整”）。如果数值超出允许范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置位为“1”。结果保存在累加器 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	-	-	-	-

举例

STL	解 释
L MD10	// 将浮点数装入累加器 1 中。
TRUNC	// 将浮点数（32 位，IEEE-FP）转换为整数（32 位）并对结果取整。结果保存在累加器 1 中。
T MD20	// 将结果（双整数）传送到 MD20。

转换之前的数值		转换之后的数值
MD10 = "100.5"	=> TRUNC =>	MD20 = "+100"
MD10 = "-100.5"	=> TRUNC =>	MD20 = "-100"

3.17 RND+ 向上取整为双整数

格式

RND+

说明

RND+ 指令（32 位 IEEE 浮点数转换为 32 位整数）将累加器 1 中的内容解释为一个 32 位 IEEE 浮点数。该指令将 32 位 IEEE 浮点数转换成为一个 32 位整数（双整数），并将结果取整为大于或等于该浮点数的最小整数（IEEE 取整方式“向上取整”）。如果数值超出允许范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置位为“1”。结果保存在累加器 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	-	-	-	-

举例

STL	解 释
L MD10	// 将浮点数（32 位，IEEE-FP）装入累加器 1 中。
RND+	// 将浮点数（32 位，IEEE FP）转换为整数（32 位），并将结果取整。结果保存在累加器 1 中。
T MD20	// 将结果（双整数）传送到 MD20。

转换前的数值		转换后的数值
MD10 = "100.5"	=> RND+ =>	MD20 = "+101"
MD10 = "-100.5"	=> RND+ =>	MD20 = "-100"

3.18 RND- 向下取整为双整数

格式

RND-

说明

RND- 指令（32 位 IEEE 浮点数转换为 32 位整数）将累加器 1 中的内容解释为一个 32 位 IEEE 浮点数。该指令将 32 位 IEEE 浮点数转换为一个 32 位整数（双整数），并将结果取整为小于或等于该浮点数的最大整数（IEEE 取整方式“向下取整”）。如果数值超出允许范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置位为“1”。结果保存在累加器 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	-	-	-	-

举例

STL	解 释
L MD10	// 将浮点数装入累加器 1 中。
RND-	// 将浮点数（32 位，IEEE FP）转换为整数（32 位），并将结果取整。结果保存在累加器 1 中。
T MD20	// 将结果（双整数）传送到 MD20。

转换前的数值		转换后的数值
MD10 = "100.5"	=> RND- =>	MD20 = "+100"
MD10 = "-100.5"	=> RND- =>	MD20 = "-101"

4 计数器指令

4.1 计数器指令概述

说明

计数器是 STEP 7 编程语言的功能单元之一，用来计数。在 CPU 存储区中留有一块计数器区域。该存储区为每一计数器保留一个 16 位的字。语句表指令集提供了 256 个计数器。你所使用的 CPU 能提供多少个计数器，请参见相应 CPU 的技术数据。

计数器指令是访问计数器存储区的唯一功能。

通过以下计数器指令，可以在此范围内改变计数值：

- FR 使能计数器（任意）
- L 将当前计数器值装入累加器 1
- LC 将当前计数器值作为 BCD 码装入累加器 1
- R 计数器复位
- S 设置计数器预设值
- CU 加计数器
- CD 减计数器

4.2 FR 使能计数器（任意）

格式

FR <计数器>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	计数器、计数范围取决于 CPU

说明

当 RLO 从“0”变为“1”时，该指令将清出沿检测标志，该标志用于设置和选择所寻址计数器的加计数或减计数。设置计数器或正常计数时不需要使能计数器。这就意味着不管设置计数器初始值、加计数器或减计数器的 RLO 是否为“1”，在使能计数器后将不能再执行这些指令。

状态字

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

举例

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”变为“1”时，使能计数器 C3。

4.3 L 将当前计数器值装入累加器 1

格式

L <计数器>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	计数器、计数范围取决于 CPU

说明

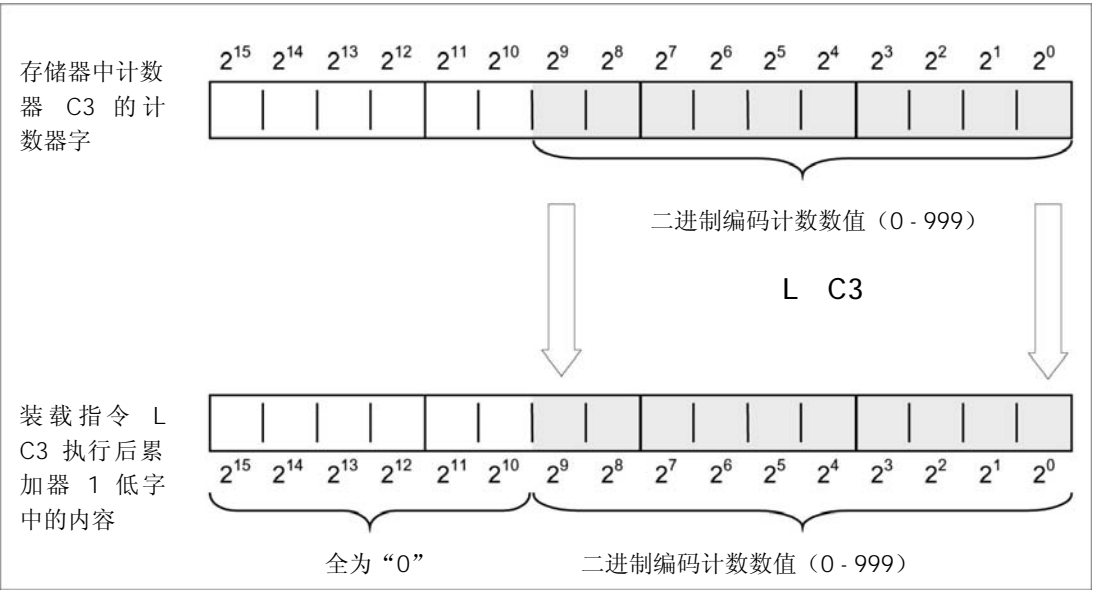
使用该指令，可以在累加器 1 的内容保存到累加器 2 中之后，将所寻址计数器的当前计数值以整数形式装入累加器 1 的低字中。

状态字

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

举例

STL	解 释
L C3	// 将计数器 C3 的计数值以二进制格式装入累加器 1 低字。



4.4 LC 将当前计数器值以 BCD 码形式装入累加器 1

格式

LC <计数器>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	计数器，计数范围取决于 CPU

说明

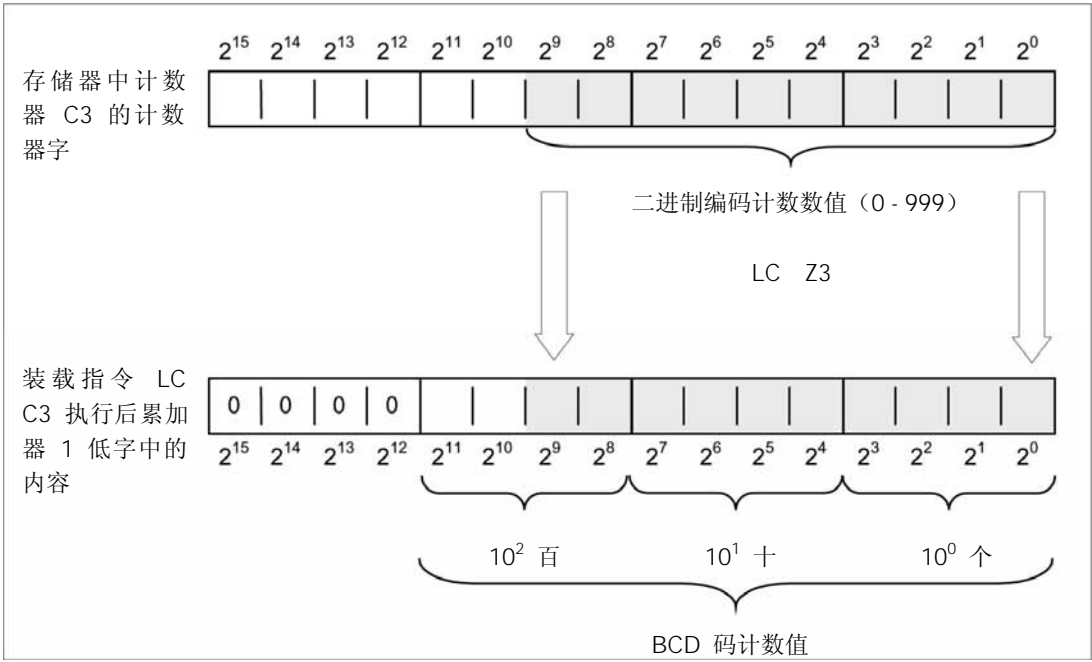
使用该指令，可以在累加器 1 的内容保存到累加器 2 中之后，将所寻址计数器的当前计数值以 BCD 形式装入累加器 1 中。

状态字

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

举例

STL	解 释
LC C3	// 将计数器 C3 的计数值以二进制编码的十进制格式（BCD）装入累加器 1 低字。



4.5 R 计数器复位

格式

R <计数器>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	计数器、计数范围取决于 CPU

说明

如果 RLO=1，该指令将对所寻址的计数器进行复位。

状态字

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

举例

STL	解 释
A I 2.3	// 检查输入 I 2.3 的信号状态。
R C3	// 当 RLO 从“0”变为“1”时，对计数器 C3 复位为“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	说 明
<计数器>	COUNTER	C	计数器、计数范围取决于 CPU

说明

当 RLO 从“0”变为“1”时，该指令将计数值从累加器 1 低字中装入计数器。
累加器 1 中的计数值必须是 0 – 999 之间的 BCD 数。

状态字

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

举例

STL	解 释
A I 2.3	// 检查输入 I 2.3 的信号状态。
L C#3	// 将计数数值“3”装入累加器 1 低字中。
S C1	// 当 RLO 从“0”变为“1”时，设置计数器 C1 的计数数值。

4.7 CU 加计数器

格式

CU <计数器>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	计数器、计数范围取决于 CPU

说明

当 RLO 从“0”变为“1”时，该指令将使所寻址计数器的计数值加“1”，并且计数小于“999”。当计数到达其上限“999”时，停止加计数。其它 RLO 状态跳变对该指令无影响；不会对溢出标志（OV）置位。

状态字

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

举例

STL	解 释
A I 2.1	// 如果在输入 I 2.1 有上升沿变化。
CU C3	// 当 RLO 从“0”变为“1”时，计数器 C3 加“1”。

4.8 CD 减计数器

格式

CD <计数器>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	计数器、计数范围取决于 CPU

说明

当 RLO 从“0”变为“1”时，该指令将使所寻址计数器的计数值减“1”，并且计数大于“0”。当计数到达其下限“0”时，停止计数。其它 RLO 状态跳变对该指令无影响；计数器不会出现负值。

状态字

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

举例

STL	解 释
L C#14	// 装载计数器预置值。
A I 0.1	// 检测到 I 0.1 的上升沿后预置计数器。
S C1	// 如果启用的话，装入计数器 1 预置值。
A I 0.0	// 每个 I 0.0 的上升沿进行一次减计数。
CD C1	// 根据输入 I 0.0 的信号状态，当 RLO 从“0”变为“1”时，计数器 C1 减“1”。
AN C1	// 使用 C1 位检测是否为“0”。
= Q 0	// 如果计数器 1 为“0”，则 Q 0.0 = 1。

5 数据块指令

5.1 数据块指令概述

说明

可以使用“打开数据块”（OPN）指令打开一个数据块作为共享数据块或背景数据块。程序本身可同时打开一个共享数据块和一个背景数据块。

下述数据块指令可供使用：

- OPN 打开数据块
- CDB 交换共享数据块和背景数据块
- L DBLG 将共享数据块的长度装入累加器 1 中
- L DBNO 将共享数据块的编号装入累加器 1 中
- L DILG 将背景数据块的长度装入累加器 1 中
- L DINO 将背景数据块的编号装入累加器 1 中

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 - 65535

指令说明

使用**打开数据块**指令，可以将数据块作为共享数据块或背景数据块打开。可以同时打开一个共享数据块和一个背景数据块。

状态字

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

举例

STL	解 释	
OPN DB10	// 打开数据块 DB10 作为共享数据块。	
L DBW35	// 将所打开数据块的数据字 35 装入累加器 1 低字中。	
T MW22	// 将累加器 1 低字中的内容传送到 MW22。	
OPN DI20	// 打开数据块 DB20 作为背景数据块。	
L DIB12	// 将所打开背景数据块的数据字节 12 装入累加器 1 低字中。	
T DBB37	// 将累加器 1 低字中的内容传送到已打开共享数据块的数据字节 37 中。	

5.3 CDB 交换共享数据块和背景数据块

格式

CDB

指令说明

使用该指令，可以交换共享数据块和背景数据块。该指令可以交换数据块寄存器。一个共享数据块可转换为一个背景数据块，反之亦然。

状态字

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

5.4 L DBLG 将共享数据块的长度装入累加器 1 中

格式

L DBLG

指令说明

使用**装载共享数据块长度**指令，可以在累加器 1 的内容保存到累加器 2 中之后，将共享数据块的长度装入累加器 1 中。

状态字

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

举例

STL	解 释
OPN DB10	// 打开数据块 DB10 作为共享数据块。
L DBLG	// 装入共享数据块的长度（DB10 的长度）。
L MD10	// 比较数据块的长度是否足够长。
<D	
JC ERRO	// 如果长度小于 MD10 中的数值，则跳转至 ERRO 跳转标号。

5.5 L DBNO 将共享数据块的编号装入累加器 1 中

格式

L DBNO

指令说明

使用**装载共享数据块编号**指令,可以在累加器 1 的内容保存到累加器 2 中之后,将共享数据块的编号装入累加器 1 的低字中。

状态字

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

5.6 L DILG 将背景数据块的长度装入累加器 1 中

格式

L DILG

指令说明

使用**装载背景数据块长度**指令,可以在累加器 1 的内容保存到累加器 2 中之后,将背景数据块的长度装入累加器 1 的低字中。

状态字

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

举例

STL	解 释
OPN DI20	// 打开数据块 DB20 作为背景数据块。
L DILG	// 装载背景数据块的长度 (DB20 的长度)。
L MW10	// 比较数据块的长度是否足够长。
<I	
JC	// 如果长度小于 MW10 中的数值,则跳转至 ERRO 跳转标号。

5.7 L DINO 将背景数据块的编号装入累加器 1 中

格式

L DINO

指令说明

使用**装载背景数据块编号**指令,可以在累加器 1 的内容保存到累加器 2 中之后,将背景数据块的编号装入累加器 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

说明

可以使用跳转指令来控制逻辑流，使其中断程序控制流而从程序的不同点重新开始扫描。你可以使用循环控制指令（LOOP），多次调用一个程序段。

跳转指令或循环控制指令的地址是一个标号。一个跳转标号最多有 4 个字符，第一个字符必须是字母。跳转标号后跟冒号“:”，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，跳转目的地总是从（不适用于 318-2）跳转指令中的布尔逻辑串开始。跳转目的地不能包括在逻辑串中。

可使用以下跳转指令无条件中断正常的程序逻辑流。

- JU 无条件跳转
- JL 跳转到标号

根据上一条指令产生的逻辑运算结果（RLO）进行跳转：

- JC 若 $RLO = 1$ ，则跳转
- JCN 若 $RLO = 0$ ，则跳转
- JCB 若 $RLO = 1$ 则跳转，同时对 BR 置位
- JNB 若 $RLO = 0$ 则跳转，同时对 BR 复位

根据状态字中的状态位的信号状态进行跳转：

- JBI 若 $BR = 1$ ，则跳转
- JNBI 若 $BR = 0$ ，则跳转
- JO 若 $OV = 1$ ，则跳转
- JOS 若 $OS = 1$ ，则跳转

根据计算结果进行跳转：

- JZ 若零，则跳转
- JN 若非零，则跳转
- JP 若正，则跳转
- JM 若负，则跳转
- JPZ 若大于或等于零，则跳转
- JNZ 若小于或等于零，则跳转
- JUO 若运算无效，则跳转

6.2 JU 无条件跳转

格式

JU <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

使用该指令，可以中断线性程序扫描，并跳转到一个跳转目标指令，该跳转与状态字的内容无关。程序扫描将从目标地址继续进行。跳转目标通过一个跳转标号来指定，可向前跳转和向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释
A I 1.0	
A I 1.2	
JC DELE	// 如果 RLO=1，则跳转到跳转标号 DELE。
L MB10	
INC 1	
T MB10	
JU FORW	// 无条件跳转到跳转标号 FORW。
DELE: L 0	
T MB10	
FORW: A I 2.1	// 在跳转到跳转标号 FORW 之后继续进行程序扫描。

6.3 JL 跳转到标号

格式

JL <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

使用该指令（通过跳转到列表进行跳转），可以进行多级跳转。最多有255个跳转目标，从该指令的下一行开始，直至该指令所跳转到的标号前一行结束。每个跳转目标包含一条无条件跳转指令（JU）。跳转目标的数量（0 – 255）存储在累加器 1 低字的低字节中。

只要累加器的内容小于 JL 指令和跳转标号之间跳转目标的数量，JL 指令就跳转到相应的一条 JU 指令。如果累加器 1 低字的低字节为“0”，则跳到第一条 JU 指令。如果累加器 1 低字的低字节为“1”，则跳到第二个 JU 指令，依此类推。如果跳转目标的数量太大，则 JL 指令跳转到目标列表中最后一个 JU 指令之后的第一条指令。

跳转目标列表必须包含 JU 指令，由其来负责在 JL 指令的地址区内进行相应的跳转。跳转列表中的任何其它指令都是非法的。

状态字

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

举例

STL	解 释	
L MB0	// 将跳转目标的数量装入累加器 1 低字低字节中	
JL LSTX	// 如果累加器 1 低字低字节中的内容大于 3，则跳转到 LSTX	
JU SEG0	// 如果累加器 1 低字低字节中的内容等于 0，则跳转到 SEG0	
JU SEG1	// 如果累加器 1 低字低字节中的内容等于 1，则跳转到 SEG1	
JU COMM	// 如果累加器 1 低字低字节中的内容等于 2，则跳转到 SEG2	
JU SEG3	// 如果累加器 1 低字低字节中的内容等于 3，则跳转到 SEG3	
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 若 RLO = 1，则跳转

格式

JC <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果 RLO=1，则该指令可以中断当前线性程序扫描，并跳转到一个跳转目标，在跳转目标继续进行线性程序扫描。跳转目标通过一个跳转标号来指定，可向前或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

如果 RLO=0，则不执行跳转。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

举例

STL	解 释
A I 1.0	
A I 1.2	
JC JOVR	// 如果 RLO=1，则跳转到 JOVR
L IW8	// 如果没有执行跳转，则继续执行程序扫描
T MW22	
JOVR: A I 2.1	// 在跳转到 JOVR 之后继续进行程序扫描。

6.5 JCN 若 RLO = 0，则跳转

格式

JCN <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果 RLO=0，则该指令可以中断当前线性程序扫描，并跳转到一个跳转目标，在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定，可向前或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

如果 RLO=1，则不执行跳转，并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

举例

STL	解 释
A I 1.0	
A I 1.2	
JCN JOVR	// 如果 RLO = 0，则跳转到 JOVR
L IW8	// 如果没有执行跳转，则继续执行程序扫描
T MW22	
JOVR: A I 2.1	// 在跳转到 JOVR 之后继续进行程序扫描

6.6 JCB 若 RLO = 1 则跳转，同时对 BR 置位

格式

JCB <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果 RLO=1，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组情况（一个、两个或三个字语句）。

如果 RLO=0，则不执行跳转。将 RLO 置为“1”，从下一条语句继续程序扫描。RLO 被拷贝到该指令的 BR 中，而与 RLO 无关。

状态字

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

举例

STL	解 释
A I 1.0	
A I 1.2	
JCB JOVR	// 如果 RLO=1，则跳转到 JOVR。将 RLO 位的内容复制到 BR 位
L IW8	// 如果没有执行跳转，则继续执行程序扫描
T MW22	
JOVR: A I 2.1	// 在跳转到 JOVR 之继续进行程序扫描

6.7 JNB 若 RLO = 0 则跳转，同时对 BR 复位

格式

JNB <跳转标号>

地 址	说 明
<跳转标号>	跳转目标符号名

说明

如果 RLO=0，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

如果 RLO=1，则不执行跳转。将 RLO 置为“1”，从下一条语句继续程序扫描。RLO 被拷贝到该指令的 BR 中，而与 RLO 无关。

状态字

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

举例

STL	解 释
A I 1.0	
A I 1.2	
JNB JOVR	// 如果 RLO=0，则跳转到 JOVR。将 RLO 位的内容复制到 BR 位
L IW8	// 如果没有执行跳转，则继续执行程序扫描
T MW22	
JOVR: A I 2.1	// 在跳转到 JOVR 之后继续进行程序扫描

6.8 JBI 若 BR = 1，则跳转

格式

JBI <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 BR=1，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

6.9 JNBI 若 BR = 0，则跳转

格式

JNBI <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 BR=0，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

6.10 JO 若 OV = 1，则跳转

格式

JO <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 OV=1，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。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进行检测。

状态字

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

举例

STL	解 释
L MW10	
L 3	
*I	// 将 MW10 的内容乘以“3”
JO OVER	// 如果结果超出最大范围（OV=1），则跳转
T MW10	// 如果没有跳转，则继续执行程序扫描
A M 4.0	
R M 4.0	
JU NEXT	
OVER: AN M 4.0	// 在跳转到 OVER 之后继续进行程序扫描
S M 4.0	
NEXT: NOP 0	// 在跳转到 NEXT 之后继续进行程序扫描

6.11 JOS 若 OS = 1，则跳转

格式

JOS <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 OS=1，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释	
L IW10		
L MW12		
*I		
L DBW25		
+I		
L MW14		
-I		
JOS OVER	// 如果三条指令中任何一条出现溢出（OS=1，见注释），则跳转	
T MW16	// 如果没有执行跳转，则继续执行程序扫描	
A M 4.0		
R M 4.0		
JU NEXT		
OVER: AN M 4.0	// 在跳转到 OVER 之后继续进行程序扫描	
S M 4.0		
NEXT: NOP 0	// 在跳转到 NEXT 之后继续进行程序扫描	

注释

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 若零，则跳转

格式

JZ <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 CC 1 = 0 并且 CC 0 = 0，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释
L MW10	
SRW 1	
JZ ZERO	// 如果已移出位 = 0，则跳转到 ZERO。
L MW2	// 如果没有执行跳转，则继续执行程序扫描。
INC 1	
T MW2	
JU NEXT	
ZERO: L MW4	// 在跳转到 ZERO 之后继续进行程序扫描。
INC 1	
T MW4	
NEXT: NOP 0	// 在跳转到 NEXT 之后继续进行程序扫描。

6.13 JN 若非零，则跳转

格式

JN <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果由状态位 CC 1 和 CC 0 指示的结果大于或小于零(CC 1=0/CC 0=1 或 CC 1=1/CC 0=0)，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释
L IW8	
L MW12	
XOW	
JN NOZE	// 如果累加器 1 低字的内容不等于“0”，则跳转到 NOZE。
AN M 4.0	// 如果没有执行跳转，则继续执行程序扫描。
S M 4.0	
JU NEXT	
NOZE: AN M 4.1	// 在跳转到 NOZE 之后继续进行程序扫描。
S M 4.1	
NEXT: NOP 0	// 在跳转到 NEXT 之后继续进行程序扫描。

6.14 JP 若正，则跳转

格式

JP <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 CC 1 = 1 并且 CC 0 = 0，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释	
L IW8		
L MW12		
-I		// 将 IW8 的内容减去 MW12 的内容。
JP POS		// 如果结果 >0（即，累加器 1 中的内容大于“0”），则跳转。
AN M 4.0		// 如果没有执行跳转，则继续执行程序扫描。
S M 4.0		
JU NEXT		
POS: AN M 4.1		// 在跳转到 POS 之后继续执行程序扫描。
S M 4.1		
NEXT: NOP 0		// 在跳转到 NEXT 之后继续执行程序扫描。

6.15 JM 若负，则跳转

格式

JM <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 $CC\ 1 = 0$ 并且 $CC\ 0 = 1$ ，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释
L IW8	
L MW12	
-I	// 将 IW8 的内容减去 MW12 的内容。
JM NEG	// 如果结果 < 0（即，累加器 1 中的内容小于“0”），则跳转。
AN M4.0	// 如果没有执行跳转，则继续执行程序扫描。
S M4.0	
JU NEXT	
NEG: AN M4.1	
S M4.1	
NEXT: NOP 0	// 在跳转到 NEG 之后继续进行程序扫描。
	// 在跳转到 NEXT 之后继续进行程序扫描。

6.16 JPZ 若大于或等于零，则跳转

格式

JPZ <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果由状态位 CC 1 和 CC 0 指示的结果大于或等于零(CC 1=0/CC 0=0 或 CC 1=1/CC 0=0)，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释
L IW8	
L MW12	
-I	// 将 IW8 的内容减去 MW12 的内容。
JPZ REG0	// 如果结果 ≥ 0 （即，累加器 1 中的内容 \geq “0”），则跳转。
AN M 4.0	// 如果没有执行跳转，则继续执行程序扫描。
S M 4.0	
JU NEXT	
REG0: AN M 4.1	// 在跳转到 REG0 之后继续进行程序扫描。
S M 4.1	
NEXT: NOP 0	// 在跳转到 NEXT 之后继续进行程序扫描。

6.17 JMZ 若小于或等于零，则跳转

格式

JMZ <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果由状态位 CC 1 和 CC 0 指示的结果小于或等于零(CC 1=0/CC 0=0 或 CC 1=0/CC 0=1)，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

举例

STL	解 释
L IW8	
L MW12	
-I	// 将 IW8 的内容减去 MW12 的内容。
JMZ RGE0	// 如果结果 ≤ 0（即，累加器 1 中的内容 ≤ “0”），则跳转。
AN M 4.0	// 如果没有执行跳转，则继续执行程序扫描。
S M 4.0	
JU NEXT	
RGE0: AN M 4.1	// 在跳转到 RGE0 之后继续进行程序扫描。
S M 4.1	
NEXT: NOP 0	// 在跳转到 NEXT 之后继续进行程序扫描。

6.18 JUO 若运算无效，则跳转

格式

JUO <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

如果状态位 CC 1 = 1 并且 CC 0 = 1，该指令将中断当前线性程序扫描，并跳转到一个跳转目标。在跳转目标处继续进行线性程序扫描。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

在以下情况下，状态位 CC 1 = 1 且 CC 0 = 1：

- 除数为 0 时
- 使用了非法指令
- 浮点数比较结果为无效数，即使用了无效格式。

状态字

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

举例

STL	解 释
L MD10	
L ID2	
/D	// 将 MD10 的内容除以 ID2 的内容。
JUO ERRO	// 如果被零除（即，ID2 = 0），则跳转。
T MD14	// 如果没有执行跳转，则继续执行程序扫描。
A M 4.0	
R M 4.0	
JU NEXT	
ERRO: AN M 4.0	// 在跳转到 ERRO 之后继续进行程序扫描。
S M 4.0	
NEXT: NOP 0	// 在跳转到 NEXT 之后继续进行程序扫描。

6.19 LOOP 循环控制

格式

LOOP <跳转标号>

地 址	说 明
<跳转标号>	跳转目标的符号名

说明

使用循环控制指令（累加器 1 低字中的值减“1”，当累加器 1 低字中的值不等于“0”时跳转），可以简化循环控制编程。在累加器 1 低字中有循环计数器。该指令可以跳转到指定跳转目标。只要累加器 1 低字中的值不为“0”就跳转，并在跳转目标处重新进行线性程序扫描。该指令将中断当前线性程序扫描，并跳转到一个跳转目标。跳转目标通过一个跳转标号来指定。可向前跳转或向后跳转。只能在一个程序块内执行跳转，即跳转指令和跳转目标必须位于同一个程序块内。跳转目标在该块内必须是唯一的。最大跳转距离为 -32768 或 +32767 个程序代码字。实际跳转的最大长度取决于程序中所使用语句的组合情况（一个、两个或三个字语句）。

状态字

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

因子为 5 的计算举例

STL	解 释		
L L#1	// 将整数常数（32 位）装入累加器 1。		
T MD20	// 将累加器 1 中的内容传送到 MD20（初始化）。		
L 5	// 将循环周期次数装入累加器 1 低字中。		
NEXT: T MW10	// 跳转标号 = 循环开始 / 将累加器 1 低字中的内容传送到循环计数器。		
L MD20			
* D			
T MD20	// 将 MD20 的当前内容乘以 MB10 的当前内容。		
L MW10	// 将相乘结果传送到 MD20。		
LOOP NEXT	// 将循环计数器的内容装入累加器 1 中。		
	// 如果累加器 1 低字中的内容大于“0”，则累加器 1 中的内容减“1”，并跳转到 NEXT 跳转标号。		
L MW24			
L 200	// 循环结束之后重新进行程序扫描。		
>I			

7 整数算术运算指令

7.1 整数算术运算指令概述

说明

算术运算指令针对累加器 1 和 2 的内容。对于两个累加器的CPU，累加器 2 的内容保持不变。

如果 CPU 具有 4 个累加器，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。而累加器 4 中原有的内容保持不变。

使用整数算术运算指令，可以进行以下两个整数（16 位和 32 位）之间的运算：

- +I 将累加器1和累加器2中的内容相加（16 位整数运算）
- -I 将累加器2中的内容减去累加器1中的内容（16 位整数运算）
- *I 将累加器1和累加器2中的内容相乘（16 位整数运算）
- /I 将累加器2中的内容除以累加器1中的内容（16 位整数运算）
- + 加整型常数（16 位，32 位）
- +D 将累加器1和累加器2中的内容相加（32 位双整数运算）
- -D 将累加器2中的内容减去累加器1中的内容（32 位双整数运算）
- *D 将累加器1和累加器2中的内容相乘（32 位双整数运算）
- /D 将累加器2中的内容除以累加器1中的内容（32 位双整数运算）
- MOD 双整数除法的余数（32位）

7.2 判断整数算术运算指令后状态字的位

说明

整数算术运算指令可以影响以下状态字中的位：CC1 和 CC0，OV 和 OS。

下表所示整数（16 位和 32 位）运算指令结果的状态字中各位的信号状态：

有效的结果范围	CC 1	CC 0	OV	OS
0	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	*

* OS 位不受指令结果的影响。

无效的结果范围	CC 1	CC 0	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
上溢（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
上溢（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

操 作	CC 1	CC 0	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 将累加器1和累加器2中的内容相加(16位整数运算)

格式

+I

说明

使用 16 位整数加法指令（+I），可以将累加器 1 低字中的内容与累加器 2 低字中的内容相加，结果保存在累加器 1 低字中。累加器 1 和累加器 2 低字中的内容解释为 16 位整数。指令的执行与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。在出现上溢/下溢时，指令产生一个 16 位整数，而不是 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 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。而累加器 4 的内容保持不变。

请参见“判断整数算术运算指令后状态字的位”。

状态字

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

状态位生成	CC 1	CC 0	OV	OS
和 = 0	0	0	0	-
- 32768 ≤ 和 < 0	0	1	0	-
32767 ≥ 和 > 0	1	0	0	-
和 = -65536	0	0	1	1
65534 ≥ 和 > 32767	0	1	1	1
-65535 ≤ 和 < -32768	1	0	1	1

举例

STL	解 释
L IW10	// 将 IW10 的数值装入累加器 1 低字。
L MW14	// 将累加器 1 低字中的内容装入累加器 2 低字中。将存储字 MW14 的值装入累加器 1 低字。
+I	// 将累加器 2 低字和累加器 1 低字中的内容相加；结果保存到累加器 1 低字中。
T DB1.DBW25	// 累加器 1 低字的内容（结果）被传送到 DB1 的 DBW25。

7.4 -I 将累加器2的内容减累加器1的内容(16位整数运算)

格式

-I

说明

使用 16 位整数减法指令 (-I)，可以将累加器 2 低字中的内容减去累加器 1 低字中的内容，结果保存在累加器 1 低字中。累加器 1 和累加器 2 低字中的内容解释为 16 位整数。指令的执行与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。在出现上溢/下溢时，指令产生一个 16 位整数，而不是 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 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。而累加器 4 的内容保持不变。

请参见“判断整数算术运算指令后状态字的位”。

状态字

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

状态位生成	CC 1	CC 0	OV	OS
差 = 0	0	0	0	-
$-32768 \leq \text{差} < 0$	0	1	0	-
$32767 \geq \text{差} > 0$	1	0	0	-
$65535 \geq \text{差} > 32767$	0	1	1	1
$-65535 \leq \text{差} < -32768$	1	0	1	1

举例

STL	解 释
L IW10	// 将 IW10 的数值装入累加器 1 低字节。
L MW14	// 将累加器 1 低字中的内容装入累加器 2 低字中。将 MW14 的值装入累加器 1 低字。
-I	// 将累加器 2 低字中的内容减去累加器 1 低字中的内容；结果保存到累加器 1 低字中。
T DB1.DBW25	// 累加器 1 低字的内容（结果）被传送到 DB1 的 DBW25。

7.5 *I 将累加器1和累加器2中的内容相乘(16位整数运算)

格式

*I

说明

使用 16 位整数乘法指令(*I)，可以将累加器 2 低字中的内容乘以累加器 1 低字中的内容。累加器 1 和累加器 2 低字中的内容解释为 16 位整数。结果作为一个 32 位整数保存在累加器 1 中。如果状态字位 OV1 = 1 且 OS = 1，则结果超出 16 位整数的范围。

指令的执行与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。

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 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。

请参见“判断整数算术运算指令后状态字的位”。

状态字

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

状态位生成	CC 1	CC 0	OV	OS
积 = 0	0	0	0	-
- 32768 ≤ 积 < 0	0	1	0	-
32767 ≥ 积 > 0	1	0	0	-
1073741824 ≥ 积 > 32767	1	0	1	1
-1073709056 ≤ 积 < -32768	0	1	1	1

举例

STL	解 释
L IW10	// 将 IW10 的数值装入累加器 1 低字节。
L MW14	// 将累加器 1 低字中的内容装入累加器 2 低字中。将 MW14 的内容装入累加器 1 低字。
*I	// 将累加器 2 低字和累加器 1 低字中的内容相乘；结果保存到累加器 1 中。
T DB1.DBD25	// 累加器 1 的内容（结果）被传送到 DB1 的 DBD25。

7.6 // 将累加器2的内容除以累加器1的内容(16位整数运算)

格式

//

说明

使用 16 位整数除法指令 (/I)，可以将累加器 2 低字中的内容除以累加器 1 低字中的内容。累加器 1 和累加器 2 低字中的内容解释为 16 位整数。结果保存在累加器 1 中，并由两个 16 位整数组成：商和余数。商保存在累加器 1 低字中，余数保存在累加器 1 高字中。指令的执行与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。

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 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。而累加器 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	-	-	-	-

状态位生成	CC 1	CC 0	OV	OS
商 = 0	0	0	0	-
$-32768 \leq \text{商} < 0$	0	1	0	-
$32767 \geq \text{商} > 0$	1	0	0	-
商 = 32768	1	0	1	1
被零除	1	1	1	1

举例

STL	解 释
L IW10	// 将 IW10 的数值装入累加器 1 低字节。
L MW14	// 将累加器 1 低字中的内容装入累加器 2 低字中。将 MW14 的值装入累加器 1 低字。
//	// 将累加器 2 低字中的内容除以累加器 1 低字中的内容；结果保存到累加器 1 中：商数存储到累加器 1 低字中，余数存储到累加器 1 高字中。
T MD20	// 累加器 1 的内容（结果）被传送到 MD20。

例如：13 被 4 除

指令执行之前累加器 2 低字中的内容 (IW10)："13"

指令执行之前累加器 1 低字中的内容 (MW14)："4"

指令 // (累加器 2 低字中的内容 / 累加器 1 低字中的内容)："13/4"

指令执行之后累加器 1 低字中的内容 (商)："3"

指令执行之后累加器 1 高字中的内容 (余数)："1"

7.7 + 加整型常数（16 位，32 位）

格式

+ <整型常数>

地 址	数据类型	说 明
<整型常数>	（16 位或 32 位整数）	要加的常数

说明

使用加整型常数指令（+ <整型常数>），可以对累加器 1 中的内容加上一个整型常数，结果保存在累加器 1 中。指令的执行与状态字位无关，而且对状态字位没有影响。

+ <16 位整型常数>：对累加器 1 低字中的内容加上一个 16 位整型常数（范围 -32768 - +32767），结果保存在累加器 1 低字中。

+ <32 位整型常数>：对累加器 1 中的内容加上一个 32 位整型常数（范围 -2,147,483,648 - +2,147,483,647），结果保存在累加器 1 中。

状态字

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

举例 1

STL	解 释
L IW10	// 将 IW10 的数值装入累加器 1 低字节。
L MW14	// 将累加器 1 低字中的内容装入累加器 2 低字中。将 MW14 的值装入累加器 1 低字。
+I	// 将累加器 2 低字和累加器 1 低字中的内容相加；结果保存到累加器 1 低字中。
+ 25	// 将累加器 1 低字的内容加“25”；结果保存到累加器 1 低字中。
T DB1.DBW25	// 将累加器 1 低字的内容（结果）传送到 DB1 的 DBW25 中。

举例 2

STL	解 释
L IW12	
L IW14	
+ 100	// 将累加器 1 低字的内容加“100”；结果保存到累加器 1 低字中。
>I	// 如果累加器 2 中的内容大于累加器 1 中的内容，或 $IW12 > (IW14 + 100)$ 。
JC NEXT	// 则条件跳转到跳转标号 NEXT。

举例 3

STL	解 释
L MD20	
L MD24	
+D	// 将累加器 1 和累加器 2 中的内容相加；结果保存到累加器 1 中。
+ L#-200	// 将累加器 1 中的内容和“-200”相加；结果保存到累加器 1 中。
T MD28	

7.8 +D 将累加器1和累加器2的内容相加(32位双整数运算)

格式

+D

说明

使用 32 位双整数加法指令 (+D)，可以将累加器 1 中的内容与累加器 2 中的内容相加，结果保存在累加器 1 中。累加器 1 和累加器 2 中的内容解释为 32 位整数。执行指令与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。

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 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。而累加器 4 的内容保持不变。

请参见“判断整数算术运算指令后状态字的位”。

状态字

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

状态位生成	CC 1	CC 0	OV	OS
和 = 0	0	0	0	-
-2147483648 ≤ 和 < 0	0	1	0	-
2147483647 ≥ 和 > 0	1	0	0	-
和 = -4294967296	0	0	1	1
4294967294 ≥ 和 > 2147483647	0	1	1	1
-4294967295 ≤ 和 < -2147483648	1	0	1	1

举例

STL	解 释
L ID10	// 将 ID10 的数值装入累加器 1。
L MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
+D	// 将累加器 2 中的内容和累加器 1 中的内容相加；结果保存到累加器 1 中。
T DB1.DBD25	// 累加器 1 的内容（结果）被传送到 DB1 的 DBD25。

7.9 -D 累加器2的内容减累加器1的内容(32位双整数运算)

格式

-D

说明

使用 32 位双整数减法指令（-D），可以将累加器 2 中的内容减去累加器 1 中的内容，结果保存在累加器 1 中。累加器 1 和累加器 2 中的内容解释为 32 位整数。执行指令与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果设置。

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 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。而累加器 4 的内容保持不变。

请参见“判断整数算术运算指令后状态字的位”。

状态字

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

状态位生成	CC 1	CC 0	OV	OS
差 = 0	0	0	0	-
$-2147483648 \leq \text{差} < 0$	0	1	0	-
$2147483647 \geq \text{差} > 0$	1	0	0	-
$4294967295 \geq \text{差} > 2147483647$	0	1	1	1
$-4294967295 \leq \text{差} < -2147483648$	1	0	1	1

举例

STL	解 释
L ID10	// 将 ID10 的数值装入累加器 1。
L MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
-D	// 将累加器 2 中的内容减去累加器 1 中的内容；结果保存到累加器 1 中。
T DB1.DBD25	// 累加器 1 的内容（结果）被传送到 DB1 的 DBD25。

7.10 *D 将累加器1和累加器2的内容相乘(32位双整数运算)

格式

*D

说明

使用 32 位双整数乘法指令（*D），可以将累加器 2 中的内容乘以累加器 1 中的内容。累加器 1 和累加器 2 中的内容解释为 32 位整数。结果作为一个 32 位整数保存在累加器 1 中。如果状态字位 OV1 = 1 且 OS = 1，则结果超出 32 位整数的范围。

执行指令与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果设置。

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 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。而累加器 4 的内容保持不变。

请参见“判断整数算术运算指令后状态字的位”。

状态字

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

状态位生成	CC 1	CC 0	OV	OS
积 = 0	0	0	0	-
-2147483648 ≤ 积 < 0	0	1	0	-
2147483647 ≥ 积 > 0	1	0	0	-
积 > 2147483647	1	0	1	1
积 < -2147483648	0	1	1	1

举例

STL	解 释
L ID10	// 将 ID10 的数值装入累加器 1。
L MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
*D	// 将累加器 2 中的内容和累加器 1 中的内容相乘；结果保存到累加器 1 中。
T DB1.DBD25	// 累加器 1 的内容（结果）被传送到 DB1 的 DBD25。

7.11 /D 累加器2的内容除以累加器1的内容(32位双整数运算)

格式

/D

说明

使用 32 位双整数除法指令 (/D)，可以将累加器 2 中的内容除以累加器 1 中的内容。累加器 1 和累加器 2 中的内容解释为 32 位整数。结果保存在累加器 1 中。结果只给出了商，没有余数。（使用指令 MOD，可以获得余数）。

执行指令与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果设置。

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 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。而累加器 4 的内容保持不变。

请参见“判断整数算术运算指令后状态字的位”。

状态字

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

状态位生成	CC 1	CC 0	OV	OS
商 = 0	0	0	0	-
$-2147483648 \leq \text{商} < 0$	0	1	0	-
$2147483647 \geq \text{商} > 0$	1	0	0	-
商 = 2147483648	1	0	1	1
被零除	1	1	1	1

举例

STL	解 释
L ID10	// 将 ID10 的数值装入累加器 1。
L MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
/D	// 将累加器 2 中的内容除以累加器 1 中的内容；结果（商）保存到累加器 1 中。
T MD20	// 累加器 1 的内容（结果）被传送到 MD20。

例如：13 被 4 除

指令执行之前累加器 2 中的内容 (ID10)：“13”

指令执行之前累加器 1 中的内容 (MD14)：“4”

指令 /D（累加器 2 中的内容 / 累加器 1 中的内容）：“13/4”

指令执行之后累加器 1 中的内容（商）：“3”

7.12 MOD 双整数除法的余数（32位）

格式

MOD

说明

使用 MOD 指令（32 位双整数除法的余数），可以将累加器 2 中的内容除以累加器 1 中的内容。累加器 1 和累加器 2 中的内容解释为 32 位整数。结果保存在累加器 1 中。结果只给出了余数，没有商。（使用指令 /D，可以获得商）。执行指令与 RLO 无关，而且对 RLO 没有影响。状态字位 CC 1、CC 0、OS 和 OV 根据运算结果设置。

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 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。而累加器 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	-	-	-	-

状态位生成	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

举例

STL	解 释
L ID10	// 将 ID10 的数值装入累加器 1。
L MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
MOD	// 将累加器 2 中的内容除以累加器 1 中的内容；结果（余数）保存到累加器 1 中。
T MD20	// 累加器 1 的内容（结果）被传送到 MD20。

例如：13 被 4 除

指令执行之前累加器 2 中的内容（ID10）： "13"

指令执行之前累加器 1 中的内容（MD14）： "4"

指令 MOD（累加器 2 中的内容 / 累加器 1 中的内容）： "13/4"

指令执行之后累加器 1 中的内容（余数）： "1"

8 浮点数运算指令

8.1 浮点数运算指令概述

说明

算术运算指令针对累加器 1 和 2 的内容。其结果保存在累加器 1 中。对于两个 CPU 的累加器，累加器 2 的内容保持不变。

如果 CPU 具有 4 个累加器，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。而累加器 4 中原有的内容保持不变。

IEEE 32 位浮点数属于实数“REAL”数据类型。

应用浮点算术运算指令，可以对于两个 32 位标准 IEEE 浮点数完成以下算术运算：

- +R 将累加器 1 和累加器 2 中的内容相加
- -R 将累加器 2 中的内容减去累加器 1 中的内容
- *R 将累加器 1 和累加器 2 中的内容相乘
- /R 将累加器 2 中的内容除以累加器 1 中的内容

应用浮点算术运算指令，可以对于一个 32 位 IEEE 浮点数完成以下算术运算：

- ABS 取绝对值
- SQR 平方运算
- SQRT 开方运算
- EXP 指数运算
- LN 自然对数运算
- SIN 正弦运算
- COS 余弦运算
- TAN 正切运算
- ASIN 反正弦运算
- ACOS 反余弦运算
- ATAN 反正切运算

8.2 判断浮点数运算指令后状态字的位

说明

基本算术类型可以影响以下状态字中的位：CC 1 和 CC 0，OV 和 OS。

下表所示为浮点数（32 位）运算指令结果的状态字中各位的信号状态：

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

* OS 位不受指令结果的影响。

无效的结果范围	CC 1	CC 0	OV	OS
下溢 $-1.175494\text{E}-38 < \text{结果} < -1.401298\text{E}-45$ （负数）	0	0	1	1
下溢 $+1.401298\text{E}-45 < \text{结果} < +1.175494\text{E}-38$ （正数）	0	0	1	1
上溢 $\text{结果} < -3.402823\text{E}+38$ （负数）	0	1	1	1
上溢 $\text{结果} > 3.402823\text{E}+38$ （正数）	1	0	1	1
非有效浮点数或非法指令 （输入值超出有效范围）	1	1	1	1

8.3 浮点数运算指令：基本指令

8.3.1 +R 将累加器1和累加器2中的内容相加(32位浮点数，IEEE-FP)

格式

+R

指令说明

使用 32 位 IEEE 浮点数加法指令 (+R)，可以将累加器 1 中的内容与累加器 2 中的内容相加，结果保存在累加器 1 中。累加器 1 和累加器 2 中的内容解释为 32 位 IEEE 浮点数。执行指令与 RLO 无关，而且对 RLO 没有影响。状态位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。

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 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。而累加器 4 的内容保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-infinite（无穷小）	0	1	1	1	上溢
-qNaN	1	1	1	1	

状态字

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

举例

STL	解 释	
OPN	DB10	
L	ID10	// 将 ID10 的数值装入累加器 1。
L	MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
+R		// 将累加器 2 中的内容和累加器 1 中的内容相加；结果保存到累加器 1 中。
T	DBD25	// 累加器 1 的内容（结果）被传送到 DB10 的 DBD25。

8.3.2 -R 将累加器2中的内容减去累加器1中的内容(32位浮点数 ,IEEE-FP)

格式

-R

说明

使用 32 位 IEEE 浮点数减法指令（-R），可以将累加器 2 中的内容减去累加器 1 中的内容，结果保存在累加器 1 中。累加器 1 和累加器 2 中的内容解释为 32 位 IEEE 浮点数。执行指令与 RLO 无关，而且对 RLO 没有影响。状态位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。

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 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。而累加器 4 的内容保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-infinite（无穷小）	0	1	1	1	上溢
-qNaN	1	1	1	1	

状态字

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

举例

STL	解 释	
OPN	DB10	
L	ID10	// 将 ID10 的数值装入累加器 1。
L	MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
-R		// 将累加器 2 中的内容减去累加器 1 中的内容；结果保存到累加器 1 中。
T	DBD25	// 累加器 1 的内容（结果）被传送到 DB10 的 DBD25。

8.3.3 *R 将累加器1和累加器2中的内容相乘(32位浮点数，IEEE-FP)

格式

*R

指令说明

使用 32 位 IEEE 浮点数乘法指令（*R），可以将累加器 2 中的内容乘以累加器 1 中的内容。累加器 1 和累加器 2 中的内容解释为 32 位 IEEE 浮点数。结果作为一个 32 位 IEEE 浮点数保存在累加器 1 中。执行指令与 RLO 无关，而且对 RLO 没有影响。状态位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。

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 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。而累加器 4 的内容保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-infinite（无穷小）	0	1	1	1	上溢
-qNaN	1	1	1	1	

状态字

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

举例

STL	解 释
OPN DB10	
L ID10	// 将 ID10 的数值装入累加器 1。
L MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
*R	// 将累加器 2 中的内容和累加器 1 中的内容相乘；结果保存到累加器 1 中。
T DBD25	// 累加器 1 的内容（结果）被传送到 DB10 的 DBD25。

8.3.4 /R 累加器2的内容除以累加器1的内容(32位浮点数，IEEE-FP)

格式

/R

指令说明

使用 32 位 IEEE 浮点数除法指令 (/R)，可以将累加器 2 中的内容除以累加器 1 中的内容。累加器 1 和累加器 2 中的内容解释为 32 位 IEEE 浮点数。结果作为一个 32 位 IEEE 浮点数保存在累加器 1 中。执行指令与 RLO 无关，而且对 RLO 没有影响。状态位 CC 1、CC 0、OS 和 OV 根据运算结果进行设置。

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 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。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-infinite（无穷小）	0	1	1	1	上溢
-qNaN	1	1	1	1	

状态字

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

举例

STL	解 释
OPN DB10	
L ID10	// 将 ID10 的数值装入累加器 1。
L MD14	// 将累加器 1 中的内容装入累加器 2 中。将 MD14 的值装入累加器 1 中。
/R	// 将累加器 2 中的内容除以累加器 1 中的内容；结果保存到累加器 1 中。
T DBD20	// 累加器 1 的内容（结果）被传送到 DB10 的 DBD20。

8.3.5 ABS 取绝对值（32 位浮点数，IEEE-FP）

格式

ABS

说明

使用 ABS（对 32 位 IEEE 浮点数取绝对值）指令，可以对累加器 1 中的浮点数（32 位，IEEE-FP）取绝对值。其结果保存在累加器 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:	-	-	-	-	-	-	-	-	-

举例

STL	解 释
L ID8	// 将数值装入累加器 1 中（例如：ID8 = -1.5E+02）。
ABS	// 取绝对值；结果保存到累加器 1 中。
T MD10	// 将结果传送到 MD10（例如：结果 = 1.5E+02）。

8.4 浮点数运算指令：扩展指令

8.4.1 SQR 平方运算（32 位）

格式

SQR

指令说明

使用 SQR（对 32 位 IEEE 浮点数求平方）指令，可以对累加器 1 中的浮点数（32 位，IEEE-FP）进行平方运算。其结果保存在累加器 1 中。该指令的执行影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释
OPN DB17	// 打开数据块 DB17。
L DBD0	// 将 DBD0 的值装入累加器 1 中。（该值必须为浮点数格式）。
SQR	// 在累加器 1 中求浮点数（32 位，IEEE FP）的平方。结果保存到累加器 1 中。
AN OV	// 扫描状态字中的 OV 位是否为“0”。
JC OK	// 如果在 SQR 执行过程中没有出现错误，则跳转到 OK 跳转标号。
BEU	// 如果在 SQR 执行过程中出现错误，则无条件结束。
OK: T DBD4	// 将累加器 1 中的内容（结果）传送到数据双字 DBD4。

8.4.2 Sqrt 开方运算 (32 位)

格式

Sqrt

指令说明

使用 Sqrt (对 32 位 IEEE 浮点数求平方根) 指令, 可以对累加器 1 中的浮点数 (32 位, IEEE-FP) 求平方根。其结果保存在累加器 1 中。输入值必须大于或等于 “0”。结果为正值。唯一例外的是 “-0” 的平方根为 “-0”。该指令会影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及累加器 3 和累加器 4 的内容, 对于具有 4 个累加器的 CPU) 保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite (无穷大)	1	0	1	1	上溢
+normalized (规格化)	1	0	0	-	
+denormalized (反向规格化)	0	0	1	1	下溢
+zero (零)	0	0	0	-	
-zero (零)	0	0	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释
L MD10	// 将 MD10 的值装入累加器 1 中。(该值必须为浮点数格式)。
Sqrt	// 在累加器 1 中求浮点数(32 位,IEEE FP)的平方根。结果保存到累加器 1 中。
AN OV	// 扫描状态字中的 OV 位是否为 “0”。
JC OK	// 如果在 Sqrt 执行过程中没有出现错误, 则跳转到 OK 跳转标号。
BEU	// 如果在 Sqrt 执行过程中出现错误, 则无条件结束。
OK: T MD20	// 将累加器 1 中的内容 (结果) 传送到 MD20。

8.4.3 EXP 指数运算（32 位）

格式

EXP

指令说明

使用 EXP（对 32 位 IEEE 浮点数求指数）指令，可以对累加器 1 中的浮点数（32 位，IEEE-FP）求指数（以 e 为底）。其结果保存在累加器 1 中。该指令的执行影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释		
L MD10	// 将 MD10 的值装入累加器 1 中。（该值必须是浮点数格式）。		
EXP	// 在累加器 1 中求浮点数（32 位，IEEE FP）的指数（以 e 为底数）。结果保存到累加器 1 中。		
AN OV	// 扫描状态字中的 OV 位是否为“0”。		
JC OK	// 如果在 EXP 执行过程中没有出现错误，则跳转到 OK 跳转标号。		
BEU	// 如果在 EXP 执行过程中出现错误，则无条件结束。		
OK: T MD20	// 将累加器 1 中的内容（结果）传送到 MD20。		

8.4.4 LN 自然对数运算（32 位）

格式

LN

指令说明

使用 LN（对 32 位 IEEE 浮点数求自然对数）指令，可以对累加器 1 中的浮点数（32 位，IEEE-FP）求自然对数。其结果保存在累加器 1 中。输入值必须大于“0”。该指令会影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-infinite（无穷小）	0	1	1	1	上溢
-qNaN	1	1	1	1	

举例

STL	解 释
L MD10	// 将 MD10 的值装入累加器 1 中。（该值必须是浮点数格式）。
LN	// 在累加器 1 中求浮点数(32 位，IEEE FP)的自然对数。结果保存到累加器 1 中。
AN OV	// 扫描状态字中的 OV 位是否为“0”。
JC OK	// 如果在指令执行过程中没有出现错误，则跳转到 OK 跳转标号。
BEU	// 如果在指令执行过程中出现错误，则无条件结束。
OK: T MD20	// 将累加器 1 中的内容（结果）传送到 MD20。

8.4.5 SIN 正弦运算（32 位）

格式

SIN

指令说明

使用 SIN（对 32 位 IEEE 浮点数求正弦）指令，可以计算弧度(角度值)的正弦。角度必须在累加器 1 中以浮点数表示。其结果保存在累加器 1 中。该指令的执行影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	上溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释
L MD10	// 将 MD10 的值装入累加器 1 中。（该值必须是浮点数格式）。
SIN	// 在累加器 1 中计算浮点数（32 位，IEEE FP）的正弦值。结果保存到累加器 1 中。
T MD20	// 将累加器 1 中的内容（结果）传送到 MD20。

8.4.6 COS 余弦运算（32 位）

格式

COS

指令说明

使用 COS（对 32 位 IEEE 浮点数求余弦）指令，可以计算弧度(角度值)的余弦。角度必须在累加器 1 中以浮点数表示。其结果保存在累加器 1 中。该指令的执行影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	上溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释
L MD10	// 将 MD10 的值装入累加器 1 中。（该值必须是浮点数格式）。
COS	// 在累加器 1 中计算浮点数（32 位，IEEE FP）的余弦值。结果保存到累加器 1 中。
T MD20	// 将累加器 1 中的内容（结果）传送到 MD20。

8.4.7 TAN 正切运算（32 位）

格式

TAN

指令说明

使用 TAN（对 32 位 IEEE 浮点数求正切）指令，可以计算弧度(角度值)的正切。角度必须在累加器 1 中以浮点数表示。其结果保存在累加器 1 中。该指令的执行影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+infinite（无穷大）	1	0	1	1	上溢
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	下溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-infinite（无穷小）	0	1	1	1	上溢
-qNaN	1	1	1	1	

举例

STL	解 释
L MD10	// 将 MD10 的值装入累加器 1 中。（该值必须是浮点数格式）。
TAN	// 在累加器 1 中计算浮点数（32 位，IEEE FP）的正切值。结果保存到累加器 1 中。
AN OV	// 扫描状态字中的 OV 位是否为“0”。
JC OK	// 如果在 TAN 执行过程中没有出现错误，则跳转到 OK 跳转标号。
BEU	// 如果在 TAN 执行过程中出现错误，则无条件结束。
OK: T MD20	// 将累加器 1 中的内容（结果）传送到 MD20。

8.4.8 ASIN 反正弦运算（32 位）

格式

ASIN

指令说明

使用 ASIN（对 32 位 IEEE 浮点数求反正弦）指令，可以计算累加器 1 中浮点数的反正弦值。输入值的允许范围：

$-1 \leq \text{输入值} \leq +1$

结果是以弧度为单位的角。其值范围为：

$-\pi/2 \leq \text{反正弦（累加器 1 中的内容）} \leq +\pi/2$ ，其中 $\pi = 3.14159...$

该指令会影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	上溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释	
L MD10	// 将 MD10 的值装入累加器 1 中。（该值必须是浮点数格式）。	
ASIN	// 在累加器 1 中计算浮点数(32 位, IEEE FP)的反正弦值。结果保存到累加器 1 中。	
AN OV	// 扫描状态字中的 OV 位是否为“0”。	
JC OK	// 如果在 ASIN 执行过程中没有出现错误，则跳转到 OK 跳转标号。	
BEU	// 如果在 ASIN 执行过程中出现错误，则无条件结束。	
OK: T MD20	// 将累加器 1 中的内容（结果）传送到 MD20。	

8.4.9 ACOS 反余弦运算（32 位）

格式

ACOS

指令说明

使用 ACOS（对 32 位 IEEE 浮点数求反余弦）指令，可以计算累加器 1 中浮点数的反余弦值。输入值的允许范围：

$$-1 \leq \text{输入值} \leq +1$$

结果是以弧度为单位的角。其值范围为：

$$0 \leq \text{反余弦（累加器 1 中的内容）} \leq \pi, \text{ 其中 } \pi = 3.14159\dots$$

该指令会影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容（以及累加器 3 和累加器 4 的内容，对于具有 4 个累加器的 CPU）保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+normalized（规格化）	1	0	0	-	
+denormalized（反向规格化）	0	0	1	1	上溢
+zero（零）	0	0	0	-	
-zero（零）	0	0	0	-	
-denormalized（反向规格化）	0	0	1	1	下溢
-normalized（规格化）	0	1	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释
L MD10	// 将 MD10 的值装入累加器 1 中。（该值必须是浮点数格式）。
ACOS	// 在累加器 1 中计算浮点数(32 位, IEEE FP)的反余弦值。结果保存到累加器 1 中。
AN OV	// 扫描状态字中的 OV 位是否为“0”。
JC OK	// 如果在 ACOS 执行过程中没有出现错误，则跳转到 OK 跳转标号。
BEU	// 如果在 ACOS 执行过程中出现错误，则无条件结束。
OK: T MD20	// 将累加器 1 中的内容（结果）传送到 MD20。

8.4.10 ATAN 反正切运算 (32 位)

格式

ATAN

指令说明

使用 ATAN (对 32 位 IEEE 浮点数求反正切) 指令, 可以计算累加器 1 中浮点数的反正切值。结果是以弧度为单位的角度。其值范围为:

$$-\pi/2 \leq \text{反正切(累加器 1 中的内容)} \leq +\pi/2, \text{ 其中 } \pi = 3.14159...$$

该指令会影响 CC 1、CC 0、OV 和 OS 状态字位。

累加器 2 的内容 (以及累加器 3 和累加器 4 的内容, 对于具有 4 个累加器的 CPU) 保持不变。

结果

累加器 1 中的结果	CC 1	CC 0	OV	OS	备 注
+qNaN	1	1	1	1	
+normalized (规格化)	1	0	0	-	
+denormalized (反向规格化)	0	0	1	1	上溢
+zero (零)	0	0	0	-	
-zero (零)	0	0	0	-	
-denormalized (反向规格化)	0	0	1	1	下溢
-normalized (规格化)	0	1	0	-	
-qNaN	1	1	1	1	

举例

STL	解 释
L MD10	// 将 MD10 的值装入累加器 1 中。(该值必须是浮点数格式)。
ATAN	// 在累加器 1 中计算浮点数(32 位, IEEE FP)的反正切值。结果保存到累加器 1 中。
AN OV	// 扫描状态字中的 OV 位是否为“0”。
JC OK	// 如果在 ATAN 执行过程中没有出现错误, 则跳转到 OK 跳转标号。
BEU	// 如果在 ATAN 执行过程中出现错误, 则无条件结束。
OK: T MD20	// 将累加器 1 中的内容 (结果) 传送到 MD20。

9 装载和传送指令

9.1 装载和传送指令概述

说明

使用装载（L）和传送（T）指令，可以对输入或输出模块与存储区之间或在存储器之间的进行信息交换。CPU 在每次扫描中将无条件执行这些指令，也就是说，这些指令不受逻辑运算结果（RLO）的影响。

下述装载和传送指令可供使用：

- L 装载
- L STW 将状态字装入累加器 1
- LAR1 AR2 将地址寄存器 2 的内容装入地址寄存器 1
- LAR1 <D> 将双整数（32 位指针）装入地址寄存器 1
- LAR1 将累加器 1 中的内容装入地址寄存器 1
- LAR2 <D> 将双整数（32 位指针）装入地址寄存器 2
- LAR2 将累加器 2 中的内容装入地址寄存器 1

- T 传送
- T STW 将累加器 1 中的内容传送到状态字
- TAR1 AR2 将地址寄存器 1 的内容传送到地址寄存器 2
- TAR1 <D> 将地址寄存器 1 的内容传送到目标地址（32 位指针）
- TAR2 <D> 将地址寄存器 2 的内容传送到目标地址（32 位指针）
- TAR1 将地址寄存器 1 中的内容传送到累加器 1
- TAR2 将地址寄存器 2 中的内容传送到累加器 1
- CAR 交换地址寄存器 1 和地址寄存器 2 的内容

9.2 L 装载

格式

L <地址>

地 址	数据类型	存储区	源地址
<地址>	字节	E、A、PE、M、L、 D、指针、参数	0...65535
	字		0...65534
	双字		0...65532

说明

使用该指令，可以在累加器 1 的原有内容保存到累加器 2 并将累加器 1 置为“0”之后，将寻址字节、字或双字装入累加器 1 中。

状态字

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

举例

STL	解 释
L IB10	// 将 IB10 装入累加器 1 低字低字节中。
L MB120	// 将 MB120 装入累加器 1 低字低字节中。
L DBB12	// 将 DBB12 装入累加器 1 低字低字节中。
L DIW15	// 将 DIW15 装入累加器 1 低字中。
L LD252	// 将 LD252 装入累加器 1 中。
L P# I 8.7	// 将指针装入累加器 1。
L OTTO	// 将参数“OTTO”装入累加器 1。
L P# ANNA	// 将指针装入累加器 1 的指定参数。（使用该指令可以装入指定参数的相对地址偏移量）。为了计算多重背景功能块中背景数据块中的绝对偏移量，必须将 AR2 寄存器的内容加上该值。

累加器 1 的内容

累加器 1 的内容	累加器 1 高字 中的高字节	累加器 1 高字 中的低字节	累加器 1 低字 中的高字节	累加器 1 低字 中的低字节
在装载指令执行之前	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
在 L MB10 (L <字节>) 执行之后	00000000	00000000	00000000	<MB10>
在 L MW10 (L <字>) 执行之后	00000000	00000000	<MB10>	<MB11>
在 L MD10 (L <双字>) 执行之后	<MB10>	<MB11>	<MB12>	<MB13>
在 L P# ANNA (功能块中) 执行之后	<86>		<相对于功能块开始的 ANNA 位偏移量> 为了计算多背景功能块中背景数据块中的绝对偏移量，必须将 AR2 寄存器的内容加上该值。	
在 L P# ANNA (功能中) 执行之后	<ANNA 数据被传送到区域的交叉地址>			
	X = “1” 或 “0”			

9.3 L STW 将状态字装入累加器 1

格式

L STW

说明

使用 L STW（使用地址 STW 装载）指令，可以将状态字的内容装入累加器 1。指令的执行与状态位无关，而且对状态位没有影响。

注意

对于 S7-300 系列 CPU，L STW 语句不能装入状态字的 FC、STA 和 OR 位。只有位 1、4、5、6、7 和 8 才能装入累加器 1 低字中的相应位。

状态字

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

举例

STL	解 释
L STW	// 将状态字的内容装入累加器 1 中。

执行 L STW 后累加器 1 的内容如下：

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 将累加器 1 中的内容装入地址寄存器 1

格式

LAR1

说明

使用该指令，可以将累加器 1 的内容（32 位指针）装入地址寄存器 AR1。累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

9.5 LAR1 <D> 将双整数(32位指针)装入地址寄存器1

格式

LAR1 <D>

地 址	数据类型	存储区	源地址
<D>	双字 指针常数	D、M、L	0...65532

说明

使用该指令，可以将寻址双字<D>的内容或指针常数装入地址寄存器 AR1。累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

例如：直接寻址

STL	解 释
LAR1 DBD20	// 将 DBD20 中的指针装入 AR1 。
LAR1 DID30	// 将 DID30 中的指针装入 AR1 。
LAR1 LD180	// 将 LD180 中的指针装入 AR1。
LAR1 MD24	// 将 MD24 的内容装入 AR1。

例如：指针常数

STL	解 释
LAR1 P#M100.0	// 将一个 32 位指针常数装入 AR1。

9.6 LAR1 AR2 将地址寄存器 2 的内容装入地址寄存器 1

格式

LAR1 AR2

说明

使用该指令（带地址 AR2 的 LAR1 指令），可以将地址寄存器 AR2 的内容装入地址寄存器 AR1。累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

9.7 LAR2 将累加器 1 中的内容装入地址寄存器 2

格式

LAR2

说明

使用该指令，可以将累加器 1 的内容（32 位指针）装入地址寄存器 AR2。
累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

9.8 LAR2 <D> 将双整数(32位指针)装入地址寄存器2

格式

LAR2 <D>

地 址	数据类型	存储区	源地址
<D>	双字 指针常数	D、M、L	0...65532

说明

使用该指令，可以将寻址双字<D>的内容或指针常数装入地址寄存器 AR2。累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

例如：直接寻址

STL	解 释
LAR2 DBD 20	// 将数据双字 DBD20 中的指针装入 AR2。
LAR2 DID 30	// 将背景数据双字 DID30 中的指针装入 AR2。
LAR2 LD 180	// 将本地数据双字 LD180 中的指针装入 AR2。
LAR2 MD 24	// 将存储双字 MD24 中的指针装入 AR2。

例如：指针常数

STL	解 释
LAR2 P#M100.0	// 将一个 32 位指针常数装入 AR2。

9.9 T 传送

格式

T <地址>

地 址	数据类型	存储区	源地址
<地址>	字节	I、Q、PQ、M、L、 D	0...65535
	字		0...65534
	双字		0...65532

说明

使用该指令，如果主控继电器接通（MCR = 1），可以将累加器 1 中的内容传送（复制）到目标地址。如果 MCR = 0，则目标地址写入“0”。从累加器 1 中复制的字节数量取决于目标地址规定的大小。在传送指令执行完后，累加器 1 还可保存数据。到直接 I/O 区的传送（存储器类型 PQ）也可将累加器 1 的内容或“0”（如果 MCR = 0）传送到过程映像输出表的相应地址（存储器类型 Q）。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
T QB10	// 将累加器 1 低字低字节中的内容传送到输出字节 QB10 中。
T MW14	// 将累加器 1 低字中的内容传送到存储字 MW14。
T DBD2	// 将累加器 1 中的内容传送到数据双字 DBD2。

9.10 T STW 将累加器 1 中的内容传送到状态字

格式

T STW

说明

使用 T STW（使用地址 STW 传送）指令，可以将累加器 1 的位 0 到位 8 传送到状态字。

指令的执行与状态位无关。

状态字

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

举例

STL	解 释
T STW	// 将累加器 1 的位 0 到位 8 传送到状态字。

累加器 1 中的位包含以下状态位：

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

*) 该位不被传送

9.11 CAR 交换地址寄存器 1 和地址寄存器 2 的内容

格式

CAR

说明

使用 CAR（交换地址寄存器）指令，可以将地址寄存器 AR1 和 AR2 中的内容进行交换。指令的执行与状态位无关，而且对状态位没有影响。

地址寄存器 AR1 中的内容移至地址寄存器 AR2 中，地址寄存器 AR2 中的内容移至地址寄存器 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 中的内容传送到累加器 1

格式

TAR1

说明

使用该指令，可以将地址寄存器 AR1 的内容传送到累加器 1（32 位指针）。

累加器 1 的原有内容保存至累加器 2 中。指令的执行与状态位无关，而且对状态位没有影响。

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的内容传送到目的地(32位指针)

格式

TAR1 <D>

地 址	数据类型	存储区	源地址
<D>	双字	D、M、L	0...65532

说明

使用该指令，可以将地址寄存器 AR1 的内容传送到寻址双字 <D>。目标地址可以是存储双字（MD）、本地数据双字（LD）、数据双字（DBD）和背景数据双字（DID）。

累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
TAR1 DBD20	// 将 AR1 中的内容传送到数据双字 DBD20。
TAR1 DID30	// 将 AR1 中的内容传送到背景数据双字 DID30。
TAR1 LD18	// 将 AR1 中的内容传送到本地数据双字 LD18。
TAR1 MD24	// 将 AR1 中的内容传送到存储双字 MD24。

9.14 TAR1 AR2 将地址寄存器1的内容传送到地址寄存器2

格式

TAR1 AR2

说明

使用该指令（使用地址 AR2 的 TAR1 指令），可以将地址寄存器 AR1 的内容传送到地址寄存器 AR2。

累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

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 中的内容传送到累加器 1

格式

TAR2

说明

使用该指令，可以将地址寄存器 AR2 的内容传送到累加器 1（32 位指针）。

累加器 1 的原有内容保存到累加器 2 中。指令的执行与状态位无关，而且对状态位没有影响。

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的内容传送到目的地(32位指针)

格式

TAR2 <D>

地 址	数据类型	存储区	源地址
<D>	双字	D、M、L	0...65532

说明

使用该指令，可以将地址寄存器 AR2 的内容传送到寻址双字 <D>。目标地址可以是存储双字 (MD)、本地数据双字 (LD)、数据双字 (DBD) 和背景双字 (DID)。
累加器 1 和累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
TAR2 DBD20	// 将 AR2 中的内容传送到数据双字 DBD20。
TAR2 DID30	// 将 AR2 中的内容传送到背景双字 DID30。
TAR2 LD18	// 将 AR2 中的内容传送到本地数据双字 LD18。
TAR2 MD24	// 将 AR2 中的内容传送到存储双字 MD24。

10 程序控制指令

10.1 程序控制指令概述

说明

下述程序控制指令可供使用：

- BE 块结束
- BEC 块有条件结束
- BEU 块无条件结束
- CALL 块调用
- CC 条件调用
- UC 无条件调用

- 调用功能块(FB)
- 调用功能(FC)
- 调用系统功能块(SFB)
- 调用系统功能(SFC)
- 调用多背景块
- 从库中调用块

- MCR (主控继电器)
- 使用 MCR 功能的重要注意事项
- MCR(将 RLO 存入 MCR 堆栈，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

说明

使用该指令，可以中止当前块中的程序扫描，并跳转到调用当前块的程序块。然后从调用程序中块调用语句后的第一条指令开始，重新进行程序扫描。同时将当前的本地数据区释放，前一本地数据区即成为当前本地数据区。调用块时打开的数据块将被重新打开。另外，还将恢复调用块的 MCR 相关性，并将 RLO 从当前块传送到调用当前块的程序块。该指令与任何条件无关。但是，如果该指令被跳过，则不结束当前程序扫描，而是从块内跳转到目标处继续程序扫描。

当用于 S5 软件时，该指令略有不同。当用于 S7 软件时，该指令的功能与 BEU 相同。

状态字

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

举例

STL	解 释		
A I 1.0			
JC NEXT	// 如果 RLO = 1 (I 1.0 = 1)，则跳转到 NEXT 跳转标号。		
L IW4	// 如果没有执行跳转，则继续执行程序扫描。		
T IW10			
A I 6.0			
A I 6.1			
S M 12.0			
BE	// 块结束		
NEXT: NOP 0	// 如果执行了跳转，则在此处继续执行程序扫描。		

10.3 BEC 块有条件结束

格式

BEC

说明

如果 RLO = 1，使用该指令，可以中断当前块中的程序扫描，并跳转到调用当前块的程序块。然后从块调用语句后的第一条指令开始，重新进行程序扫描。并将当前的本地数据区释放，前一本本地数据区即成为当前本地数据区。重新打开调用块时打开的数据块将。恢复调用块的 MCR 相关性。

将 RLO (= 1) 从被中止的块传送到被调用的块。如果 RLO = 0，则不执行该指令。将RLO 置为“1”，从该指令后的下一条指令继续程序扫描。

状态字

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

举例

STL	解 释
A I 1.0	// 刷新 RLO。
BEC	// 如果 RLO = 1，结束块。
L IW4	// 如果没有执行 BEC，RLO = 0，则此处继续执行程序扫描。
T MW10	

10.4 BEU 块无条件结束

格式

BEU

说明

使用该指令，可以中止当前块中的程序扫描，并跳转到调用当前块的程序块。然后从块调用语句后的第一条指令开始，重新进行程序扫描。同时将当前的本地数据区释放，前一本地数据区即成为当前本地数据区。调用块时打开的数据块将被重新打开。另外，还将恢复调用块的 MCR 相关性，并将 RLO 从当前块传送到调用当前块的程序块。该指令与任何条件无关。但是，如果该指令被跳过，则不结束当前程序扫描，而是从块内跳转到目标处继续程序扫描。

状态字

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

举例

STL	解 释	
A I 1.0		
JC NEXT	// 如果 RLO = 1 (I 1.0 = 1)，则跳转到 NEXT 跳转标号。	
L IW4	// 如果没有执行跳转，则从此处继续执行程序扫描。	
T IW10		
A I 6.0		
A I 6.1		
S M 12.0		
BEU	// 块无条件结束	
NEXT: NOP 0	// 如果执行了跳转，则从此处继续执行程序扫描。	

10.5 CALL 块调用

格式

CALL <逻辑块标识符>

说明

使用该指令，可以调用功能（FC）或功能块（FB）、系统功能（SFC）或系统功能块（SFB），或调用由西门子公司提供的标准预编程块。该指令可以调用作为地址输入的 FC 和 SFC 或 FB 和 SFB，与 RLO 或其它条件无关。如果调用一个 FB 或 SFB，则必须指定一个相关的背景数据块。被调用块处理完成后，调用块程序继续逻辑处理。逻辑块的地址可以是绝对地址或符号地址。在 SFB/SFC 调用后，将恢复寄存器的内容。

例如：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

如果使用的是语句表编辑器（STL Editor），上表中的 n、n1 和 n2 必须是有效的、已存在的块。同样，在使用之前必须定义符号名。

传递参数（增量编辑方式）

调用块可通过一个变量表与被调用块交换参数。当输入一个有效的调用语句时，语句表程序中的变量表可自动扩展。

如果调用一个功能块（FB）、系统功能块（SFB）、功能（FC）或系统功能（SFC），并且被调用块的变量声明表中有 IN、OUT 和 IN_OUT 声明，则这些变量作为一个形式参数表被添加到调用块中。

当调用功能（FC）和系统功能（SFC）时，则必须在调用逻辑块中将实际参数赋值给形式参数。

当调用功能块（FB）和系统功能块（SFB）时，只需定义与以前调用相比必须进行修改的实际参数。执行完FB后，实际参数保存在背景数据块中。如果实际参数是一个数据块，则必须指定完整的绝对地址，例如 DB1.DBW2。

IN 参数可指定为常数、绝对地址或符号地址。OUT 和 IN_OUT 参数必须指定为绝对地址或符号地址。必须保证所有地址和常数与要传送的数据类型相符。

调用指令将返回地址(选择符和相对地址)、两个当前数据块的选择符以及 MA 位保存在 B (块) 堆栈中。此外，调用指令将取消 MCR 相关性，然后生成被调用块的本地数据区。

状态字

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

举例 1：为 FC6 调用赋值参数

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

举例 2：调用不带参数的 SFC

STL	解 释
CALL	SFC43 // 调用 SFC43，重新触发看门狗定时器（无参数）。

举例 3：调用带背景数据块 DB1 的 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

举例 4：调用带背景数据块 DB2 的 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

每次调用 FB或 SFB都必须有一个背景数据块。在上述举例中，数据块 DB1 和 DB2 必须在调用之前已存在。

10.6 调用功能块

格式

CALL FB n1 , DB n1

说明

使用该指令，可调用用户定义的功能块（FB）。调用指令能够调用作为地址输入的功能块，与 RLO 或其它条件无关。如果使用调用指令调用一个功能块，必须为它提供一个背景数据块。被调用块处理完成后，调用块程序继续逻辑处理。逻辑块的地址可以是绝对地址或符号地址。

传递参数（增量编辑方式）

调用块可通过变量表与被调用块交换参数。当输入一个有效的调用语句时，语句表程序中的变量表可自动扩展。

如果调用一个功能块，并且被调用块的变量声明表中有 IN、OUT 和 IN_OUT 声明，则这些变量作为一个形式参数表被添加到调用块中。

当调用功能块时，只需定义与以前调用相比必须进行修改的实际参数。执行完FB后，实际参数保存在背景数据块中。如果实际参数是一个数据块，则必须指定完整的绝对地址，例如 DB1.DBW2。

IN 参数可指定为常数、绝对地址或符号地址。OUT 和 IN_OUT 参数必须指定为绝对地址或符号地址。必须保证所有地址和常数与要传送的数据类型相符。

调用指令将返回地址（选择符和相对地址）、两个当前数据块的选择符以及 MA 位保存在 B（块）堆栈中。此外，调用指令将取消 MCR 相关性，然后生成被调用块的本地数据区。

状态字

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

举例 1：调用带背景数据块 DB1 的 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

举例 2：调用带背景数据块 DB2 的 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

每一次功能块（FB）调用都必须有一个背景数据块。在上述举例中，数据块 DB1 和 DB2 必须在调用之前已存在。

10.7 调用功能

格式

CALL FC n

注意

如果使用的是语句表编辑器（STL Editor），“n”必须指向已存在的有效块。在使用之前，还必须定义符号名。

说明

使用该指令，可调用功能（FC）。调用指令能够调用作为地址输入的功能，与 RLO 或其它条件无关。被调用块处理完成后，调用块程序继续逻辑处理。逻辑块的地址可以是绝对地址或符号地址。

传递参数（增量编辑方式）

调用块可通过一个变量表与被调用块交换参数。当输入一个有效的调用语句时，语句表程序中的变量表可自动扩展。

如果调用一个功能，并且被调用块的变量声明表中有 IN、OUT 和 IN_OUT 声明，则这些变量作为一个形式参数表被添加到调用块中。

当调用功能（FC）时，则必须在调用逻辑块中将实际参数赋值给形式参数。

IN 参数可指定为常数、绝对地址或符号地址。OUT 和 IN_OUT 参数必须指定为绝对地址或符号地址。必须保证所有地址和常数与要传送的数据类型相符。

调用指令将返回地址（选择符和相对地址）、两个当前数据块的选择符以及 MA 位保存在 B（块）堆栈中。此外，调用指令将取消 MCR 相关性，然后生成被调用块的本地数据区。

状态字

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

例如：为 FC6 调用赋值参数

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 调用系统功能块

格式

CALL SFB n1 , DB n2

说明

使用该指令，可调用由西门子公司提供的标准功能块（SFB）。调用指令能够调用作为地址输入的系统功能块（SFB），与 RLO 或其它条件无关。调用一个系统功能块时必须为它提供一个背景数据块。调用块程序继续逻辑处理。逻辑块的地址可以是绝对地址或符号地址。

传递参数（增量编辑方式）

调用块可通过一个变量表与被调用块交换参数。当输入一个有效的调用语句时，语句表程序中的变量表可自动扩展。

如果调用一个系统功能块，并且被调用块的变量声明表中有 IN、OUT 和 IN_OUT 声明，则这些变量作为一个形式参数表被添加到调用块中。

当调用系统功能块时，只需定义与以前调用相比必须进行修改的实际参数。执行完SFB后，实际参数保存在背景数据块中。如果实际参数是一个数据块，则必须指定完整的绝对地址，例如 DB1.DBW2。

IN 参数可指定为常数、绝对地址或符号地址。OUT 和 IN_OUT 参数必须指定为绝对地址或符号地址。必须保证所有地址和常数与要传送的数据类型相符。

调用指令将返回地址（选择符和相对地址）、两个当前数据块的选择符以及 MA 位保存在 B（块）堆栈中。此外，调用指令将取消 MCR 相关性，然后生成被调用块的本地数据区。

状态字

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

举例

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

注意

每次调用系统功能块都必须有一个背景数据块。在上述举例中，SFB4 和 DB4 必须在调用之前已存在。

10.9 调用系统功能

格式

CALL SFC n

注意

如果使用的是语句表编辑器（STL Editor），“n”必须指向已存在的有效块。在使用之前，还必须定义符号名。

说明

使用该指令，可调用系统功能（SFC）。调用指令能够调用作为地址输入的 SFC，与 RLO 或其它条件无关。被调用块处理完成后，调用块程序继续逻辑处理。逻辑块的地址可以是绝对地址或符号地址。

传递参数（增量编辑方式）

调用块可通过一个变量表与被调用块交换参数。当输入一个有效的调用语句时，语句表程序中的变量表可自动扩展。

如果调用一个系统功能，并且被调用块的变量声明表中有 IN、OUT 和 IN_OUT 声明，则这些变量作为一个形式参数表被添加到调用块中。

当调用系统功能（SFC）时，则必须在调用逻辑块中将实际参数赋值给形式参数。IN 参数可指定为常数、绝对地址或符号地址。OUT 和 IN_OUT 参数必须指定为绝对地址或符号地址。必须保证所有地址和常数与要传送的数据类型相符。

调用指令将返回地址（选择符和相对地址）、两个当前数据块的选择符以及 MA 位保存在 B（块）堆栈中。此外，调用指令将取消 MCR 相关性，然后生成被调用块的本地数据区。

状态字

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

例如：调用一个无参数的系统功能块（SFC）

STL	解 释
CALL SFC43	// 调用 SFC43 ，重新触发看门狗定时器（无参数）。

10.10 调用多背景块

格式

CALL # 变量名

说明

通过声明一个具有功能块数据类型的静态变量，创建一个多重背景。在程序元素目录中只包含已声明的多背景块。

状态字

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

10.11 从库中调用块

SIMATIC 管理器中的库变量可以用于选择以下程序块：

- 集成在 CPU 操作系统中的程序块（“Standard Library（标准库）”）
- 保存在库中以便再次使用的程序块

10.12 CC 条件调用

格式

CC <逻辑块标识符>

说明

使用该指令，可以在 RLO=1 时调用一个逻辑块。该指令用于无参数调用 FC 或 FB 类型的逻辑块。除了不能使用调用程序传递参数外，CC 指令与 CALL 指令的用法相同。该指令可将返回地址（选择符和相对地址）、两个当前数据块的选择符以及 MA 位保存在 B（块）堆栈中，取消激活 MCR 相关性，生成被调用块的本地数据区，并开始执行被调用程序程序。逻辑块的地址可以绝对地址或相对地址。

状态字

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

举例

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	// 如果 I 2.0 = 1，从调用功能返回处执行；如果 I 2.0 = 0，直接在 A I 2.0 语句后执行。	

注意

如果 CALL 指令调用的是一个功能块（FB）或一个系统功能块（SFB），必须在语句中指定一个背景数据块（数据块号）。通过 CC 指令调用时，则不能分配一个数据块。

根据所使用的程序段，在从梯形逻辑编程语言(LAD)转换为语句表编程语言(STL)过程中，程序编辑器（Program Editor）既可以生成 UC 指令也可以生成 CC 指令。所以最好使用 CALL 指令，以避免程序错误。

10.13 UC 无条件调用

格式

UC <逻辑块标识符>

说明

使用该指令，可以调用 FC 或 SFC 类型的逻辑块。除了不能与被调用块传递参数外，UC 指令与 CALL 指令的用法相同。该指令可将返回地址（选择符和相对地址）、两个当前数据块的选择符以及 MA 位保存在 B（块）堆栈中，取消激活 MCR 相关性，生成被调用块的本地数据区，并开始执行被调用程序程序。

状态字

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

举例 1

STL		解 释
UC	FC6	// 调用功能 FC6（无参数）。

举例 2

STL		解 释
UC	SFC43	// 调用系统功能 SFC43（无参数）。

注意

如果 CALL 指令调用的是一个功能块（FB）或一个系统功能块（SFB），必须在语句中指定一个背景数据块（数据块号）。通过 UC 指令调用时，则不能分配一个数据块。

根据所使用的程序段，在从梯形逻辑编程语言(LAD)转换为语句表编程语言(STL)过程中，程序编辑器（Program Editor）既可以生成 UC 指令也可以生成 CC 指令。所以最好使用 CALL 指令，以避免程序错误。

10.14 MCR (主控继电器)

使用 MCR 功能的重要注意事项:



警告
为防止人身伤害或财产损失，禁止使用 MCR 功能代替硬接线机械式主控继电器实现紧急停机功能。

说明

主控继电器（MCR）是一种继电器梯形图逻辑主开关，用于激活或取消能量流。根据 MCR 指令，可执行由以下位逻辑和传送指令触发的操作：

- = <位>
- S <位>
- R <位>
- T <字节>, T <字>, T <双字>

如果 MCR 是“0”，则 T 指令使用字节、字和双字将“0”写入存储区。S 和 R 指令则保持现有的值不变。指令 “=” 可以将“0”写入被寻址的位。

指令取决于 MCR 以及对 MCR 信号状态的反应

MCR 的信号状态	= <位>	S <位> , R <位>	T <字节> , T <字> , T <双字>
0 (“关断”)	写入 “0”。 （掉电时，模仿继电器的静止状态）	不写入。 （掉电时，模仿保持其当前状态的继电器）	写入 “0”。 （掉电时，模仿生成数值 “0” 的元件）
1 (“接通”)	正常处理	正常处理	正常处理

MCR(– MCR 开始 ,)MCR – MCR 结束

MCR 由一个 1 位宽、8 位深的堆栈控制。当所有 8 个输入项都为“1”时，MCR 激活。使用 MCR(指令，可以将 RLO 位复制到 MCR 堆栈中。使用)MCR 指令，可以删除堆栈中的最后一个输入项，并将总出的位置置 1。MCR(和)MCR 指令必须成对使用。发生故障时，即，如果连续 8 次以上使用 MCR(指令，或在 MCR 堆栈为空时尝试执行一个)MCR 指令，将触发 MCRF 错误报文。

MCRA – 激活 MCR , MCRD – 取消激活 MCR

MCRA 和 MCRD 指令必须成对使用。在 MCRA 和 MCRD 之间的指令根据 MCR 位的信号状态执行。在 MCRA-MCRD 程序段之外的指令执行与 MCR 位的信号状态无关。

必须在被调用块中使用 MCRA 指令，对块中功能 (FC) 和功能块 (FB) 的 MCR 相关性进行编程。

10.15 使用 MCR 功能的重要注意事项



通过 MCRA 激活主控继电器时应注意：

- 如果 MCR 去活，则 MCR(和)MCR 之间的程序段的所有赋值 (T, =) 都写入 “0” 值。
- 在 MCR(指令之前，如果 RLO = 0，则无法激活 MCR。



危险：PLC 处于 STOP 状态，或未定义运行特性！

在 VAR_TEMP 中定义临时变量之后，编译器也可以使用写指令访问本地数据，以便计算地址。这就意味着以下指令顺序会将 PLC 置为 STOP 状态，或造成未定义的运行特性：

形式参数访问

- 访问类型为 STRUCT、UDT、ARRAY、STRING 的复杂的 FC 参数组件
- 从具有多背景能力程序块(V2版本的块)中的 IN_OUT 区访问类型为 STRUCT、UDT、ARRAY、STRING 的复杂的 FB 参数组件
- 访问具有多背景能力的功能块参数(V2版本的块)，其地址大于 8180.0
- 在具有多重背景能力(V2版本的块)的功能块中访问类型为BLOCK_DB的参数。任何后续的数据访问都会将 CPU 切换为“STOP（停止）”状态。T 0、C 0、FC0 或 FB0 用于 TIMER、COUNTER、BLOCK_FC 和 BLOCK_FB。

参数传递

- 通过调用传递参数。

LAD/FBD

- 在 LAD 或 FBD 中 T 分支和中间输出以 RLO = 0 开始。

补救

从释放它们对 MCR 的相关性，来解决以上问题：

1. 在出现问题的语句或程序段之前，利用 MCRD 指令，去活主控继电器。
2. 在出现问题的语句或程序段之后，利用 MCRA 指令，再次激活主控继电器。

10.16 MCR(将 RLO 存入 MCR 堆栈 , 开始 MCR

使用 MCR 功能的重要注意事项

格式

MCR(

说明

使用 MCR((打开一个 MCR 程序段) 指令 , 可以将 RLO 保存在 MCR 堆栈中 , 同时打开一个 MCR 程序段。MCR 程序段是指 MCR(和)MCR 之间的指令。MCR(和)MCR 必须组合使用。

如果 RLO=1, 则 MCR 激活。并执行 MCR 程序段中的指令。

如果 RLO=0, 则 MCR 去活。

根据下表执行 MCR 程序段中的指令。

取决于 MCR 位的信号状态的指令

MCR 的信号状态	= <位>	S <位> , R <位>	T <字节> , T <字> , T <双字>
0 (“断开”)	写入 “0”。 (掉电时 , 模仿继电器的静止状态)	不写入。 (掉电时 , 模仿保持其当前状态的继电器)	写入 “0”。 (掉电时 , 模仿生成数值 “0” 的元件)
1 (“接通”)	正常处理	正常处理	正常处理

MCR(和)MCR 指令可以嵌套使用。最大嵌套深度为 8 级。最多可有 8 个嵌套堆栈输入项。在堆栈满时 , 执行 MCR(会产生一个 MCR 堆栈故障 (MCRF) 。

状态字

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

举例

STL	解 释
MCRA	// 激活 MCR 程序段。
A I 1.0	
MCR(// 将 RLO 存入 MCR 堆栈, 打开 MCR 程序段。当 RLO=1 (I 1.0 = “1”) 时, MCR= “接通”; 当 RLO=0 (I 1.0 = “0”) 时, MCR= “断开”。
A I 4.0	
= Q 8.0	// 如果 MCR= “断开”, 不管 I 4.0 的状态如何, 将 Q 8.0 置位为 “0”。
L MW20	
T QW10	// 如果 MCR= “断开”, 则将 “0” 传送到 QW10。
)MCR	// 结束 MCR 程序段
MCRD	// 取消激活 MCR 程序段
A I 1.1	
= Q 8.1	// 这些指令在 MCR 区域以外, 与 MCR 位的信号状态无关。

10.17)MCR 结束 MCR

使用 MCR 功能的重要注意事项

格式

)MCR

说明

使用)MCR 指令（结束 MCR 程序段），可以从 MCR 堆栈中删除一个输入项，并结束 MCR 程序段。最后一个 MCR 堆栈位置被释放，并设置为“1”。指令 MCR(和)MCR 必须组合使用。在堆栈为空时，执行)MCR 指令会产生一个 MCR 堆栈故障（MCRF）。

状态字

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

举例

STL	解 释
MCRA	// 激活 MCR 程序段
A I 1.0	
MCR(// 将 RLO 存入 MCR 堆栈中，打开 MCR 程序段。当 RLO=1（I 1.0 = “1”）时，MCR= “接通”；当 RLO=0（I 1.0 = “0”）时，MCR= “断开”。
A I 4.0	
= Q 8.0	// 如果 MCR= “断开”，不管 I 4.0 的状态如何，将 Q 8.0 置位为 “0”。
L MW20	
T QW10	// 如果 MCR= “断开”，则传送 “0” 到 QW10。
)MCR	// 结束 MCR 程序段
MCRD	// 取消激活 MCR 程序段
A I 1.1	
= Q 8.1	// 这些指令在 MCR 程序段以外，与 MCR 位的信号状态无关。

10.18 MCRA 激活 MCR 程序段

使用 MCR 功能的重要注意事项

格式

MCRA

说明

使用 MCRA 指令（主控继电器激活），可以激活其后指令的 MCR 相关性。MCRA 指令必须与 MCRD 指令（主控继电器取消激活）组合使用。MCRA 和 MCRD 之间的指令根据 MCR 位的信号状态执行。
指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
MCRA	// 激活 MCR 程序段
A I 1.0	
MCR(// 将 RLO 存入 MCR 堆栈中，打开 MCR 程序段。当 RLO=1 (I 1.0 = “1”) 时，MCR= “接通”；当 RLO=0 (I 1.0 = “0”) 时，MCR= “断开”。
A I 4.0	
= Q 8.0	// 如果 MCR= “断开”，不管 I 4.0 的状态如何，将 Q 8.0 置位为 “0”。
L MW20	
T QW10	// 如果 MCR= “断开”，则传送 “0” 到 QW10。
)MCR	// 结束 MCR 程序段
MCRD	// 取消激活 MCR 程序段
A I 1.1	
= Q 8.1	// 这些指令在 MCR 程序段以外，与 MCR 位的信号状态无关。

10.19 MCRD 取消激活 MCR 程序段

使用 MCR 功能的重要注意事项

格式

MCRD

说明

使用 MCRD 指令（主控继电器去活），可以取消激活其后指令的 MCR 相关性。MCRD 指令必须与 MCRA 指令（主控继电器激活）组合使用。MCRA 和 MCRD 之间的指令根据 MCR 位的信号状态执行。
指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
MCRA	// 激活 MCR 程序段
A I 1.0	
MCR(// 将 RLO 存入 MCR 堆栈中，打开 MCR 程序段。当 RLO=1（I 1.0 = “1”）时，MCR= “接通”；当 RLO=0（I 1.0 = “0”）时，MCR= “断开”。
A I 4.0	
= Q 8.0	// 如果 MCR= “断开”，不管 I 4.0 的状态如何，将 Q 8.0 置位为 “0”。
L MW20	
T QW10	// 如果 MCR= “断开”，则传送 “0” 到 QW10。
)MCR	// 结束 MCR 程序段
MCRD	// 取消激活 MCR 程序段
A I 1.1	
= Q 8.1	// 这些指令在 MCR 程序段以外，与 MCR 位的信号状态无关。

11 移位和循环移位指令

11.1 移位指令

11.1.1 移位指令概述

说明

使用移位指令,可以将累加器 1 低字中的内容或整个累加器的内容向左或向右逐位移动(请参见“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,如果将十进制数“3”的等效二进制数左移 3 位,则累加器中的结果是十进制数“24”的二进制数。如果将十进制数“16”的等效二进制数右移 2 位,则累加器中的结果是十进制数“4”的二进制数。

移位指令后的数字或累加器 2 低字中低字节的数值指出要移动的位数。执行移位指令所空出的位既可以用零填入,也可以用符号位的信号状态填入(“0”代表“正”,“1”代表“负”)。最后移出的位装入状态字的 CC 1 位。状态字的 CC 0 和 OV 位清零。可用跳转指令判断 CC 1 位的状态。移位操作是无条件的,也就是说,它们的执行是无条件的,也不影响逻辑运算结果。

下述移位指令可供使用:

- SSI 移位有符号整数(16 位)
- SSD 移位有符号双整数(32 位)
- SLW 字(16 位)左移
- SRW 字(16 位)右移
- SLD 双字(32 位)左移
- SRD 双字(32 位)右移

11.1.2 SSI 移位有符号整数（16 位）

格式：

SSI

SSI <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	移位的位数在 0 和 15 之间。

说明

使用 SSI（右移有符号整数）指令，只将累加器 1 低字中的内容逐位右移。由移位指令空出的位用符号位（位 15）的信号状态填充。最后移出的位被装入状态字位 CC 1。要移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

SSI <数值>：移位位数通过地址 <数值> 定义。允许数值范围为 0 – 15。如果 <数值> 大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果 <数值> 等于“0”，则移位指令相当于空操作（NOP）。

SSI：移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果移位位数大于 16，则始终产生相同结果（ACCU 1 = 16#0000，CC 1 = 0，或 ACCU 1 = 16#FFFF，CC 1 = 1）。如果移位位数大于“0”，则状态字位 CC 0 和 OV 被置为“0”。如果移位位数等于“0”，则移位指令相当于空操作（NOP）。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
SSI 6 执行之前	0101	1111	0110	0100	1001	1101	0011	1011
SSI 6 执行之后	0101	1111	0110	0100	1111	1110	0111	0100

举例 1

STL	解 释
L MW4	// 将数值装入累加器 1 中。
SSI 6	// 将累加器 1 中的内容右移 6 位，包括符号位。
T MW8	// 将结果传送到 MW8。

举例 2

STL	解 释
L +3	// 将数值 “+3” 装入累加器 1 中。
L MW20	// 将累加器 1 中的内容装入累加器 2 中。将 MW20 的值装入累加器 1 中。
SSI	// 移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 低字中的内容右移 3 位，包括符号位；空出的位以符号位的信号状态填充。
JP NEXT	// 如果最后移出的位（CC 1） = 1，则跳转到 NEXT 跳转标号。

11.1.3 SSD 移位有符号双整数（32 位）

格式：

SSD
SSD <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	移位的位数在 0 和 32 之间。

说明

使用 SSD（右移有符号双整数）指令，可以将累加器 1 中的内容逐位右移。由移位指令空出的位用符号位的信号状态填充。最后移出的位被装入状态字位 CC 1。要移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

SSD <数值>：移位位数通过地址<数值>定义。允许数值范围为 0 – 32。如果 <数值> 大于 “0”，则 CC 0 和 OV 状态字位被置为 “0”。如果 <数值> 等于 “0”，则移位指令相当于空操作（NOP）。

SSD：移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果移位位数大于 32，则总是产生相同结果（ACCU 1 = 32#00000000，CC 1 = 0 或 ACCU 1 = 32#FFFFFFF，CC 1 = 1）。如果移位位数大于 “0”，则状态字位 CC 0 和 OV 被置为 “0”。如果移位位数等于 “0”，则移位指令相当于空操作（NOP）。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
SSD 7 执行之前	1000	1111	0110	0100	0101	1101	0011	1011
SSD 7 执行之后	1111	1111	0001	1110	1100	1000	1011	1010

举例 1

STL	解 释
L MD4	// 将数值装入累加器 1 中。
SSD 7	// 将累加器 1 中的位右移 7 位，包括符号位。
T MD8	// 将结果传送到 MD8。

举例 2

STL	解 释
L +3	// 将数值 “+3” 装入累加器 1 中。
L MD20	// 将累加器 1 中的内容装入累加器 2 中。将 MD20 的值装入累加器 1 中。
SSD	// 移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 中的内容右移 3 位，包括符号位；空出的位以符号位的信号状态填充。
JP NEXT	// 如果最后移出的位（CC 1） = 1，则跳转到 NEXT 跳转标号。

11.1.4 SLW 字 (16 位) 左移

格式：

SLW

SLW <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	移位的位数在 0 和 15 之间。

说明

使用 SLW（字左移）指令，只将累加器 1 低字中的内容逐位左移。由移位指令空出的位用“0”填充。最后移出的位被装入状态字位 CC 1。要移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

SLW <数值>：移位位数通过地址<数值>定义。允许数值范围为 0 – 15。如果 <数值> 大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果 <数值> 等于“0”，则移位指令相当于空操作（NOP）。

SLW：移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果移位位数大于 16，则总是产生相同结果：ACCU 1- L = 0，CC 1 = 0，CC 0 = 0 和 OV = 0。如果移位位数小于等于 16，则状态字位 CC 0 和 OV 被置为“0”。如果移位位数等于“0”，则移位指令相当于空操作（NOP）。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
SLW 5 执行之前	0101	1111	0110	0100	0101	1101	0011	1011
SLW 5 执行之后	0101	1111	0110	0100	1010	0111	0110	0000

举例 1

STL	解 释
L MW4	// 将数值装入累加器 1 中。
SLW 5	// 将累加器 1 中的内容左移 5 位。
T MW8	// 将结果传送到 MW8。

举例 2

STL	解 释
L +3	// 将数值 “+3” 装入累加器 1 中。
L MW20	// 将累加器 1 中的内容装入累加器 2 中。将存储字 MW20 的值装入累加器 1 中。
SLW	// 移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 低字中的内容左移 3 位。
JP NEXT	// 如果最后移出的位 (CC 1) = 1, 则跳转到 NEXT 跳转标号。

11.1.5 SRW 字 (16 位) 右移

格式：

SRW

SRW <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	移位的位数在 0 和 15 之间。

说明

使用 SRW（字右移）指令，只将累加器 1 低字中的内容逐位右移。由移位指令空出的位用“0”填充。最后移出的位被装入状态字位 CC 1。要移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

SRW <数值>：移位位数通过地址<数值>定义。允许数值范围为 0 – 15。如果 <数值> 大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果 <数值> 等于“0”，则移位指令相当于空操作（NOP）。

SRW：移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果移位位数大于 16，则总是产生相同结果：ACCU 1- L = 0, CC 1 = 0, CC 0 = 0 和 OV = 0。如果移位位数小于等于 16，则状态字位 CC 0 和 OV 被置为“0”。如果移位位数等于“0”，则移位指令相当于空操作（NOP）。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
	31...16	15...0
SRW 6 执行之前	0101	1111	0110	0100	0101	1101	0011	1011
SRW 6 执行之后	0101	1111	0110	0100	0000	0001	0111	0100

举例 1

STL	解 释
L MW4	// 将数值装入累加器 1 中。
SRW 6	// 将累加器 1 低字中的位右移 6 位。
T MW8	// 将结果传送到 MW8。

举例 2

STL	解 释
L +3	// 将数值 “+3” 装入累加器 1 中。
L MW20	// 将累加器 1 中的内容装入累加器 2 中。将存储字 MW20 的值装入累加器 1 中。
SRW	// 移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 低字中的内容右移 3 位。
SPP NEXT	// 如果最后移出的位 (CC 1) = 1，则跳转到 NEXT 跳转标号。

11.1.6 SLD 双字 (32 位) 左移

格式：

SLD

SLD <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	移位的位数在 0 和 32 之间。

说明

使用 SLD（双字左移）指令，可以将累加器 1 中的内容逐位左移。由移位指令空出的位用“0”填充。最后移出的位被装入状态字位 CC 1。要移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

SLD <数值>：移位位数通过地址<数值>定义。允许数值范围为 0 – 32。如果 <数值> 大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果 <数值> 等于“0”，则移位指令相当于空操作（NOP）。

SLD：移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果移位位数大于 32，则总是产生相同结果：ACCU 1 = 0，CC 1 = 0，CC 0 = 0 和 OV = 0。如果移位位数小于等于 32，则状态字位 CC 0 和 OV 被置为“0”。如果移位位数等于“0”，则移位指令相当于空操作（NOP）。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
SLD 5 执行之前	0101	1111	0110	0100	0101	1101	0011	1011
SLD 5 执行之后	1110	1100	1000	1011	1010	0111	0110	0000

举例 1

STL	解 释
L MD4	// 将数值装入累加器 1 中。
SLD 5	// 将累加器 1 中的内容左移 5 位。
T MD8	// 将结果传送到 MD8。

举例 2

STL	解 释
L +3	// 将数值“+3”装入累加器 1 中。
L MD20	// 将累加器 1 中的内容装入累加器 2 中。将存储字 MD20 的值装入累加器 1 中。
SLD	// 移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 中的内容左移 3 位。
JP NEXT	// 如果最后移出的位（CC 1） = 1，则跳转到 NEXT 跳转标号。

11.1.7 SRD 双字（32 位）右移

格式：

SRD
SRD <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	移位的位数在 0 和 32 之间。

说明

使用 SRD（双字右移）指令，可以将累加器 1 中的内容逐位右移。由移位指令空出的位用“0”填充。最后移出的位被装入状态字位 CC 1。要移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

SRD <数值>：移位位数通过地址<数值>定义。允许数值范围为 0 – 32。如果 <数值> 大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果 <数值> 等于“0”，则移位指令相当于空操作（NOP）。

SRD：移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果移位位数大于 32，则总是产生相同结果：ACCU 1 = 0，CC 1 = 0，CC 0 = 0 和 OV = 0。如果移位位数小于等于 32，则状态字位 CC 0 和 OV 被置为“0”。如果移位位数等于“0”，则移位指令相当于空操作（NOP）。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
SRD 7 执行之前	0101	1111	0110	0100	0101	1101	0011	1011
SRD 7 执行之后	0000	0000	1011	1110	1100	1000	1011	1010

举例 1

STL	解 释
L MD4	// 将数值装入累加器 1 中。
SRD 7	// 将累加器 1 中的内容右移 7 位。
T MD8	// 将结果传送到 MD8。

举例 2

STL	解 释
L +3	// 将数值“+3”装入累加器 1 中。
L MD20	// 将累加器 1 中的内容装入累加器 2 中。将存储字 MD20 的值装入累加器 1 中。
SRD	// 移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 中的内容右移 3 位。
JP NEXT	// 如果最后移出的位（CC 1） = 1，则跳转到 NEXT 跳转标号。

11.2 循环移位指令

11.2.1 循环移位指令概述

说明

使用循环移位指令，可以将累加器 1 中的全部内容循环地逐位左移或右移（参见“CPU 寄存器”）。循环移位指令触发类同于第 11.1 节描写的移位指令。所不同的是，空位填以从累加器中移出的位。

循环移位指令后面的数字或累加器 2 低字低字节中的数字表示需要循环移动的位数。根据指令，通过状态字的 CC 1 位执行循环。状态字的 CC 0 位复位为“0”。

下述循环移位指令可供使用：

- RLD 双字（32 位）循环左移
- RRD 双字（32 位）循环右移
- RLDA 通过 CC 1（32 位）对累加器 1 循环左移
- RRDA 通过 CC 1（32 位）对累加器 1 循环右移

11.2.2 RLD 双字（32 位）循环左移

格式

RLD

RLD <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	循环移位的位数在 0 和 32 之间。

说明

使用 RLD（双字循环左移）指令，可以将累加器 1 中的内容逐位循环左移。通过循环移位指令空出的位以从累加器 1 移出位的信号状态填充。最后循环移出的位被装入状态字位 CC 1。要循环移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

RLD <数值>：循环移位位数通过地址<数值>定义。允许数值范围为 0 – 32。如果 <数值> 大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果 <数值> 等于“0”，则循环移位指令相当于空操作（NOP）。

RLD：循环移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果累加器 2 低字低字节中的数值大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果循环移位位数等于“0”，则循环移位指令相当于空操作（NOP）。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
位	31...16	15...0
RLD 4 执行之前	0101	1111	0110	0100	0101	1101	0011	1011
RLD 4 执行之后	1111	0110	0100	0101	1101	0011	1011	0101

举例 1

STL	解 释
L MD2	// 将数值装入累加器 1 中。
RLD 4	// 将累加器 1 中的内容循环左移 4 位。
T MD8	// 将结果传送到 MD8。

举例 2

STL	解 释
L +3	// 将数值“+3”装入累加器 1 中。
L MD20	// 将累加器 1 中的内容装入累加器 2 中。将存储字 MD20 的值装入累加器 1 中。
RLD	// 循环移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 中的内容循环左移 3 位。
JP NEXT	// 如果最后循环移出的位（CC 1） = 1，则跳转到 NEXT 跳转标号。

11.2.3 RRD 双字（32 位）循环右移

格式：

RRD

RRD <数值>

地 址	数据类型	说 明
<数值>	整数，无符号	循环移位的位数在 0 和 32 之间。

说明

使用 RRD（双字循环右移）指令，可以将累加器 1 中的内容逐位循环右移。通过循环移位指令空出的位以从累加器 1 移出位的信号状态填充。最后循环移出的位被装入状态字位 CC 1。要循环移位的位数可以通过地址<数值>或通过累加器 2 低字低字节中的数值定义。

RRD <数值>：循环移位位数通过地址<数值>定义。允许数值范围为 0 – 32。如果 <数值> 大于“0”，则 CC 0 和 OV 状态字位被置为“0”。如果 <数值> 等于“0”，则循环移位指令相当于空操作（NOP）。

RRD：循环移位的位数通过累加器 2 低字低字节中的数值定义。允许数值范围为 0 – 255。如果累加器 2 低字低字节中的数值大于“0”，则状态字位被置为“0”。

状态字

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

举例

内 容	累加器 1 高字				累加器 1 低字			
位	31..16	15...0
RRD 4 执行之前	0101	1111	0110	0100	0101	1101	0011	1011
RRD 4 执行之后	1011	0101	1111	0110	0100	0101	1101	0011

举例 1

STL	解 释
L MD2	// 将数值装入累加器 1 中。
RRD 4	// 将累加器 1 中的内容循环右移 4 位。
T MD8	// 将结果传送到 MD8。

举例 2

STL	解 释
L +3	// 将数值“+3”装入累加器 1 中。
L MD20	// 将累加器 1 中的内容装入累加器 2 中。将存储字 MD20 的值装入累加器 1 中。
RRD	// 循环移位位数为累加器 2 低字低字节中的数值 => 将累加器 1 中的内容循环右移 3 位。
JP NEXT	// 如果最后循环移出的位 (CC 1) = 1, 则跳转到 NEXT 跳转标号。

11.2.4 RLDA 通过 CC 1 (32 位) 对累加器 1 循环左移

格式

RLDA

说明

使用该指令, 可以将累加器 1 中的全部内容通过 CC 1 循环左移 1 位。状态位 CC 0 和 OV 被置为“0”。

状态字

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

举例

内 容	CC 1	累加器 1 高字				累加器 1 低字			
位		31...16	15...0
RLDA 执行之前	X	0101	1111	0110	0100	0101	1101	0011	1011
RLDA 执行之后	0	1011	1110	1100	1000	1011	1010	0111	011X
(X = 0 或 1, CC 1 的先前信号状态)									

STL	解 释
L MD2	// 将 MD2 的数值装入累加器 1。
RLDA	// 将累加器 1 中的内容通过 CC 1 循环左移 1 位。
JP NEXT	// 如果最后循环移出的位 (CC 1) = 1, 则跳转到 NEXT 跳转标号。

11.2.5 RRDA 通过 CC 1 (32 位) 对累加器 1 循环右移

格式

RRDA

说明

使用该指令，可以将累加器 1 中的全部内容通过 CC 1 循环右移 1 位。状态位 CC 0 和 OV 被置为“0”。

状态字

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

举例

内 容	CC 1	累加器 1 高字				累加器 1 低字			
位		31...16	15...0
RRDA 执行之前	X	0101	1111	0110	0100	0101	1101	0011	1011
RRDA 执行之后	1	X010	1111	1011	0010	0010	1110	1001	1101
	(X = 0 或 1, CC 1 的先前信号状态)								

STL	解 释
L MD2	// 将 MD2 的数值装入累加器 1。
RRDA	// 将累加器 1 中的内容通过 CC 1 循环右移 1 位。
JP NEXT	// 如果最后循环移出的位 (CC 1) = 1, 则跳转到 NEXT 跳转标号。

12 定时器指令

12.1 定时器指令概述

说明

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。

下述定时器指令可供使用：

- FR 使能定时器（任意）
- L 将当前定时值作为整数装入累加器 1
- LC 将当前定时值作为 BCD 码装入累加器 1
- R 复位定时器
- SD 延时接通定时器
- SE 扩展脉冲定时器
- SF 延时断开定时器
- SP 脉冲定时器
- SS 保持型延时接通定时器

12.2 存储器中定时器的位置和定时器的组件

存储器区域

在 CPU 的存储器中有为定时器保留有存储区。该存储区为每一定时器地址保留一个 16 位的字。梯形图指令集支持 256 个定时器。请参考有关 CPU 的技术资料来建立有效数量的定时器字。

下列功能可以访问定时器存储区：

- 定时器指令
- 利用时钟定时刷新定时器字。这是 CPU 在 RUN 模式下的功能，按时基规定的时间间隔递减所给定的时间值，一直到时间值等于“0”。

时间值

定时器字的 0 - 9 位包含二进制码的时间值。时间值按单位个数给出。时间刷新按时基规定的时间间隔对时间值递减一个时间单位。时间值逐渐连续减少，直至等于“0”。时间值可以以二进制、十六进制和二十进制（BCD）格式输入累加器 1 的低位字。

你可以使用下列格式预装一个时间值：

- W#16#txyz
 - 其中，t = 时基（即时间间隔或分辨率）
 - 其中，xyz = 二十进制格式的时间值
- S5T#aH_bM_cS_dMS
 - 其中，H=小时，M=分钟，S=秒，MS=毫秒；用户定义变量：a, b, c, d
 - 自动选择时基，时间值按其所取时基取整为下一个较小的数。

可输入的最大时间值是9,990秒，或2H_46M_30S（2小时46分30秒）。

时基

定时器字的 12 - 13 位包含二进制码的时基。时基定义了时间值递减的单位时间间隔。最小时基为 10 ms；最大时基为 10 s。

时 基	时基的二进制码
10 毫秒	00
100 毫秒	01
1 秒	10
10 秒	11

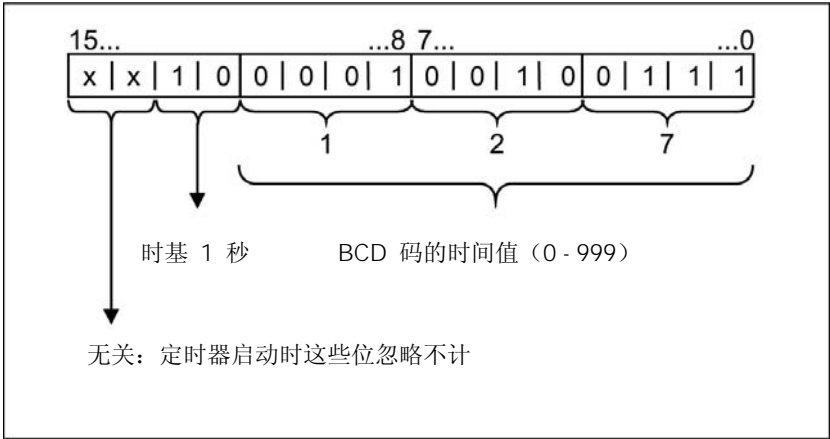
数值不允许超过 2h46m30s(2小时46分30秒)。对于较高的时间值(例如, 2h10ms)而言，如果分辨率过高将被截尾为有效分辨率。S5TIME 的通用格式具有如下所示的范围和分辨率：

分辨率	范 围
0.01 秒	10 毫秒 - 9 秒 990 毫秒
0.1 秒	100 毫秒 - 1 分 39 秒 900 毫秒
1 秒	1 秒 - 16 分 39 秒
10 秒	10 秒 - 2 小时 46 分 30 秒

在累加器 1 中的位

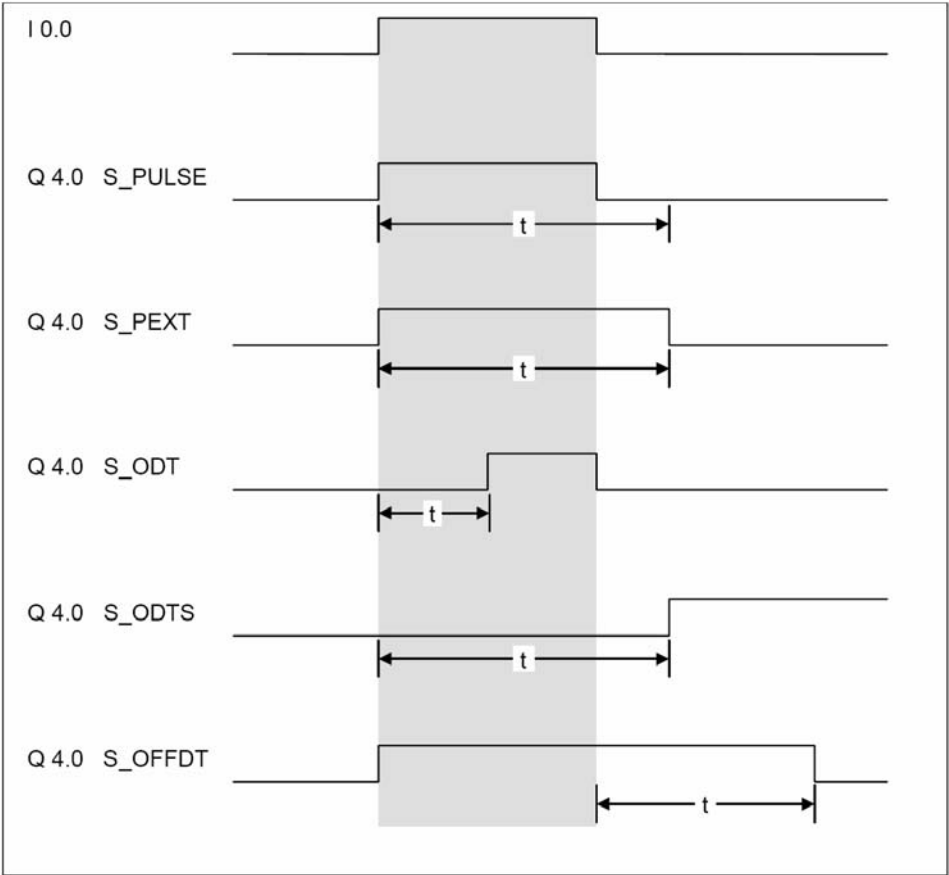
当定时器启动时，累加器 1 的内容用作时间值。累加器 1 的 0 - 11 位为二 - 十进制格式的时间值（BCD 格式：四位一组表示一位十进制数值的二进制码）。12 - 13 位为二进制码的时基。

下图所示为累加器 1 低字中的内容，其中定时值为 127，时基为 1 秒：



选择合适的定时器

本概述旨在用于帮助您选择正确的定时器。



定时器	说 明
S_PULSE 脉冲定时器	输出信号为“1”的最长时间等于编程设定的时间值 t。如果输入信号变为“0”，则输出为“0”。
S_PEXT 延时脉冲定时器	不管输入信号为“1”的时间有多长，输出信号为“1”的时间长度等于编程设定的时间值。
S_ODT 延时接通定时器	只有当编程设定的时间已经结束并且输入信号仍为“1”时，输出信号才从“0”变为“1”。
S_ODTS 保持型延时接通定时器	只有当编程设定的时间已经结束时，输出信号才从“0”变为“1”，而与输入信号为“1”的时间长短无关。
S_OFFDT 延时断开定时器	当输入信号变为“1”或定时器在运行时，输出信号变为“1”。当输入信号从“1”变为“0”时，定时器启动。

12.3 FR 使能定时器（任意）

格式

FR <定时器>

地 址	数据类型	存储区	说 明
<定时器>	TIMER	T	定时器编号，范围与 CPU 有关

指令说明

当 RLO 从“0”变为“1”时，该指令将清除用于启动寻址定时器的边沿检测标志。位于 FR 指令之前的 RLO 位从“0”到“1”的变化，可以使能定时器。

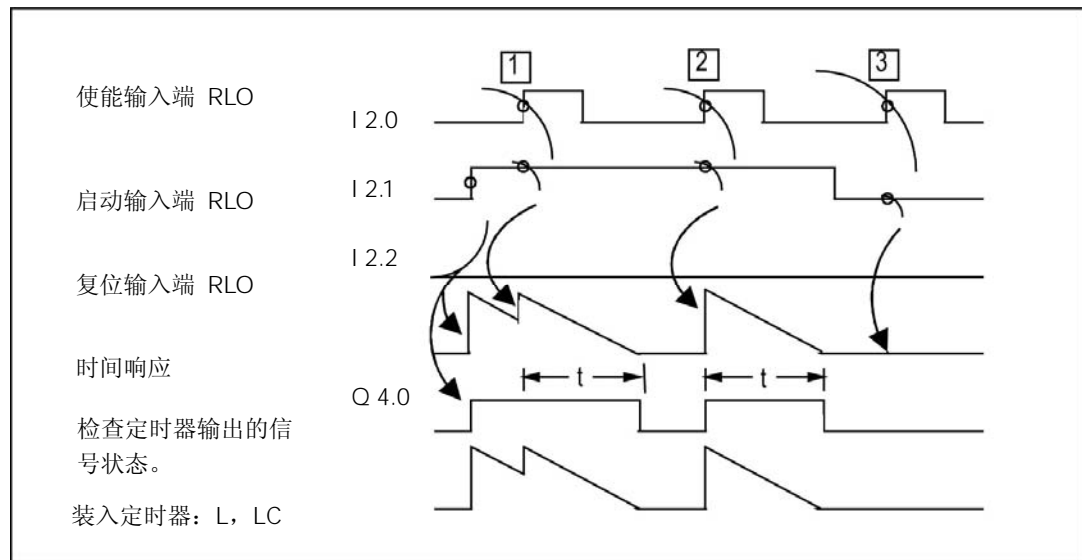
使能定时器指令并不是启动定时器的必要条件，也不是正常定时器操作的必要条件。它只是用于触发一个正在运行的定时器再启动。只有在 RLO = 1 启动操作连续进行时才能实现再启动。

状态字

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

举例

STL	解 释
A I 2.0	
FR T1	// 使能定时器 T1。
A I 2.1	
L S5T#10s	// 预设累加器 1 为 10 秒。
SI T1	// 以脉冲定时器方式启动定时器 T1。
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	// 二进制码的格式装载定时器 T1 的当前时间值。
T MW10	



t = 设定的时间间隔

(1) 当定时器正在运行时，如果在使能输入端 RLO 从“0”变为“1”，将重新启动定时器。定时器重新启动时间为由编程设定的时间。如果在使能输入端 RLO 从“1”变为“0”，则对定时器没有影响。

(2) 如果在定时器没有运行时，在使能输入端 RLO 从“0”变为“1”，并且在启动输入端 RLO 仍为“1”，定时器也将根据编程设定的时间以脉冲定时器格式被启动。

(3) 如果在使能输入端 RLO 从“0”变为“1”，且在启动输入端 RLO 为“0”，则对定时器没有影响。

12.4 L 将当前定时值作为整数装入累加器 1

格式

L <定时器>

地 址	数据类型	存储区	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

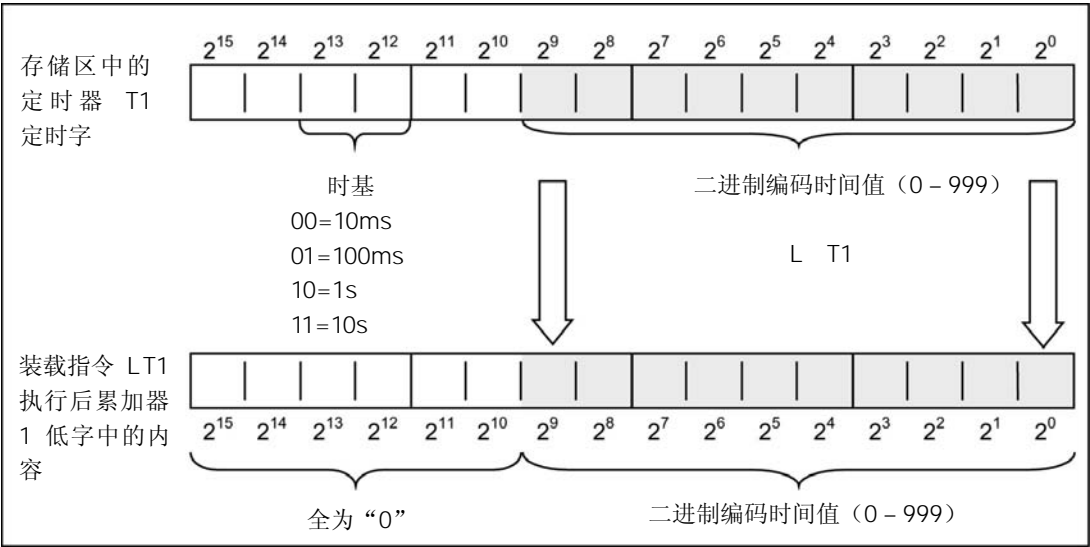
使用该指令，可以在累加器 1 的内容保存到累加器 2 中之后，从寻址定时器字中将当前时间值以二进制整数（不包括时基）装入累加器 1 低字中。

状态字

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

举例

STL	解 释
LT1	// 将定时器 T1 的当前定时值以二进制格式装入累加器 1 低字。



注意

该指令只能将当前定时值的二进制码装入累加器 1 的低字中，不能装入时基。装入的时间值等于初始时间减去定时器启动后所经历的时间。

12.5 LC 将当前定时器值作为 BCD 码装入累加器 1

格式

LC <定时器>

地 址	数据类型	存储区	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

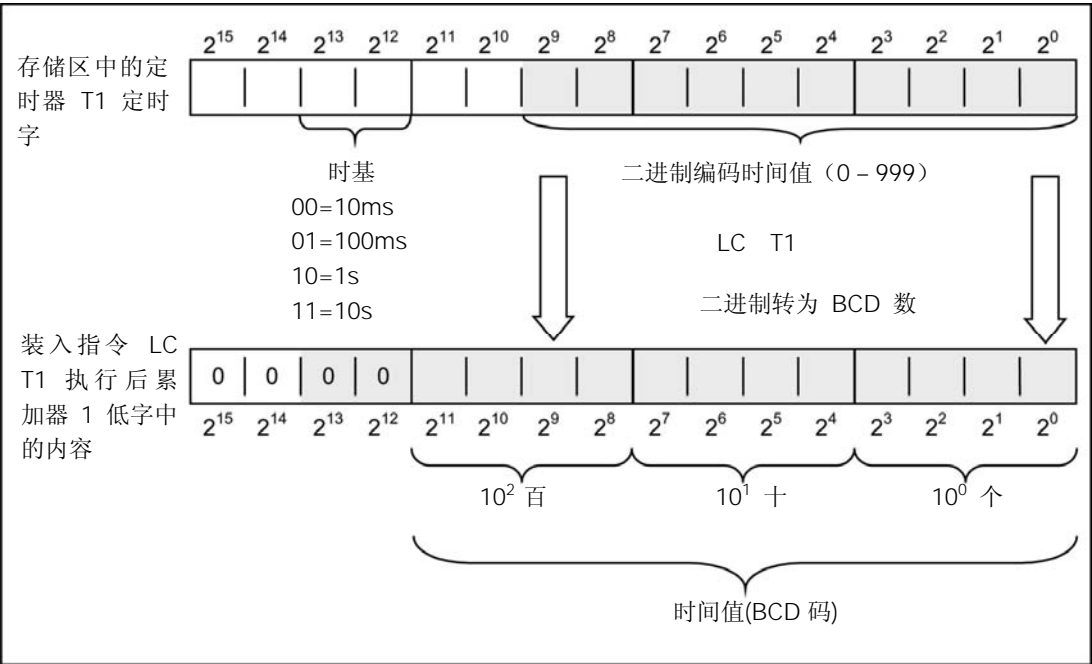
使用该指令，可以在累加器 1 的内容保存到累加器 2 中之后，从寻址定时器字中将当前时间值和时基以二 - 十进制格式（BCD 码）装入累加器 1 中。

状态字

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

举例

STL	解 释
LC T1	// 将定时器 T1 的当前定时值和时基以二 - 十进制格式（BCD）装入累加器 1 低字。



12.6 R 复位定时器

格式

R <定时器>

地 址	数据类型	存储区	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

该指令在 RLO 从“0”变为“1”时，停止当前定时功能，并对寻址定时器字的时间值和时基清零。

状态字

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

举例

STL	解 释
A I 2.1	
R T1	// 检查输入 I 2.1 的信号状态。当 RLO 从“0”变为“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	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

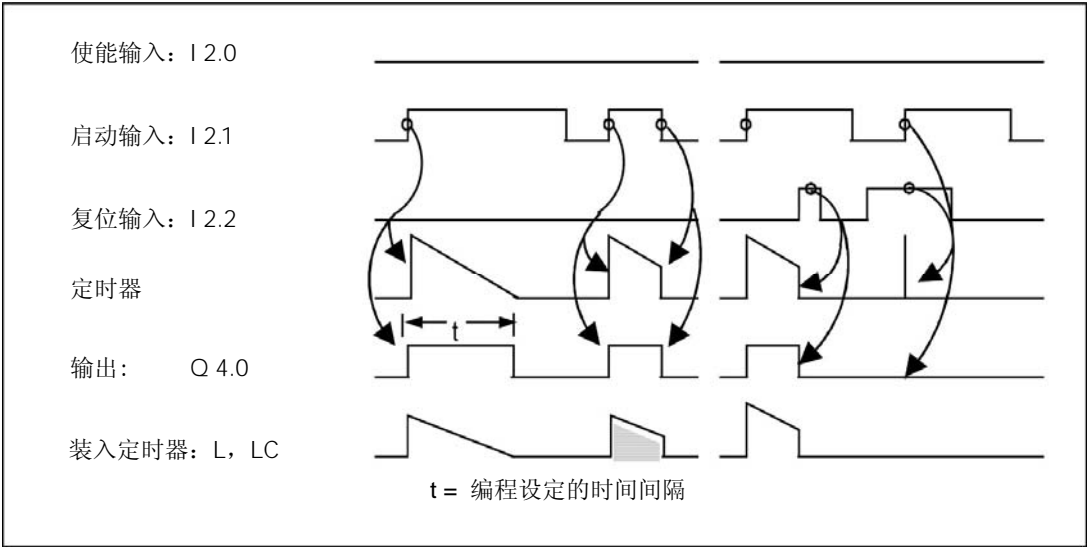
该指令可以在 RLO 从“0”变为“1”时，启动寻址的定时器。
只要 RLO = 1，即开始进行设定时间计时。如果在设定时间到时前 RLO 变为“0”，则定时器停止。该定时器启动指令需要将时间值和时基以 BCD 码的格式保存在累加器 1 的低字中。
请参见“定时器在存储器中的位置和定时器组件”。

状态字

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

举例

STL	解 释
A I 2.0	
FR T1	// 使能定时器 T1。
A I 2.1	
L S5T#10s	// 预置累加器 1 为 10 秒。
SP T1	// 以脉冲定时器方式启动定时器 T1。
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	// 以二进制码格式装载定时器 T1 的当前时间值。
T MW10	
LC T1	// 以 BCD 格式装载定时器 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	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

该指令可以在 RLO 从“0”变为“1”时，启动寻址的定时器。

即使定时期间 RLO 变为“0”，编程设定的时间间隔也将继续计时。如果在编程设定的时间间隔到时前，RLO 从“0”变为“1”，则重新开始以编程设定的时间间隔进行计时。该定时器启动指令需要将时间值和时基以 BCD 码的格式保存在累加器 1 的低字中。

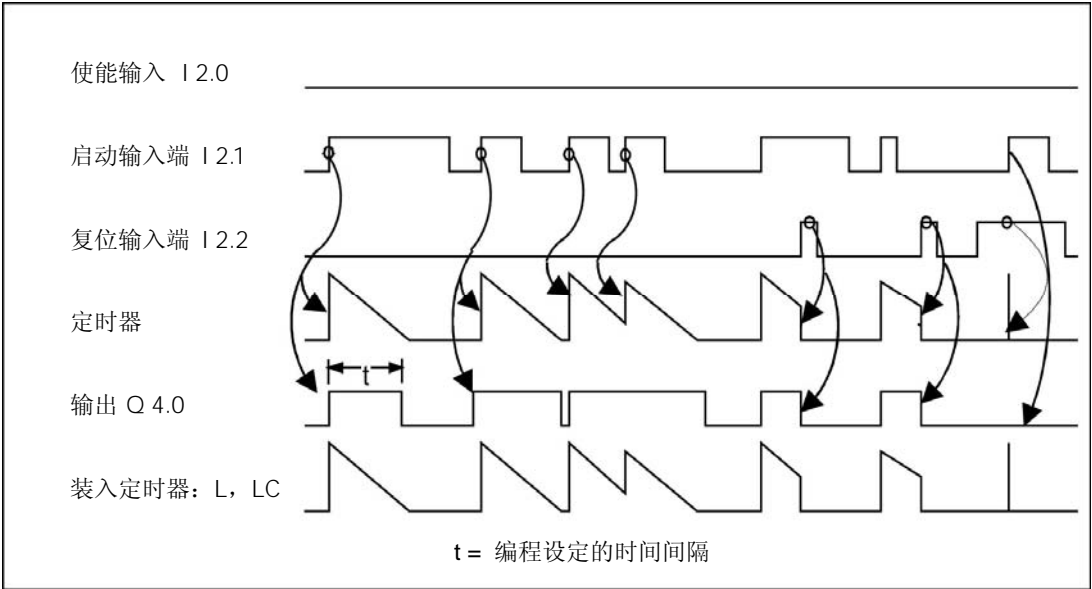
请参见“定时器在存储器中的位置和定时器组件”。

状态字

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

举例

STL	解 释
A I 2.0	
FR T1	// 使能定时器 T1。
A I 2.1	
L S5T#10s	// 预置累加器 1 为 10 秒。
SE T1	// 以扩展脉冲定时器方式启动定时器 T1。
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	// 以二进制码格式装载定时器 T1 的当前时间值。
T MW10	
LC T1	// 以 BCD 格式装载定时器 T1 的当前时间值。
T MW12	



12.9 SD 延时接通定时器

格式

SD <定时器>

地 址	数据类型	存储区	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

该指令可以在 RLO 从“0”变为“1”时，启动寻址的定时器。

只要 RLO = 1，即开始以编程设定的时间进行计时。如果在编程设定的时间间隔到时前，RLO 变为“0”，则定时器停止。该定时器启动指令需要将时间值和时基以 BCD 码的格式保存在累加器 1 的低字中。

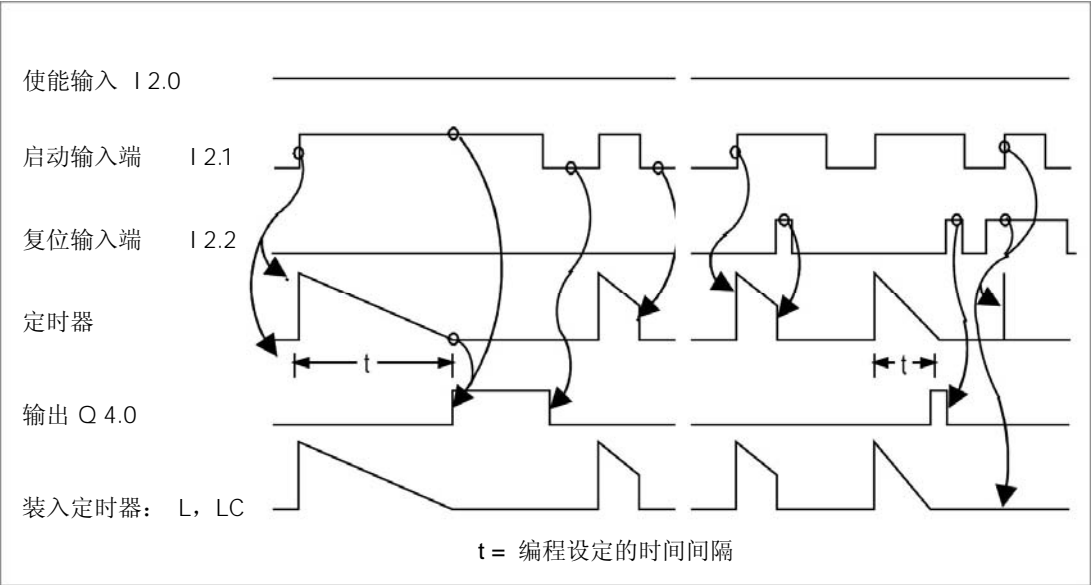
请参见“定时器在存储器中的位置和定时器组件”。

状态字

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

举例

STL	解 释
A I 2.0	
FR T1	// 使能定时器 T1。
A I 2.1	
L S5T#10s	// 预设累加器 1 为 10 秒。
SD T1	// 以延时接通定时器方式启动定时器 T1。
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	// 以二进制码格式装载定时器 T1 的当前时间值。
T MW10	
LC T1	// 以 BCD 格式装载定时器 T1 的当前时间值。
T MW12	



12.10 SS 保持型延时接通定时器

格式

SS <定时器>

地 址	数据类型	存储区	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

该指令可以在 RLO 从“0”变为“1”时，启动寻址的定时器。即使定时期间 RLO 变为“0”，定时器也将以编程设定的时间间隔进行计时。如果在编程设定的时间间隔到时前，RLO 从“0”变为“1”，则以编程设定的时间间隔重新开始计时。该定时器启动指令需要将时间值和时基以 BCD 码的格式保存在累加器 1 的低字中。

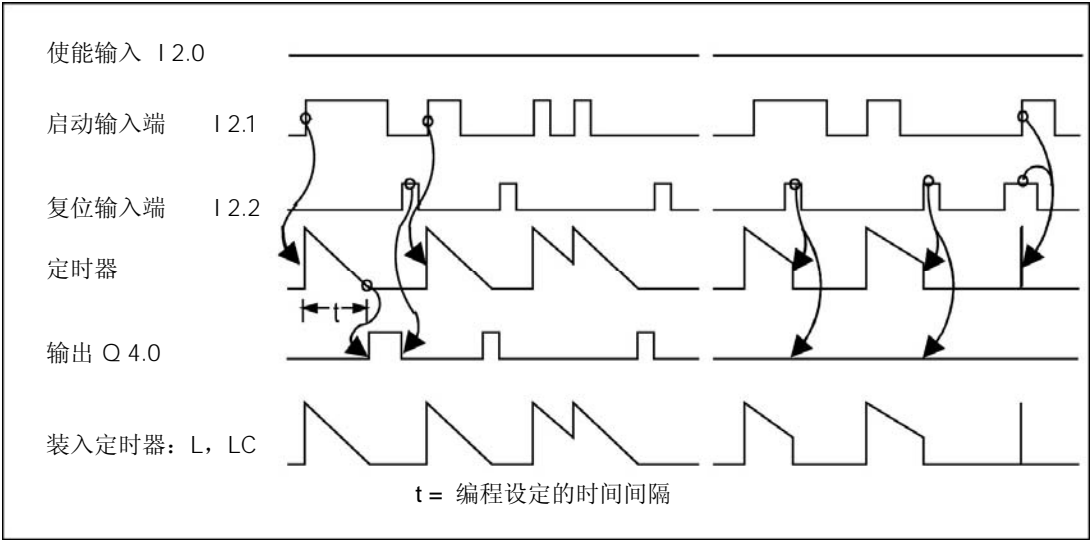
请参见“定时器在存储器中的位置和定时器组件”。

状态字

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

举例

STL	解 释
A I 2.0	
FR T1	// 使能定时器 T1。
A I 2.1	
L S5T#10s	// 预设累加器 1 为 10 秒。
SS T1	// 以保持型延时接通定时器方式启动定时器 T1。
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	// 以二进制码格式装载定时器 T1 的当前时间值。
T MW10	
LC T1	// 以 BCD 格式装载定时器 T1 的当前时间值。
T MW12	



12.11 SF 延时断开定时器

格式

SF <定时器>

地 址	数据类型	存储区	说 明
<定时器>	TIMER	T	定时器编号、范围与 CPU 有关

指令说明

该指令可以在 RLO 从“1”变为“0”时，启动寻址的定时器。

只要 RLO = 0，即以编程设定的时间进行计时。如果在编程设定的时间间隔到时前 RLO 变为“1”，则定时器停止计时。该定时器启动指令需要将时间值和时基以 BCD 码的格式保存在累加器 1 的低字中。

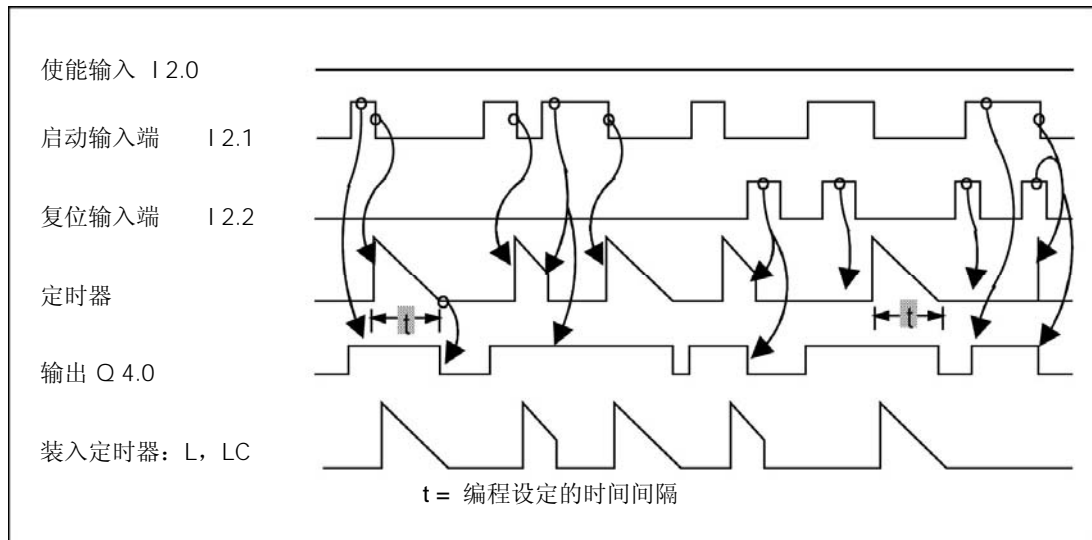
请参见“定时器在存储器中的位置和定时器组件”。

状态字

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

举例

STL	解 释
A I 2.0	
FR T1	// 使能定时器 T1。
A I 2.1	
L S5T#10s	// 预设累加器 1 为 10 秒。
SF T1	// 以延时断开定时器方式启动定时器 T1。
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	// 以二进制码格式装载定时器 T1 的当前时间值。
T MW10	
LC T1	// 以 BCD 格式装载定时器 T1 的当前时间值。
T MW12	



13 字逻辑指令

13.1 字逻辑指令概述

说明

字逻辑指令按照布尔逻辑将成对的字（16 位）和双字（32 位）逐位进行比较。每个字或双字都必须分别存放在 2 个累加器之一中。

对于字，其实是把累加器 2 低字的内容与累加器 1 低字的内容进行逻辑运算。结果保存在累加器 1 的低字中，同时覆盖原内容。

对于双字，其实是把累加器 2 的内容与累加器 1 的内容进行逻辑运算。结果保存在累加器 1 中，同时覆盖原内容。

如果结果不等于“0”，则状态字的位 CC 1 被置为“1”。如果结果等于“0”，则状态字的位 CC 1 被置为“0”。

下述字逻辑指令可供使用：

- AW 字“与”（16 位）
- OW 字“或”（16位）
- XOW 字“异或”（16位）
- AD 双字“与”（32 位）
- OD 双字“或”（32位）
- XOD 双字“异或”（32位）

13.2 AW 字“与”(16 位)

格式

AW
AW <常数>

地 址	数据类型	说 明
<常数>	WORD, 16 位常数	与累加器 1 低字内容按位进行“与”运算

指令说明

该指令可以根据布尔逻辑运算“与”关系，将累加器 1 的低字内容与累加器 2 的低字内容或一个 16 位常数逐位进行逻辑运算。只有进行逻辑运算的两个字的相应位都为“1”，结果的对应位才为“1”。结果保存在累加器 1 的低字中。累加器 1 的高字和累加器 2（以及累加器 3 和累加器 4，对于具有 4 个累加器的 CPU）保持不变。状态位 CC 1 被置为运算结果（如果结果不等于“0”，则 CC 1 = 1）。状态位 CC 0 和 OV 被置为“0”。

AW：累加器 1 低字与累加器 2 低字进行逻辑运算。
AW <常数>：累加器 1 与 16 位常数进行逻辑运算。

状态字

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

举例

位	15...0
AW 执行之前累加器 1 的低字内容	0101	1001	0011	1011
累加器 2 低字或 16 位常数：	1111	0110	1011	0101
AW 执行之后的结果（累加器 1 低字）	0101	0000	0011	0001

举例 1

STL	解 释
L IW20	// 将 IW20 的内容装入累加器 1 低字。
L IW22	// 将累加器 1 中的内容装入累加器 2 中。将 IW22 的内容装入累加器 1 低字中。
AW	// 将累加器 1 低字中的内容与累加器 2 低字中的内容进行“与”运算；结果保存到累加器 1 低字中。
T MW 8	// 将结果传送到 MW8。

举例 2

STL	解 释
L IW20	// 将 IW20 的内容装入累加器 1 低字。
AW W#16#0FFF	// 将累加器 1 低字中的内容与 16 位常数(0000_1111_1111_1111)进行“与”运算；结果保存到累加器 1 低字中。
JP NEXT	// 如果结果不等于“0”(CC 1 = 1)，则跳转到 NEXT 跳转标号。

13.3 OW 字“或”(16 位)

格式

OW

OW <常数>

地 址	数据类型	说 明
<常数>	WORD, 16 位常数	与累加器 1 低字内容按位进行“或”运算

指令说明

该指令可以根据布尔逻辑运算“或”关系，将累加器 1 的低字内容与累加器 2 的低字内容或一个 16 位常数逐位进行逻辑运算。只有进行逻辑运算的两个字中至少有一个的相应位为“1”，结果字的对应位才为“1”。结果保存在累加器 1 的低字中。累加器 1 的高字和累加器 2（以及累加器 3 和累加器 4，对于具有 4 个累加器的 CPU）保持不变。执行指令与 RLO 无关，而且对 RLO 没有影响。状态位 CC 1 被置为运算结果（如果结果不等于“0”，则 CC 1 = 1）。状态字位 CC 0 和 OV 被置为“0”。

OW：累加器 1 低字与累加器 2 低字进行逻辑运算。

OW <常数>：累加器 1 低字与 16 位常数进行逻辑运算。

状态字

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

举例

位	15...0
OW 执行之前累加器 1 的低字内容	0101	0101	0011	1011
累加器 2 低字或 16 位常数：	1111	0110	1011	0101
OW 执行之后的结果（累加器 1 低字）	1111	0111	1011	1111

举例 1

STL	解 释
L IW20	// 将 IW20 的内容装入累加器 1 低字。
L IW22	// 将累加器 1 中的内容装入累加器 2 中。将输入字 IW22 的内容装入累加器 1 低字中。
OW	// 将累加器 1 低字中的内容与累加器 2 低字中的内容进行“或”运算；结果保存到累加器 1 低字中。
T MW8	// 将结果传送到 MW8。

举例 2

STL	解 释
L IW20	// 将 IW20 的内容装入累加器 1 低字。
OW W#16#0FFF	// 将累加器 1 低字中的内容与 16 位常数(0000_1111_1111_1111)进行“或”运算；结果保存到累加器 1 低字中。
JP NEXT	// 如果结果不等于“0”（CC 1 = 1），则跳转到 NEXT 跳转标号。

13.4 XOW 字“异或”(16位)

格式

XOW
XOW <常数>

地 址	数据类型	说 明
<常数>	WORD, 16 位常数	与累加器 1 低字内容按位进行“异或”运算

指令说明

该指令根据布尔逻辑运算“异或”关系，将累加器 1 的低字内容与累加器 2 的低字内容或一个 16 位常数逐位进行逻辑运算。当进行逻辑运算的两个字中相对应的位只有一个为“1”，则结果字中的对应位才为“1”。结果保存在累加器 1 的低字中。累加器 1 的高字和累加器 2 保持不变。状态位 CC 1 被置为运算结果（如果结果不等于“0”，则 CC 1 = 1）。状态字位 CC 0 和 OV 被置为“0”。可多次使用异或运算函数。如果选中地址有奇数个“1”，则逻辑运算结果为“1”。
XOW：累加器 1 低字与累加器 2 低字进行逻辑运算。
XOW <常数>：累加器 1 低字与 16 位常数进行逻辑运算。

状态字

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

举例

位	15...0
XOW 执行之前累加器 1 的内容	0101	0101	0011	1011
累加器 2 低字或 16 位常数:	1111	0110	1011	0101
XOW 执行之后的结果 (累加器 1)	1010	0011	1000	1110

举例 1

STL	解 释
L IW20	// 将 IW20 的内容装入累加器 1 低字。
L IW22	// 将累加器 1 中的内容装入累加器 2 中。将 IW24 的内容装入累加器 1 低字中。
XOW	// 将累加器 1 低字中的内容与累加器 2 低字中的内容进行“异或”运算；结果保存到累加器 1 低字中。
T MW8	// 将结果传送到 MW8。

举例 2

STL	解 释
L IW20	// 将 IW20 的内容装入累加器 1 低字。
XOW16#0FFF	// 将累加器 1 低字中的内容与 16 位常数 (0000_1111_1111_1111) 进行“异或”运算；结果保存到累加器 1 低字中。
JP NEXT	// 如果结果不等于“0” (CC 1 = 1)，则跳转到 NEXT 跳转标号。

13.5 AD 双字“与”(32 位)

格式

AD

AD <常数>

地 址	数据类型	说 明
<常数>	DWORD, 32 位常数	与累加器 1 的内容按位进行“与”运算

指令说明

该指令根据布尔逻辑运算“与”关系，将累加器 1 的内容与累加器 2 的内容或一个 32 位常数逐位进行逻辑运算。只有进行逻辑运算的两个双字的相应位都为“1”，结果双字的对应位才为“1”。结果保存在累加器 1 中。累加器 2（以及累加器 3 和累加器 4，对于具有 4 个累加器的 CPU）保持不变。状态位 CC 1 被置为运算结果（如果结果不等于“0”，则 CC 1 = 1）。状态字位 CC 0 和 OV 被置为“0”。

AD: 累加器 1 与累加器 2 进行逻辑运算。

AD <常数>: 累加器 1 与 32 位常数进行逻辑运算。

状态字

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

举例

位	31...0
UD 执行之前累加器 1 的内容	0101	0000	1111	1100	1000	1001	0011	1011	
累加器 2 或 32 位常数	1111	0011	1000	0101	0111	0110	1011	0101	
UD 执行之后的结果（累加器 1）	0101	0000	1000	0100	0000	0000	0011	0001	

举例 1

STL	解 释
L ID20	// 将 ID20 的内容装入累加器 1。
L ID24	// 将累加器 1 中的内容装入累加器 2 中。将 ID24 的内容装入累加器 1 中。
AD	// 将累加器 1 中的内容与累加器 2 中的内容进行“与”运算；结果保存到累加器 1 中。
T MD8	// 将结果传送到 MD8。

举例 2

STL	解 释
L ID 20	// 将 ID20 的内容装入累加器 1。
AD DW#16#0FFF_EF21	// 将累加器 1 中的内容与 32 位常数（0000_1111_1111_1110_1111_0010_0001）进行“与”运算；结果保存到累加器 1 中。
JP NEXT	// 如果结果不等于“0”（CC 1 = 1），则跳转到 NEXT 跳转标号。

13.6 OD 双字“或”(32位)

格式

OD
OD <常数>

地 址	数据类型	说 明
<常数>	DWORD, 32 位常数	与累加器 1 的内容按位进行“或”运算

指令说明

该指令可以根据布尔逻辑运算“或”关系，将累加器 1 的内容与累加器 2 的内容或一个 32 位常数逐位进行逻辑运算。只有进行逻辑运算的两个双字中至少有一个的相应位为“1”，结果双字的对应位才为“1”。结果保存在累加器 1 中。累加器 2（以及累加器 3 和累加器 4，对于具有 4 个累加器的 CPU）保持不变。状态位 CC 1 被置为运算结果（如果结果不等于“0”，则 CC 1 = 1）。状态字位 CC 0 和 OV 被置为“0”。

OD: 累加器 1 与累加器 2 进行逻辑运算。
OD <常数>: 累加器 1 与 32 位常数进行逻辑运算。

状态字

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

举例

位	31...0
OD 执行之前累加器 1 的内容	0101	0000	1111	1100	1000	0101	0011	1011
累加器 2 或 32 位常数	1111	0011	1000	0101	0111	0110	1011	0101
OD 执行之后的结果（累加器 1）	1111	0011	1111	1101	1111	0111	1011	1111

举例 1

STL	解 释
L ID20	// 将 ID20 的内容装入累加器 1。
L ID24	// 将累加器 1 中的内容装入累加器 2 中。将输入双字 ID24 的内容装入累加器 1 中。
OD	// 将累加器 1 中的内容与累加器 2 中的内容进行“或”运算；结果保存到累加器 1 中。
T MD8	// 将结果传送到存储双字 MD8。

举例 2

STL	解 释
L ID20	// 将 ID20 的内容装入累加器 1。
OD DW#16#0FFF_EF21	// 将累加器 1 中的内容与 32 位常数 (0000_1111_1111_1110_1111_0010_0001) 进行“或”运算；结果保存到累加器 1 中。
JP NEXT	// 如果结果不等于“0”(CC 1 = 1)，则跳转到 NEXT 跳转标号。

13.7 XOD 双字“异或”(32位)

格式

XOD
XOD <常数>

地 址	数据类型	说 明
<常数>	DWORD, 32 位常数	与累加器 1 内容按位进行“异或”运算

指令说明

该指令根据布尔逻辑运算“异或”关系，将累加器 1 的内容与累加器 2 的内容或一个 32 位常数逐位进行逻辑运算。当进行逻辑运算的两个双字中相对应的位只有一个为“1”，则结果双字中的对应位才为“1”。结果保存在累加器 1 中。累加器 2 保持不变。状态位 CC 1 被置为运算结果（如果结果不等于“0”，则 CC 1 = 1）。状态字位 CC 0 和 OV 被置为“0”。

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

XOD: 累加器 1 与累加器 2 进行逻辑运算。

XOD <常数>: 累加器 1 与 32 位常数进行逻辑运算。

状态字

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

举例

位	31...0
XOD 执行之前累加器 1 的内容	0101	0000	1111	1100	1000	0101	0011	1011
累加器 2 或 32 位常数	1111	0011	1000	0101	0111	0110	1011	0101
XOD 执行之后的结果（累加器 1）	1010	0011	0111	1001	1111	0011	1000	1110

举例 1

STL	解 释
L ID20	// 将 ID20 的内容装入累加器 1。
L ID24	// 将累加器 1 中的内容装入累加器 2 中。将 ID24 的内容装入累加器 1 中。
XOD	// 将累加器 1 中的内容与累加器 2 中的内容进行“异或”运算；结果保存到累加器 1 中。
T MD8	// 将结果传送到 MD8。

举例 2

STL	解 释
L ID20	// 将 ID20 的内容装入累加器 1。
XOD DW#16#0FFF_EF21	// 将累加器 1 中的内容与 32 位常数（0000_1111_1111_1111_1110_0010_0001）进行“异或”运算；结果保存到累加器 1 中。
JP NEXT	// 如果结果不等于“0”（CC 1 = 1），则跳转到 NEXT 跳转标号。

14 累加器操作指令

14.1 累加器和地址寄存器操作指令概述

说明

下列指令可用于处理一个或两个累加器：

- TAK 累加器 1 与累加器 2 进行互换
- PUSH 带有两个累加器的 CPU
- PUSH 带有四个累加器的 CPU
- POP 带有两个累加器的 CPU
- POP 带有四个累加器的 CPU

- ENT 进入累加器堆栈
- LEAVE 离开累加器堆栈
- INC 增加累加器 1 低字的低字节内容
- DEC 减少累加器 1 低字的低字节内容

- +AR1 累加器 1 加到地址寄存器 1
- +AR2 累加器 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 累加器 1 与累加器 2 进行互换

格式

TAK

说明

该指令交换累加器 1 和累加器 2 中的内容。指令的执行与状态位无关，而且对状态位没有影响。对于具有四个累加器的 CPU，累加器 3 和累加器 4 的内容保持不变。

状态字

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

例如：从较大值中减去较小值

STL	解 释
L MW10	// 将 MW10 的内容装入累加器 1 低字。
L MW12	// 将累加器 1 低字中的内容装入累加器 2 低字中。将 MW12 的内容装入累加器 1 低字。
>I	// 检查累加器 2 低字中的内容(MW10)是否大于累加器 1 低字中的内容(MW12)。
SPB NEXT	// 如果累加器 2 低字中的内容(MW10)大于累加器 1 低字中的内容(MW12)，则跳转到 NEXT 跳转标号。
TAK	// 交换累加器 1 和累加器 2 中的内容。
NEXT: -I	// 从累加器 1 低字的内容中减去累加器 2 低字的内容。
T MW14	// 将结果(=较大值减去较小值)传送到存储字 MW14。

内 容	累加器 1	累加器 2
执行 TAK 指令之前	<MW12>	<MW10>
执行 TAK 指令之后	<MW10>	<MW12>

14.3 POP 带有两个累加器的 CPU

格式

POP

说明

该指令将累加器 2 的全部内容复制到累加器 1。

累加器 2 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
T MD10	// 将累加器 1 中的内容 (= 数值 A) 传送到 MD10。
POP	// 复制累加器 2 的全部内容到累加器 1。
T MD14	// 将累加器 1 中的内容 (= 数值 B) 传送到 MD14。

内 容	累加器 1	累加器 2
执行 POP 指令之前	数值 A	数值 B
执行 POP 指令之后	数值 B	数值 B

14.4 POP 带有四个累加器的 CPU

格式

POP

说明

该指令将累加器 2 的全部内容复制到累加器 1，累加器 3 的内容复制到累加器 2，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。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
T MD10	// 将累加器 1 中的内容 (= 数值 A) 传送到 MD10。
POP	// 复制累加器 2 的全部内容到累加器 1。
T MD14	// 将累加器 1 中的内容 (= 数值 B) 传送到 MD14。

内 容	累加器 1	累加器 2	累加器 3	累加器 4
执行 POP 指令之前	数值 A	数值 B	数值 C	数值 D
执行 POP 指令之后	数值 B	数值 C	数值 D	数值 D

14.5 PUSH 带有两个累加器的 CPU

格式

PUSH

说明

该指令将累加器 1 的全部内容复制到累加器 2。
累加器 1 保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
L MW10	// 将 MW10 的内容装入累加器 1。
PUSH	// 复制累加器 1 的全部内容到累加器 2。

内 容	累加器 1	累加器 2
执行 PUSH 指令之前	<MW10>	<X>
执行 PUSH 指令之后	<MW10>	<MW10>

14.6 PUSH 带有四个累加器的 CPU

格式

PUSH

说明

该指令将累加器 3 的内容复制到累加器 4，累加器 2 的内容复制到累加器 3，累加器 1 的内容复制到累加器 2，累加器 1 的内容保持不变。指令的执行与状态位无关，而且对状态位没有影响。

状态字

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

举例

STL	解 释
L MW10	// 将 MW10 的内容装入累加器 1。
PUSH	// 将累加器 1 的内容复制到累加器 2，累加器 2 的内容复制到累加器 3，累加器 3 的内容复制到累加器 4。

内 容	累加器 1	累加器 2	累加器 3	累加器 4
执行 PUSH 指令之前	数值 A	数值 B	数值 C	数值 D
执行 PUSH 指令之后	数值 A	数值 A	数值 B	数值 C

14.7 ENT 进入累加器堆栈

格式

ENT

说明

该指令将累加器 3 的内容复制到累加器 4，累加器 2 的内容复制到累加器 3。如果直接在一个装载指令的前面编写 ENT 指令，可以将中间结果保存到累加器 3 中。

举例

STL	解 释
L DBD0	// 将 DBD0 的值装入累加器 1 中。(该值必须是浮点数格式)。
L DBD4	// 将累加器 1 中的数值复制到累加器 2。将 DBD4 的值装入累加器 1。 (该值必须是浮点数格式)。
+R	// 将累加器 1 和累加器 2 的内容以浮点数相加 (32 位, IEEE FP), 结果保存到累加器 1 中。
L DBD8	// 将累加器 1 中的数值复制到累加器 2。将 DBD8 的值装入累加器 1。
ENT	// 将累加器 3 中的内容装入累加器 4 中。将累加器 2 的内容 (中间结果) 复制到累加器 3 中。
L DBD12	// 将 DBD12 的值装入累加器 1 中。
-R	// 从累加器 2 的内容中减去累加器 1 的内容, 结果保存在累加器 1 中。 复制累加器 3 的内容到累加器 2。复制累加器 4 的内容到累加器 3。
/R	// 将累加器 2 的内容 (DBD0 + DBD4) 除以累加器 1 的内容 (DBD8 - DBD12)。结果保存在累加器 1 中。
T DBD16	// 将累加器 1 中的内容 (结果) 传送到 DBD16。

14.8 LEAVE 离开累加器堆栈

格式

LEAVE

说明

该指令将累加器 3 的内容复制到累加器 2，累加器 4 的内容复制到累加器 3。如果直接在一个移位或循环移位指令的前面编写 LEAVE 指令，则该指令类似于一个算术运算指令。累加器 1 和累加器 4 的内容保持不变。

14.9 INC 增加累加器 1 低字的低字节内容

格式

INC <8 位整数>

参 数	数据类型	说 明
<8 位整数>	8 位整数常数	累加器 1 低字低字节内容与常数相加， 常数范围为 0 – 255

说明

该指令将累加器 1 低字低字节中的内容与 8 位整数相加，结果保存在累加器 1 低字低字节中。累加器 1 低字高字节、累加器 1 高字和累加器 2 中的内容保持不变。指令的执行与状态位无关，而且对状态位没有影响。

注意

这些指令不适合 16 位和 32 位的算术运算，因为累加器 1 低字的低字节运算时不向高字节进位。如进行 16 位和 32 位的算术运算，请用 +I 或 +D 指令。

状态字

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

举例

STL	解 释
L MB22	// 装入 MB22 的值。
INC 1	// “将累加器 1（MB22）加 1”指令；结果保存到累加器 1 低字低字节中。
T MB22	// 将累加器 1 低字低字节中的内容（结果）传回到 MB22。

14.10 DEC 减少累加器 1 低字的低字节内容

格式

DEC <8位整数>

地 址	数据类型	说 明
<8 位整数>	8 位整数常数	累加器 1 低字低字节的内容减去常数，常数范围为 0 – 255

说明

该指令从累加器 1 低字低字节的内容中减去 8 位整数，结果保存在累加器 1 低字低字节中。累加器 1 低字高字节、累加器 1 高字和累加器 2 中的内容保持不变。指令的执行与状态位无关，而且对状态位没有影响。

注意

这些指令不适合 16 位和 32 位的算术运算，因为累加器 1 低字的低字节运算时不向高字节进位。如进行 16 位和 32 位的算术运算，请用 -I 或 -D 指令。

状态字

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

举例

STL	解 释
L MB250	// 装入 MB250 的值。
DEC 1	// “将累加器 1 低字低字节减 1” 指令；结果保存到累加器 1 低字低字节中。
T MB250	// 将累加器 1 低字低字节中的内容（结果）回传到 MB250。

14.11 +AR1 将累加器 1 加到地址寄存器 1

格式

+AR1
+AR1 <P#Byte.Bit>

参 数	数据类型	说 明
<P#Byte.Bit>	指针常数	加到地址寄存器 1 的地址

说明

该指令将语句中或累加器 1 低字中定义的偏移量加至地址寄存器 1。首先将整数（16 位）扩展为带有其正确符号的 24 位数，然后加到地址寄存器 1 的最低有效 24 位（地址寄存器 1 中部分相关地址）。地址寄存器 1 中 ID 区部分（位 24、25 和 26）保持不变。指令的执行与状态位无关，而且对状态位没有影响。

+AR1：要加到地址寄存器 1 中的整数（16 位）由累加器 1 低字中的数值定义。允许范围 -32768 - +32767。

+AR1 <P#Byte.Bit>：通过 <P#Byte.Bit> 地址定义要加上的偏移量。

状态字

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

举例 1

STL	解 释
L +300	// 将数值装入累加器 1 低字中。
+AR1	// 将累加器 1 低字中的内容（整数，16 位）加到地址寄存器 1。

举例 2

STL	解 释
+AR1 P#300.0	// 将偏移量 300.0 加到地址寄存器 1。

14.12 +AR2 将累加器 1 加到地址寄存器 2

格式

+AR2

+AR2 <P#Byte.Bit>

参 数	数据类型	说 明
<P#Byte.Bit>	指针常数	加到地址寄存器 2 的地址

说明

该指令将语句中或累加器 1 低字中定义的偏移量加至地址寄存器 2。首先将整数（16 位）扩展为带有其正确符号的 24 位数，然后加到地址寄存器 2 的最低有效 24 位（地址寄存器 2 中部分相关地址）。地址寄存器 2 中 ID 区部分（位 24、25 和 26）保持不变。指令的执行与状态位无关，而且对状态位没有影响。

+AR2：要加到地址寄存器 2 中的整数（16 位）由累加器 1 低字中的数值定义。允许范围 -32768 - +32767。

+AR2 <P#Byte.Bit>：通过 <P#Byte.Bit> 地址定义要加上的偏移量。

状态字

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

举例 1

STL	解 释
L +300	// 将数值装入累加器 1 低字中。
+AR2	// 将累加器 1 低字中的内容（整数，16 位）加到地址寄存器 2。

举例 2

STL	解 释
+AR2 P#300.0	// 将偏移量 300.0 加到地址寄存器 2。

14.13 BLD 程序显示指令（空）

格式

BLD <编号>

地 址	说 明
<编号>	BLD 指令的编号，范围为 0 - 255

说明

使用该指令（程序显示指令，空指令），既不执行任何功能，也不影响状态位。
该指令用于编程器（PG）的图形显示。当在语句表中显示 LAD 或 FBD 程序时，
可自动生成。地址 <编号> 是指 BLD 指令的标识号，由编程器产生。

状态字

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

14.14 NOP 0 空操作指令

格式

NOP 0

说明

NOP 0 指令（空操作指令 0）既不执行任何功能，也不影响状态位。指令代码含有一个 16 个“0”的位模式。该指令只用于编程器（PG）显示程序。

状态字

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

14.15 NOP 1 空操作指令

格式

NOP 1

说明

NOP 1 指令（空操作指令 1）既不执行任何功能，也不影响状态位。指令代码含有一个 16 个“1”的位模式。该指令只用于编程器（PG）显示程序。

状态字

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

A 所有语句表指令一览

A.1 按德文助记符分类的语句表指令

德文助记符	英文助记符	程序元素分类	说 明
+	+	整数算术运算指令	加上一个整数常数（16 位，32 位）
=	=	位逻辑指令	赋值
))	位逻辑指令	嵌套闭合
+AR1	+AR1	累加器指令	AR1 加累加器 1 至地址寄存器 1
+AR2	+AR2	累加器指令	AR2 加累加器 1 至地址寄存器 2
+D	+D	整数算术运算指令	作为双整数（32 位），将累加器 1 和 累加器 2 中的内容相加
-D	-D	整数算术运算指令	作为双整数（32 位），将累加器 2 中的内容减去累加器 1 中的内容
*D	*D	整数算术运算指令	作为双整数（32 位），将累加器 1 和 累加器 2 中的内容相乘
/D	/D	整数算术运算指令	作为双整数（32 位），将累加器 2 中的内容除以累加器 1 中的内容
? D	? D	比较指令	双整数（32 位）比较 ==, <>, >, <, >=, <=
+I	+I	整数算术运算指令	作为整数（16 位），将累加器 1 和累加器 2 中的内容相加
-I	-I	整数算术运算指令	作为整数（16 位），将累加器 2 中的内容减去累加器 1 中的内容
*I	*I	整数算术运算指令	作为整数（16 位），将累加器 1 和 累加器 2 中的内容相乘
/I	/I	整数算术运算指令	作为整数（16 位），将累加器 2 中的内容除以累加器 1 中的内容
? I	? I	比较指令	整数（16 位）比较 ==, <>, >, <, >=, <=
+R	+R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 1 和累加器 2 中的内容相加
-R	-R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 2 中的内容减去累加器 1 中的内容
*R	*R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 1 和累加器 2 中的内容相乘
/R	/R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 2 中的内容除以累加器 1 中的内容

德文助记符	英文助记符	程序元素分类	说 明
? R	? R	比较指令	比较两个浮点数（32 位）==, <>, >, <, >=, <=
ABS	ABS	浮点算术运算指令	浮点数取绝对值（32 位，IEEE-FP）
ACOS	ACOS	浮点算术运算指令	浮点数反余弦运算（32 位）
ASIN	ASIN	浮点算术运算指令	浮点数反正弦运算（32 位）
ATAN	ATAN	浮点算术运算指令	浮点数反正切运算（32 位）
AUF	OPN	数据块调用指令	打开数据块
BE	BE	程序控制指令	块结束
BEA	BEU	程序控制指令	无条件块结束
BEB	BEC	程序控制指令	条件块结束
BLD	BLD	程序控制指令	程序显示指令（空）
BTD	BTD	转换指令	BCD 转成整数（32 位）
BTI	BTI	转换指令	BCD 转成整数（16 位）
CALL	CALL	程序控制指令	块调用
CALL	CALL	程序控制指令	调用多背景块
CALL	CALL	程序控制指令	从库中调用块
CC	CC	程序控制指令	条件调用
CLR	CLR	位逻辑指令	RLO 清零（=0）
COS	COS	浮点算术运算指令	浮点余弦运算（32 位）
DEC	DEC	累加器指令	减少累加器 1 低字的低字节
DTB	DTB	转换指令	双整数（32 位）转成 BCD
DTR	DTR	转换指令	双整数（32 位）转成浮点数（32 位，IEEE-FP）
ENT	ENT	累加器指令	进入累加器栈
EXP	EXP	浮点算术运算指令	浮点数指数运算（32 位）
FN	FN	位逻辑指令	脉冲下降沿
FP	FP	位逻辑指令	脉冲上升沿
FR	FR	计数器指令	使能计数器（任意）（任意，FR C 0 – C 255）
FR	FR	定时器指令	使能定时器（任意）
INC	INC	累加器指令	增加累加器 1 低字的低字节
INVD	INVD	转换指令	对双整数求反码（32 位）
INVI	INVI	转换指令	对整数求反码（16 位）
ITB	ITB	转换指令	整数（16 位）转成 BCD
ITD	ITD	转换指令	整数（16 位）转成双整数（32 位）
L	L	装入/传送指令	装入
L DBLG	L DBLG	装入/传送指令	将共享数据块的长度装入累加器 1 中
L DBNO	L DBNO	装入/传送指令	将共享数据块的块号装入累加器 1 中
L DILG	L DILG	装入/传送指令	将背景数据块的长度装入累加器 1 中
L DINO	L DINO	装入/传送指令	将背景数据块的块号装入累加器 1 中
L STW	L STW	装入/传送指令	将状态字装入累加器 1

德文助记符	英文助记符	程序元素分类	说 明
L	L	装入/传送指令	将当前定时值作为整数装入累加器 1（当前定时值可以是 0 - 255 之间的一个数字，例如 L T 32）
L	L	装入/传送指令	将当前计数值装入累加器 1（当前计数值可以是 0 - 255 之间的一个数字，例如 L C 15）
LAR1	LAR1	装入/传送指令	将累加器 1 中的内容装入地址寄存器 1
LAR1	LAR1	装入/传送指令	将两个双整数（32 位指针）装入地址寄存器 1
LAR1	LAR1	装入/传送指令	将地址寄存器 2 的内容装入地址寄存器 1
LAR2	LAR2	装入/传送指令	将累加器 2 中的内容装入地址寄存器 1
LAR2	LAR2	装入/传送指令	将两个双整数（32 位指针）装入地址寄存器 2
LC	LC	计数器指令	将当前计数值作为 BCD 码装入累加器 1（当前计数值可以是 0 - 255 之间的一个数字，例如 LC C 15）
LC	LC	定时器指令	将当前定时值作为 BCD 码装入累加器 1（当前定时值可以是 0 - 255 之间的一个数字，例如 LC T 32）
LEAVE	LEAVE	累加器指令	离开累加器栈
LN	LN	浮点算术运算指令	浮点数自然对数运算（32 位）
LOOP	LOOP	跳转指令	循环
MCR(MCR(程序控制指令	将 RLO 存入 MCR 堆栈，开始 MCR
)MCR)MCR	程序控制指令	结束 MCR
MCRA	MCRA	程序控制指令	激活 MCR 区域
MCRD	MCRD	程序控制指令	去活 MCR 区域
MOD	MOD	整数算术运算指令	双整数形式的除法，其结果为余数（32 位）
NEGD	NEGD	转换指令	对双整数求补码（32 位）
NEGI	NEGI	转换指令	对整数求补码（16 位）
NEGR	NEGR	转换指令	对浮点数求反（32 位，IEEE-FP）
NOP 0	NOP 0	累加器指令	空指令
NOP 1	NOP 1	累加器指令	空指令
NOT	NOT	位逻辑指令	RLO 取反
O	O	位逻辑指令	“或”
O(O(位逻辑指令	“或”操作嵌套开始
OD	OD	字逻辑指令	双字“或”（32 位）
ON	ON	位逻辑指令	“或非”
ON(ON(位逻辑指令	“或非”操作嵌套开始
OW	OW	字逻辑指令	字“或”（16 位）
POP	POP	累加器指令	POP
POP	POP	累加器指令	带有两个累加器的 CPU

德文助记符	英文助记符	程序元素分类	说 明
POP	POP	累加器指令	带有四个累加器的 CPU
PUSH	PUSH	累加器指令	带有两个累加器的 CPU
PUSH	PUSH	累加器指令	带有四个累加器的 CPU
R	R	位逻辑指令	复位
R	R	计数器指令	复位计数器(当前计数值可以是 0-255 之间的一个数字, 例如 R C 15)
R	R	定时器指令	复位定时器(当前定时值可以是 0-255 之间的一个数字, 例如 R T 32)
RLD	RLD	移位和循环移位指令	双字循环左移 (32 位)
RLDA	RLDA	移位和循环移位指令	通过 CC 1 累加器 1 循环左移 (32 位)
RND	RND	转换指令	取整
RND+	RND+	转换指令	向上舍入为双整数
RND-	RND-	转换指令	向下舍入为双整数
RRD	RRD	移位和循环移位指令	双字循环右移 (32 位)
RRDA	RRDA	移位和循环移位指令	通过 CC 1 累加器 1 循环右移 (32 位)
S	S	位逻辑指令	置位
S	S	计数器指令	置位计数器(当前计数值可以是 0-255 之间的一个数字, 例如 S C 15)
SA	SF	定时器指令	延时断开定时器
SAVE	SAVE	位逻辑指令	把 RLO 存入 BR 寄存器
SE	SD	定时器指令	延时接通定时器
SET	SET	位逻辑指令	置位
SI	SP	定时器指令	脉冲定时器
SIN	SIN	浮点算术运算指令	浮点数正弦运算 (32 位)
SLD	SLD	移位和循环移位指令	双字左移 (32 位)
SLW	SLW	移位和循环移位指令	字左移 (16 位)
SPA	JU	跳转指令	无条件跳转
SPB	JC	跳转指令	若 RLO = 1, 则跳转
SPBB	JCB	跳转指令	若 RLO = 1 且 BR = 1, 则跳转
SPBI	JB	跳转指令	若 BR = 1, 则跳转
SPBIN	JNBI	跳转指令	若 BR = 0, 则跳转
SPBN	JCN	跳转指令	若 RLO = 0, 则跳转
SPBNB	JNB	跳转指令	若 RLO = 0 且 BR = 1, 则跳转
SPL	JL	跳转指令	跳转到标号
SPM	JM	跳转指令	若负, 则跳转
SPMZ	JMZ	跳转指令	若负或零, 则跳转
SPN	JN	跳转指令	若非零, 则跳转
SPO	JO	跳转指令	若 OV = 1, 则跳转
SPP	JP	跳转指令	若正, 则跳转
SPPZ	JPZ	跳转指令	若正或零, 则跳转
SPS	JOS	跳转指令	若 OS = 1, 则跳转

德文助记符	英文助记符	程序元素分类	说 明
SPU	JUO	跳转指令	若无效数，则跳转
SPZ	JZ	跳转指令	若零，则跳转
SQR	SQR	浮点算术运算指令	浮点数平方运算（32 位）
SQRT	SQRT	浮点算术运算指令	浮点数平方根运算（32 位）
SRD	SRD	移位和循环移位指令	双字右移（32 位）
SRW	SRW	移位和循环移位指令	字右移（16 位）
SS	SS	定时器指令	保持型延时接通定时器
SSD	SSD	移位和循环移位指令	移位有符号双整数（32 位）
SSI	SSI	移位和循环移位指令	移位有符号整数（16 位）
SV	SE	定时器指令	延时脉冲定时器
T	T	装入/传送指令	传送
T STW	T STW	装入/传送指令	将累加器 1 中的内容传送到状态字
TAD	CAD	转换指令	交换累加器 1 中的字节顺序（32 位）
TAK	TAK	累加器指令	累加器 1 与累加器 2 进行互换
TAN	TAN	浮点算术运算指令	浮点数正切运算（32 位）
TAR	CAR	装入/传送指令	交换地址寄存器 1 和地址寄存器 2 的内容
TAR1	TAR1	装入/传送指令	将地址寄存器 1 中的内容传送到累加器 1
TAR1	TAR1	装入/传送指令	将地址寄存器 1 的内容传送到目的地（32 位指针）
TAR1	TAR1	装入/传送指令	将地址寄存器 1 的内容传送到地址寄存器 2
TAR2	TAR2	装入/传送指令	将地址寄存器 2 中的内容传送到累加器 1
TAR2	TAR2	装入/传送指令	将地址寄存器 2 的内容传送到目的地（32 位指针）
TAW	CAW	转换指令	交换累加器 1 低字中的字节顺序（16 位）
TDB	CDB	转换指令	交换共享数据块和背景数据块
TRUNC	TRUNC	转换指令	截尾取整
U	A	位逻辑指令	“与”
U(A(位逻辑指令	“与”操作嵌套开始
UC	UC	程序控制指令	无条件调用
UD	AD	字逻辑指令	双字“与”（32 位）
UN	AN	位逻辑指令	“与非”
UN(AN(位逻辑指令	“与非”操作嵌套开始
UW	AW	字逻辑指令	字“与”（16 位）
X	X	位逻辑指令	“异或”
X(X(位逻辑指令	“异或”操作嵌套开始
XN	XN	位逻辑指令	“异或非”
XN(XN(位逻辑指令	“异或非”操作嵌套开始
XOD	XOD	字逻辑指令	双字“异或”（32 位）
XOW	XOW	字逻辑指令	字“异或”（16 位）

德文助记符	英文助记符	程序元素分类	说 明
ZR	CD	计数器指令	减计数器
ZV	CU	计数器指令	加计数器

A.2 按英文助记符分类的语句表指令（国际）

英文助记符	德文助记符	程序元素分类	说 明
+	+	整数算术运算指令	加上一个整数常数（16 位，32 位）
=	=	位逻辑指令	赋值
))	位逻辑指令	嵌套闭合
+AR1	+AR1	累加器指令	AR1 加累加器 1 至地址寄存器 1
+AR2	+AR2	累加器指令	AR2 加累加器 1 至地址寄存器 2
+D	+D	整数算术运算指令	作为双整数（32 位），将累加器 1 和 累加器 2 中的内容相加
-D	-D	整数算术运算指令	作为双整数（32 位），将累加器 2 中的内容减去累加器 1 中的内容
*D	*D	整数算术运算指令	作为双整数（32 位），将累加器 1 和 累加器 2 中的内容相乘
/D	/D	整数算术运算指令	作为双整数（32 位），将累加器 2 中的内容除以累加器 1 中的内容
? D	? D	比较指令	双整数（32 位）比较 ==, <>, >, <, >=, <=
+I	+I	整数算术运算指令	作为整数（16 位），将累加器 1 和 累加器 2 中的内容相加
-I	-I	整数算术运算指令	作为整数（16 位），将累加器 2 中的内容减去累加器 1 中的内容
*I	*I	整数算术运算指令	作为整数（16 位），将累加器 1 和 累加器 2 中的内容相乘
/I	/I	整数算术运算指令	作为整数（16 位），将累加器 2 中的内容除以累加器 1 中的内容
? I	? I	比较指令	整数（16 位）比较 ==, <>, >, <, >=, <=
+R	+R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 1 和累加器 2 中的内容相加
-R	-R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 2 中的内容减去累加器 1 中的内容
*R	*R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 1 和累加器 2 中的内容相乘
/R	/R	浮点算术运算指令	作为浮点数（32 位，IEEE-FP），将累加器 2 中的内容除以累加器 1 中的内容

英文助记符	德文助记符	程序元素分类	说 明
? R	? R	比较指令	比较两个浮点数（32 位）==, <>, >, <, >=, <=
A	U	位逻辑指令	“与”
A(U(位逻辑指令	“与”操作嵌套开始
ABS	ABS	浮点算术运算指令	浮点数取绝对值（32 位，IEEE-FP）
ACOS	ACOS	浮点算术运算指令	浮点数反余弦运算（32 位）
AD	UD	字逻辑指令	双字“与”（32 位）
AN	UN	位逻辑指令	“与非”
AN(UN(位逻辑指令	“与非”操作嵌套开始
ASIN	ASIN	浮点算术运算指令	浮点数反正弦运算（32 位）
ATAN	ATAN	浮点算术运算指令	浮点数反正切运算（32 位）
AW	UW	字逻辑指令	字“与”（16 位）
BE	BE	程序控制指令	块结束
BEC	BEB	程序控制指令	条件块结束
BEU	BEA	程序控制指令	无条件块结束
BLD	BLD	程序控制指令	程序显示指令（空）
BTB	BTB	转换指令	BCD 转成整数（32 位）
BTI	BTI	转换指令	BCD 转成整数（16 位）
CAD	TAD	转换指令	Change Byte Sequence in ACCU 1 (32-bit)
CALL	CALL	程序控制指令	块调用
CALL	CALL	程序控制指令	调用多背景块
CALL	CALL	程序控制指令	从库中调用块
CAR	TAR	装入/传送指令	交换地址寄存器 1 和地址寄存器 2 的内容
CAW	TAW	转换指令	Change Byte Sequence in ACCU 1-L (16-bit)
CC	CC	程序控制指令	条件调用
CD	ZR	计数器指令	减计数器
CDB	TDB	转换指令	交换共享数据块和背景数据块
CLR	CLR	位逻辑指令	RLO 清零 (=0)
COS	COS	浮点算术运算指令	浮点数余弦运算（32 位）
CU	ZV	计数器指令	加计数器
DEC	DEC	累加器指令	减少累加器 1 低字的低字节
DTB	DTB	转换指令	双整数（32 位）转成 BCD
DTR	DTR	转换指令	双整数（32 位）转成浮点数（32 位，IEEE-FP）
ENT	ENT	累加器指令	进入累加器栈
EXP	EXP	浮点算术运算指令	浮点数指数运算（32 位）
FN	FN	位逻辑指令	脉冲下降沿
FP	FP	位逻辑指令	脉冲上升沿
FR	FR	计数器指令	使能计数器（任意）（任意，FR C 0 - C 255）

英文助记符	德文助记符	程序元素分类	说 明
FR	FR	定时器指令	使能定时器（任意）
INC	INC	累加器指令	增加累加器 1 低字的低字节
INVD	INVD	转换指令	对双整数求反码（32 位）
INVI	INVI	转换指令	对整数求反码（16 位）
ITB	ITB	转换指令	整数（16 位）转成 BCD
ITD	ITD	转换指令	整数（16 位）转成双整数（32 位）
JB	SPBI	跳转指令	若 BR = 1，则跳转
JC	SPB	跳转指令	若 RLO = 1，则跳转
JCB	SPBB	跳转指令	若 RLO = 1 且 BR = 1，则跳转
JCN	SPBN	跳转指令	若 RLO = 0，则跳转
JL	SPL	跳转指令	跳转到标号
JM	SPM	跳转指令	若负，则跳转
JMZ	SPMZ	跳转指令	若负或零，则跳转
JN	SPN	跳转指令	若非零，则跳转
JNB	SPBNB	跳转指令	若 RLO = 0 且 BR = 1，则跳转
JNBI	SPBIN	跳转指令	若 BR = 0，则跳转
JO	SPO	跳转指令	若 OV = 1，则跳转
JOS	SPS	跳转指令	若 OS = 1，则跳转
JP	SPP	跳转指令	若正，则跳转
JPZ	SPPZ	跳转指令	若正或零，则跳转
JU	SPA	跳转指令	无条件跳转
JUO	SPU	跳转指令	若无效数，则跳转
JZ	SPZ	跳转指令	若零，则跳转
L	L	装入/传送指令	装入
L DBLG	L DBLG	装入/传送指令	将共享数据块的长度装入累加器 1 中
L DBNO	L DBNO	装入/传送指令	将共享数据块的块号装入累加器 1 中
L DILG	L DILG	装入/传送指令	将背景数据块的长度装入累加器 1 中
L DINO	L DINO	装入/传送指令	将背景数据块的块号装入累加器 1 中
L STW	L STW	装入/传送指令	将状态字装入累加器 1
L	L	定时器指令	将当前定时值作为整数装入累加器 1（当前定时值可以是 0 - 255 之间的一个数字，例如 L T 32）
L	L	计数器指令	将当前计数值装入累加器 1（当前计数值可以是 0 - 255 之间的一个数字，例如 L C 15）
LAR1	LAR1	装入/传送指令	将累加器 1 中的内容装入地址寄存器 1
LAR1 <D>	LAR1<D>	装入/传送指令	将两个双整数（32 位指针）装入地址寄存器 1
LAR1 AR2	LAR1 AR2	装入/传送指令	将地址寄存器 2 的内容装入地址寄存器 1
LAR2	LAR2	装入/传送指令	将累加器 2 中的内容装入地址寄存器 1

英文助记符	德文助记符	程序元素分类	说 明
LAR2 <D>	LAR2 <D>	装入/传送指令	将两个双整数（32 位指针）装入地址寄存器 2
LC	LC	计数器指令	将当前计数值作为 BCD 码装入累加器 1（当前计数值可以是 0 - 255 之间的一个数字，例如 LC C 15）
LC	LC	定时器指令	将当前定时值作为 BCD 码装入累加器 1（当前定时值可以是 0 - 255 之间的一个数字，例如 LC T 32）
LEAVE	LEAVE	累加器指令	离开累加器栈
LN	LN	浮点算术运算指令	浮点数自然对数运算（32 位）
LOOP	LOOP	跳转指令	循环
MCR(MCR(程序控制指令	将 RLO 存入 MCR 堆栈，开始 MCR
)MCR)MCR	程序控制指令	结束 MCR
MCRA	MCRA	程序控制指令	激活 MCR 区域
MCRD	MCRD	程序控制指令	去活 MCR 区域
MOD	MOD	整数算术运算指令	双整数形式的除法，其结果为余数（32 位）
NEGD	NEGD	转换指令	对双整数求补码（32 位）
NEGI	NEGI	转换指令	对整数求补码（16 位）
NEGR	NEGR	转换指令	对浮点数求反（32 位，IEEE-FP）
NOP 0	NOP 0	累加器指令	空指令
NOP 1	NOP 1	累加器指令	空指令
NOT	NOT	位逻辑指令	RLO 取反
O	O	位逻辑指令	“或”
O(O(位逻辑指令	“或”操作嵌套开始
OD	OD	字逻辑指令	双字“或”（32 位）
ON	ON	位逻辑指令	“或非”
ON(ON(位逻辑指令	“或非”操作嵌套开始
OPN	AUF	数据块调用指令	打开数据块
OW	OW	字逻辑指令	字“或”（16 位）
POP	POP	累加器指令	POP
POP	POP	累加器指令	带有两个累加器的 CPU
POP	POP	累加器指令	带有四个累加器的 CPU
PUSH	PUSH	累加器指令	带有两个累加器的 CPU
PUSH	PUSH	累加器指令	带有四个累加器的 CPU
R	R	位逻辑指令	复位
R	R	计数器指令	复位计数器（当前计数值可以是 0 - 255 之间的一个数字，例如 R C 15）
R	R	定时器指令	复位定时器（当前定时值可以是 0 - 255 之间的一个数字，例如 R T 32）
RLD	RLD	移位和循环移位指令	双字循环左移（32 位）
RLDA	RLDA	移位和循环移位指令	通过 CC 1 累加器 1 循环左移（32 位）

英文助记符	德文助记符	程序元素分类	说 明
RND	RND	转换指令	取整
RND-	RND-	转换指令	向下舍入为双整数
RND+	RND+	转换指令	向上舍入为双整数
RRD	RRD	移位和循环移位指令	双字循环右移（32 位）
RRDA	RRDA	移位和循环移位指令	通过 CC 1 累加器 1 循环右移（32 位）
S	S	位逻辑指令	置位
S	S	计数器指令	置位计数器（当前计数值可以是 0–255 之间的一个数字，例如 S C 15）
SAVE	SAVE	位逻辑指令	把 RLO 存入 BR 寄存器
SD	SE	定时器指令	延时接通定时器
SE	SV	定时器指令	延时脉冲定时器
SET	SET	位逻辑指令	置位
SF	SA	定时器指令	延时断开定时器
SIN	SIN	浮点算术运算指令	浮点数正弦运算（32 位）
SLD	SLD	移位和循环移位指令	双字左移（32 位）
SLW	SLW	移位和循环移位指令	字左移（16 位）
SP	SI	定时器指令	脉冲定时器
SQR	SQR	浮点算术运算指令	浮点数平方运算（32 位）
SQRT	SQRT	浮点算术运算指令	浮点数平方根运算（32 位）
SRD	SRD	移位和循环移位指令	双字右移（32 位）
SRW	SRW	移位和循环移位指令	字右移（16 位）
SS	SS	定时器指令	保持型延时接通定时器
SSD	SSD	移位和循环移位指令	移位有符号双整数（32 位）
SSI	SSI	移位和循环移位指令	移位有符号整数（16 位）
T	T	装入/传送指令	传送
T STW	T STW	装入/传送指令	将累加器 1 中的内容传送到状态字
TAK	TAK	累加器指令	累加器 1 与累加器 2 进行互换
TAN	TAN	浮点算术运算指令	浮点数正切运算（32 位）
TAR1	TAR1	装入/传送指令	将地址寄存器 1 中的内容传送到累加器 1
TAR1	TAR1	装入/传送指令	将地址寄存器 1 的内容传送到目的地（32 位指针）
TAR1	TAR1	装入/传送指令	将地址寄存器 1 的内容传送到地址寄存器 2
TAR2	TAR2	装入/传送指令	将地址寄存器 2 中的内容传送到累加器 1
TAR2	TAR2	装入/传送指令	将地址寄存器 2 的内容传送到目的地（32 位指针）
TRUNC	TRUNC	转换指令	截尾取整
UC	UC	程序控制指令	无条件调用
X	X	位逻辑指令	“异或”
X(X(位逻辑指令	“异或”操作嵌套开始
XN	XN	位逻辑指令	“异或非”

英文助记符	德文助记符	程序元素分类	说 明
XN(XN(位逻辑指令	“异或非”操作嵌套开始
XOD	XOD	字逻辑指令	双字“异或”(32 位)
XOW	XOW	字逻辑指令	字“异或”(16 位)

B 编程举例

B.1 编程举例概述

实际应用

每一个语句表指令都可触发一特定的操作。将这些指令组合成程序，就能够完成很多种类的自动化任务。本章提供语句表指令实际应用的如下例子：

- 使用位逻辑指令控制传送带
- 使用位逻辑指令检测传送带的运动方向
- 使用定时器指令生成时钟脉冲
- 使用计数器和比较指令监视存储空间
- 使用整数算术运算指令解题
- 设定加热炉的加热时间

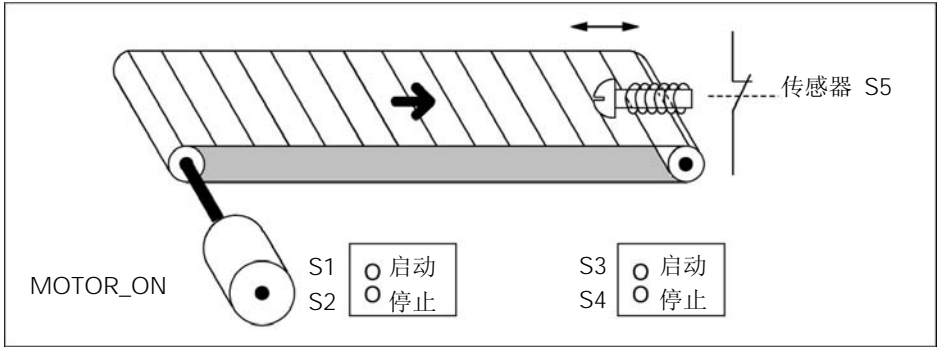
使用的指令

助记符	程序元素分类	说 明
AW	字逻辑指令	字“与”
OW	字逻辑指令	字“或”
CD, CU	计数器指令	减计数器，加计数器
S, R	位逻辑指令	置位，复位
NOT	位逻辑指令	RLO 取反
FP	位逻辑指令	脉冲上升沿
+I	整数算术运算指令	作为整数，将累加器 1 和 累加器 2 中的内容相加
/I	整数算术运算指令	作为整数，将累加器 2 的内容除以累加器 1 的内容
*I	整数算术运算指令	作为整数，将累加器 1 和 累加器 2 中的内容相乘
>=I, <=I	比较指令	整数比较
A, AN	位逻辑指令	“与”，“与非”
O, ON	位逻辑指令	“或”，“或非”
=	位逻辑指令	赋值
INC	累加器指令	加累加器 1
BE, BEC	程序控制指令	块结束和条件块结束
L, T	装入/传送指令	装入和传送
SE	定时器指令	延时脉冲定时器

B.2 例如：位逻辑指令

举例 1：控制传送带

下图所示为一个通过电气启动的传送带。在传送带的起点有两个按钮开关：用于启动的 S1 和用于停止的 S2。在传送带的尾端也有两个按钮开关：用于启动的 S3 和用于停止的 S4。可以从任一端启动或停止传送带。另外，当传送带上的物件到达末端时，传感器 S5 使传送带停机。



绝对地址编程和符号地址编程

可使用绝对地址或代表传送带系统不同部件的符号地址编写传送带控制程序。
你需要作一个符号表，使选择的符号与绝对地址相对应（参见“STEP 7 在线帮助”）。

系统部件	绝对地址	符 号	符号表
启动按钮开关	I 1.1	S1	I 1.1 S1
停止按钮开关	I 1.2	S2	I 1.2 S2
启动按钮开关	I 1.3	S3	I 1.3 S3
停止按钮开关	I 1.4	S4	I 1.4 S4
传感器	I 1.5	S5	I 1.5 S5
电机	Q 4.0	MOTOR_ON	Q 4.0 MOTOR_ON

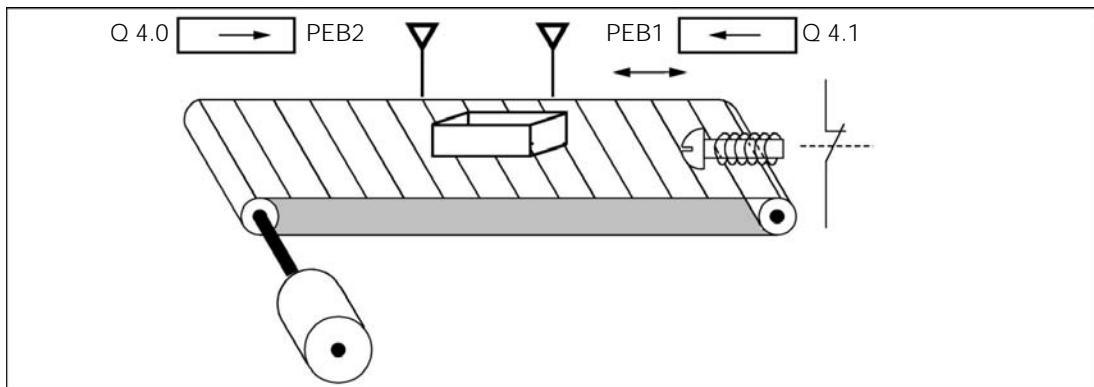
绝对地址编程		符号地址编程	
O	I 1.1	O	S1
O	I 1.3	O	S3
S	Q 4.0	S	MOTOR_ON
O	I 1.2	O	S2
O	I 1.4	O	S4
ON	I 1.5	ON	S5
R	Q 4.0	R	MOTOR_ON

控制传送带的语句表程序

STL	解 释
O I 1.1	// 按动一个启动开关，可以接通电机。
O I 1.3	
S Q 4.0	
O I 1.2	// 按动一个停止开关或打开传送带端部的常闭接点，可以切断电机。
O I 1.4	
ON I 1.5	
R Q 4.0	

举例 2：检测传送带的运动方向

下图所示为一个装配有两个光电传感器（PEB1 和 PEB2）的传送带，用来检测包裹在传送带上的移动方向。每一个光电传感器都可象常开接点一样使用。



绝对地址编程和符号地址编程

可使用代表传送带系统不同部件的绝对地址或符号地址编写传送带系统方向显示功能启动程序。

需要制作一个符号表，使选择的符号与绝对地址相对应（参见“STEP 7 在线帮助”）。

系统部件	绝对地址	符号	符号表
光电传感器 1	I 0.0	PEB1	I 0.0 PEB1
光电传感器 2	I 0.1	PEB2	I 0.1 PEB2
向右运动显示	Q 4.0	RIGHT	Q 4.0 RIGHT
向左运动显示	Q 4.1	LEFT	Q 4.1 LEFT
脉冲存储位 1	M 0.0	PMB1	M 0.0 PMB1
脉冲存储位 2	M 0.1	PMB2	M 0.1 PMB2

绝对地址编程		符号地址编程	
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	// 如果在输入 I 0.0 上信号状态从“0”变为“1”（上升沿），同时输入 I 0.1 的信号状态为“0”，则传送带上的包裹向左移动。
FP M 0.0	
AN I 0.1	
S Q 4.1	
A I 0.1	// 如果在输入 I 0.1 上信号状态从“0”变为“1”（上升沿），同时输入 I 0.0 的信号状态为“0”，则传送带上的包裹向右移动。如果有一个光电传感器被遮挡，这就意味着在光电传感器间有一个包裹。
FP M 0.1	
AN I 0.0	
S Q 4.0	
AN I 0.0	// 如果没有一个光电传感器被遮挡，则在光电传感器之间没有包裹。方向指示灯熄灭。
AN I 0.1	
R Q 4.0	
R Q 4.1	

B.3 例如：定时器指令

时钟脉冲发生器

当需要产生周期重复的信号时，可以使用时钟脉冲发生器或闪烁继电器。时钟脉冲发生器在信号系统是公用的，它可控制指示灯的闪烁。

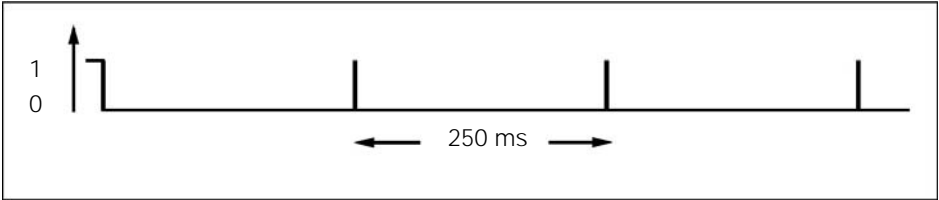
当使用 S7-300 时，能够使用专用组织块中的时间驱动处理来实现时钟脉冲发生器功能。下面以语句表程序表示的例子，来说明使用定时器功能来产生时钟脉冲。示例程序表示如何使用定时器实现自由设定时钟脉冲发生器功能。

生成时钟脉冲的语句表（脉冲占空比 1:1）

STL	解 释
U T1	// 如果定时器 T 1 计时已到，
L S5T#250ms	// 将时间值 250 ms 装入 T 1，并
SV T1	// 以延时脉冲定时器方式启动定时器 T1。
NOT	// 取反逻辑运算的结果。
BEB	// 如果定时器运行，则结束当前块。
L MB100	// 如果定时已到，装入存储字节 MB100 的内容，
INC 1	// 内容加 1，
T MB100	// 并将结果传送到 MB100。

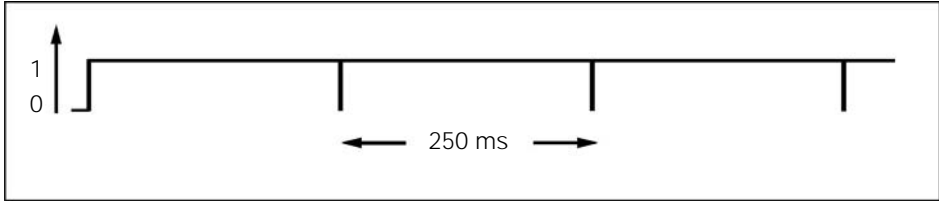
信号检查

定时器 T1 的信号检查生成以下逻辑运算结果（RLO）。



只要时间一结束，定时器就重新启动。因此，由语句 AN T1 进行的信号检查只短暂地生成信号状态“1”。

取反 RLO:



每 250 ms RLO 为“0”一次。然后 BEC 语句没有结束块处理。而是使存储字节 MB100 的内容加“1”。

存储字节 MB100 的内容每 250 ms 如下变化一次:

0 -> 1 -> 2 -> 3 -> ... -> 254 -> 255 -> 0 -> 1 ...

获得专用频率

从存储字节 MB100 的每个位，可以获得以下频率:

MB100 位	频率, [赫兹]	持续时间
M 100.0	2.0	0.5 s (250 ms 为“1”/ 250 ms 为“0”)
M 100.1	1.0	1 s (0.5 s 为“1” / 0.5 s 为“0”)
M 100.2	0.5	2 s (1 s 为“1” / 1 s 为“0”)
M 100.3	0.25	4 s (2 s 为“1” / 2 s 为“0”)
M 100.4	0.125	8s (4 s 为“1” / 4 s 为“0”)
M 100.5	0.0625	16s (8 s 为“1” / 8 s 为“0”)
M 100.6	0.03125	32s (16 s 为“1” / 16 s 为“0”)
M 100.7	0.015625	64s (32 s 为“1” / 32 s 为“0”)

语句表

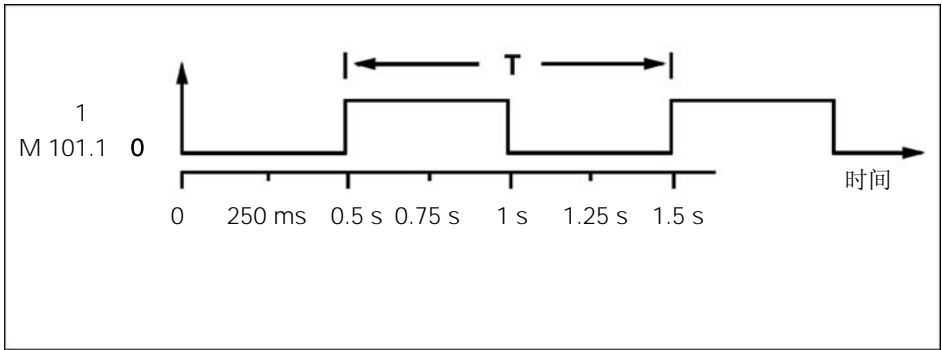
STL	解 释
A M10.0	// 出现故障时, M 10.0 = 1。当出现故障时, 故障指示灯以 1 Hz 频率闪烁。
A M100.1	
= Q 4.0	

存储字节 MB 101 的各位信号状态

扫描周期	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	时间值 [ms]
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 位 1 (M 101.1) 的信号状态

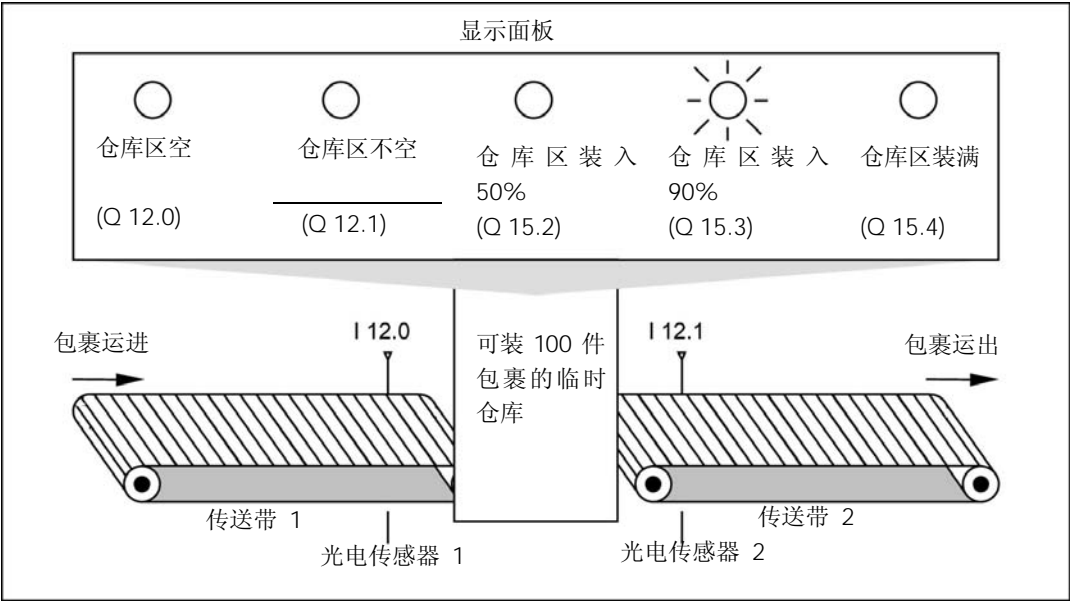
频率 = 1/T = 1/1 s = 1 Hz



B.4 例如：计数器和比较指令

装有计数器和比较器的仓库区

下图系统中包括了两台传送带，在两台传送带之间有一个临时仓库区。传送带 1 将包裹运送至仓库区。传送带 1 靠近仓库区一端安装的光电传感器确定已有多少包裹运送至仓库区。传送带 2 将临时库区中的包裹运送至装货场，在这里货物由卡车运送至顾客。传送带 2 靠近仓库区一端安装的光电传感器确定已有多少包裹从仓库区运送至装货场。含 5 个指示灯的显示面板表示临时仓库区的占用程度。



启动显示面板上指示灯的语句表程序

STL	解 释
A I 0.0	// 由光电传感器 1 产生的每个脉冲
CU C1	// 都可使计数器 C 1 的计数值加“1”，用以计算放入仓储区内的包裹数。
	//
A I 0.1	// 由光电传感器 2 产生的每个脉冲
CD C1	// 都可使计数器 C 1 的计数值减“1”，用以计算离开仓储区内的包裹数。
	//
AN C1	// 如果计数数值为“0”，
= Q 4.0	// 则“仓储区空”指示灯亮。
	//
A C1	// 如果计数数值不为“0”，
= A 4.1	// 则“仓储区不空”指示灯亮。
	//
L 50	
L C1	
<=I	// 如果计数数值大于或等于 50，
= Q 4.2	// 则“仓储区装满 50%”指示灯亮。
	//
L 90	
>=I	// 如果计数数值大于或等于 90，
= Q 4.3	// 则“仓储区装满 90%”指示灯亮。
	//
L Z1	
L 100	
>=I	// 如果计数数值大于或等于 100，
= Q 4.4	// 则“仓储区满”指示灯亮。（也可以使用输出 Q 4.4 锁定传送带 1）。

B.5 例如：整数算术运算指令

解决算术问题

以下示例程序告诉你如何使用 3 种整数算术运算指令产生如下列方程一样的结果：

$$MD4 = ((IW0 + DBW3) \times 15) / MW2$$

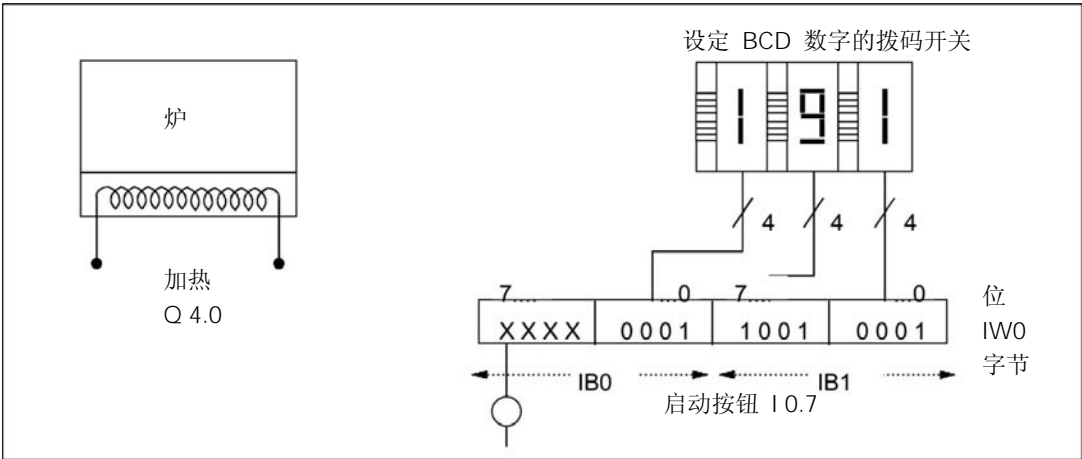
语句表

STL	解 释
L IW0	// 将输入字 IW0 的值装入累加器 1 中。
L DB5.DBW3	// 将 DB5 的共享数据字 DBW3 的值装入累加器 1。累加器 1 的原有内容移入累加器 2。
+I I 0.1	// 将累加器 1 和累加器 2 低字中的内容相加。结果保存在累加器 1 的低字中。累加器 2 和累加器 1 的高字内容保持不变。
L +15	// 将常数数值 +15 装入累加器 1。累加器 1 的原有内容移入累加器 2。
*I	// 将累加器 2 低字中的内容和累加器 1 低字中的内容相乘。结果保存在累加器 1 中。累加器 2 的内容保持不变。
L MW2	// 将存储字 MW2 的数值装入累加器 1。累加器 1 的原有内容移入累加器 2。
/I	// 将累加器 2 低字中的内容除以累加器 1 低字中的内容相乘。结果保存在累加器 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双字 MD4。两个累加器的内容保持不变。

B.6 例如：字逻辑指令

加热炉

操作员按启动按钮开始加热炉。操作员能够使用如图所示的拨码开关设定加热时间。操作员设定的值以二十进制（BCD）格式用[秒]为单位显示。



系统部件	绝对地址
启动按钮	I 0.7
个位数拨码开关	I 1.0 到 I 1.3
十位数拨码开关	I 1.4 到 I 1.7
百位数拨码开关	I 0.0 到 I 0.3
开始加热	Q 4.0

语句表

STL	解 释
A T1	// 如果定时器在计时，
= Q 4.0	// 则打开加热装置。
BEC	// 如果定时器在计时，则停止进一步处理。以防止定时器 T1 被按钮再次启动。
L IW0	
AW W#16#0FFF	// 屏蔽输入位 I 0.4 至 I 0.7（即将其复位为“0”）。 时间值（单位[秒]）以 BCD 码格式保存在累加器 1 低字。
OW W#16#2000	将时基（单位[秒]）写入累加器 1 低字的位 12 和位 13。
A I 0.7	
SE T1	// 如果按钮被按下，以延时脉冲定时器方式启动定时器 T1。

C 参数传递

块参数以数值的形式传送。对于功能块，被调用块使用背景数据块中的实际参数的复制值。对于功能，实际值的复制值存储在本地数据堆栈中。将不复制指针。在调用之前，将输入值(INPUT)复制到背景数据块或 L 堆栈中。调用以后，将输出值(OUTPUT)被复制回变量中。在被调用的块中，只能使用复制值。所需的 STL 指令位于调用块中并且用户不可见。

注：

如果将位、输入、输出或外设 I/O 存储区用作某一功能的实际地址，则对它们的处理方法将不同于对其它地址的处理方法。在此，直接对其刷新，而不是通过 L 堆栈进行刷新。



注意：

对被调用的块编程时，请确保编写被声明为 OUTPUT 的参数。否则，输出值将会是随机值！对于功能块，输出值将是上一次调用时来自背景数据块的值；对于功能，输出值将是位于 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 的值为“0”，则将会得到随机值。
 - 如果在块内跳转，请确保不要跳过任何编写了 OUTPUT 参数的地方。不要忘记 BEC 和 MCR 指令的作用。
-

