3.8 INVI 对整数求反码（16 位）

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	// **将结果传送到存储字 MW10。**

内 容	**累加器 1 低字**			
位	15...	..	..	...0
INVI **执行之前**	0110	0011	1010	1110
INVI **执行之后**	1001	1100	0101	0001

3.9 INVD 对双整数求反码（32 位）

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	// 将结果传送到存储双字 MD10。

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

3.10 NEGI 对整数求补码（16 位）

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 都设定为运算结果的一个功能。

状态字

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

状态字生成	CC 1	CC 0	OV	OS
结果 = 0	0	0	0	-
-32768 ≤ **结果** ≤ -1	0	1	0	-
32767 ≥ **结果** ≥ 1	1	0	0	-
结果 = 2768	0	1	1	1

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	// **将结果传送到存储字 MW10。**

内 容	**累加器 1 低字**			
位	15...	..	..	...0
NEGI **执行之前**	0101	1101	0011	1000
BEGI **执行之后**	1010	0010	1100	1000

3.11 NEGD 对双整数求补码（32 位）

格式

NEGD

说明

使用对双整数求补码指令（NEGD），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，而且对 RLO 没有影响。状态位 CC 1、CC 0、OS 和 OV 都设定为运算结果的一个功能。

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 ≤ **结果** ≤ -1	0	1	0	-
2,147,483,647 ≥ **结果** ≥ 1	1	0	0	-
结果 = 2,147,483,648	0	1	1	1

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	// **将结果传送到存储双字** MD10。

内 容	**累加器 1 高字**				**累加器 1 低字**			
位	31...	..	..	...16	15...	..	..	...0
NEGD **执行之前**	0101	1111	0110	0100	0101	1101	0011	1000
ITD **执行之后**	1010	0000	1001	1011	1010	0010	1100	1000
	（X = 0 **或 1，该位不用于转换**）							

3.12 NEGR 对浮点数求反（32 位，IEEE-FP）

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。该指令可转换累加器 1 中位 31 的信号状态（尾数的符号位）。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（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	// 将结果传送到存储双字 MD10（例如：结果 = -1.5E+02）。

3.13 CAW 交换累加器 1 低字中的字节顺序（16 位）

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	// **将存储字 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	// **将结果传送到存储字 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 **执行之前**	**数值** A	**数值** B	**数值** C	**数值** D
CAW **执行之后**	**数值** A	**数值** B	**数值** D	**数值** C

3.14 CAD 交换累加器 1 中的字节顺序（32 位）

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	// 将存储双字 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	// 将结果传送到存储双字 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 执行之前	数值 A	数值 B	数值 C	数值 D
CAD 执行之后	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 浮点数转换成为 32 位整数）将累加器 1 中的内容作为一个 32 位 IEEE 浮点数进行编译（32 位，IEEE-FP）。使用该指令，可以将 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。如果有数值超出这一范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置为“1”。结果保存在累加器 1 中。

出错时，将不进行转换，并指示溢出（利用 NaN 或无法表示为一个 32 位整数的浮点数）。

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 低字中。
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）传送到存储双字 MD20。

转换之前的数值		转换之后的数值
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 浮点数转换成为 32 位整数）将累加器 1 中的内容作为一个 32 位 IEEE 浮点数进行编译。使用该指令，可以将 32 位 IEEE 浮点数转换成为一个 32 位整数（双整数）。其结果为被转换浮点数的整数部分（IEEE 取整方式“截尾取整”）。如果有数值超出这一范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置为“1”。结果保存在累加器 1 中。

出错时，将不进行转换，并指示溢出（利用 NaN 或无法表示为一个 32 位整数的浮点数）。

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）传送到存储双字 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+ 取整为较大的双整数

格式

RND+

说明

RND+ 指令（32 位 IEEE 浮点数转换成为 32 位整数）将累加器 1 中的内容作为一个 32 位 IEEE 浮点数进行编译。使用该指令，可以将 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”）。如果有数值超出这一范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置为“1”。结果保存在累加器 1 中。

出错时，将不进行转换，并指示溢出（利用 NaN 或无法表示为一个 32 位整数的浮点数）。

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
T MD20	存在累加器 1 中。
	// 将结果（双整数）传送到存储双字 MD20。

转换之前的数值		转换之后的数值
MD10 = "100.5"	=> RND+ =>	MD20 = "+101"
MD10 = "-100.5"	=> RND+ =>	MD20 = "-100"

3.18 RND- 取整为较小的双整数

格式

RND-

说明

RND- 指令（32 位 IEEE 浮点数转换成为 32 位整数）将累加器 1 中的内容作为一个 32 位 IEEE 浮点数进行编译。使用该指令，可以将 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”）。如果有数值超出这一范围，则状态位 OV（溢出位）和 OS（存储溢出位）被置为“1”。结果保存在累加器 1 中。

出错时，将不进行转换，并指示溢出（利用 NaN 或无法表示为一个 32 位整数的浮点数）。

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 低字中。
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）传送到存储双字 MD20。

转换之前的数值		转换之后的数值
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。在你的 CPU 中可找到多少可用的计数器，请参考 CPU 技术数据。

计数器指令是访问计数器存储区的唯一功能。

通过使用以下计数器指令，可以在这一范围内改变计数值：

- 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 **复位计数器**
- S **计数器置位**
- 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，范围与** CPU **有关**

说明

当 RLO **从“** 0 **”变为“** 1 **”时，使用该指令，可以清零用于设置和选择寻址计数器的加计数或减计数的边沿检测标志。计数器置位或正常计数时不必使能计数器。这就意味着不管计数器置位、加计数器或减计数器的** RLO **是否恒为“** 1 **”，在执行之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，范围与 CPU 有关

说明

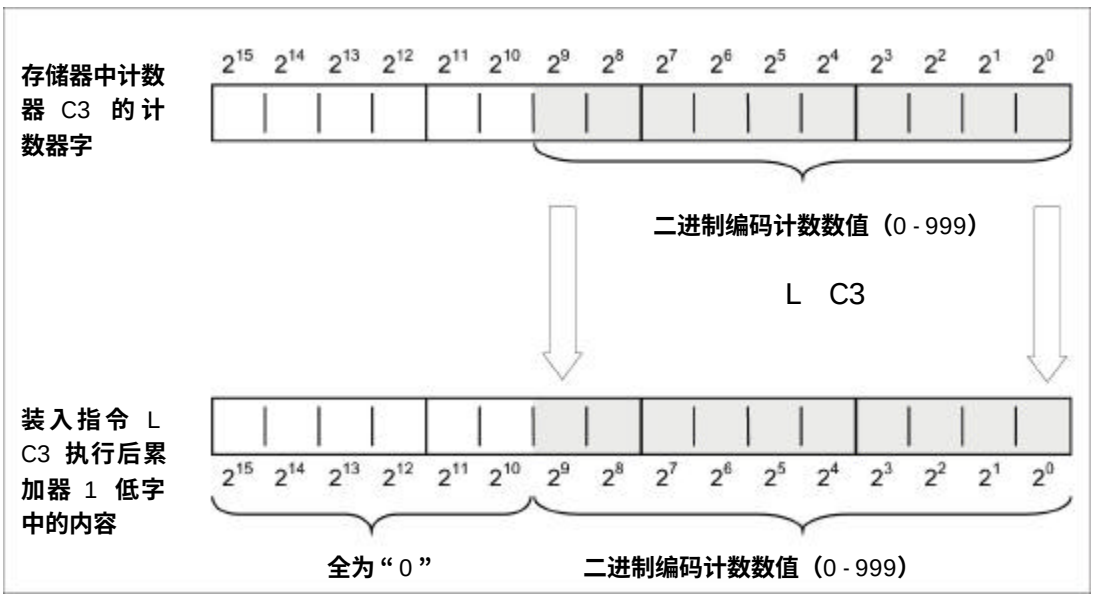
使用该指令，可以在累加器 1 的内容保存到累加器 2 中之后，将寻址计数器的当前计数作为一个整数装入累加器 1 的低字中。

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 将当前计数器值作为 BCD 码装入累加器 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，范围与 CPU 有关

说明

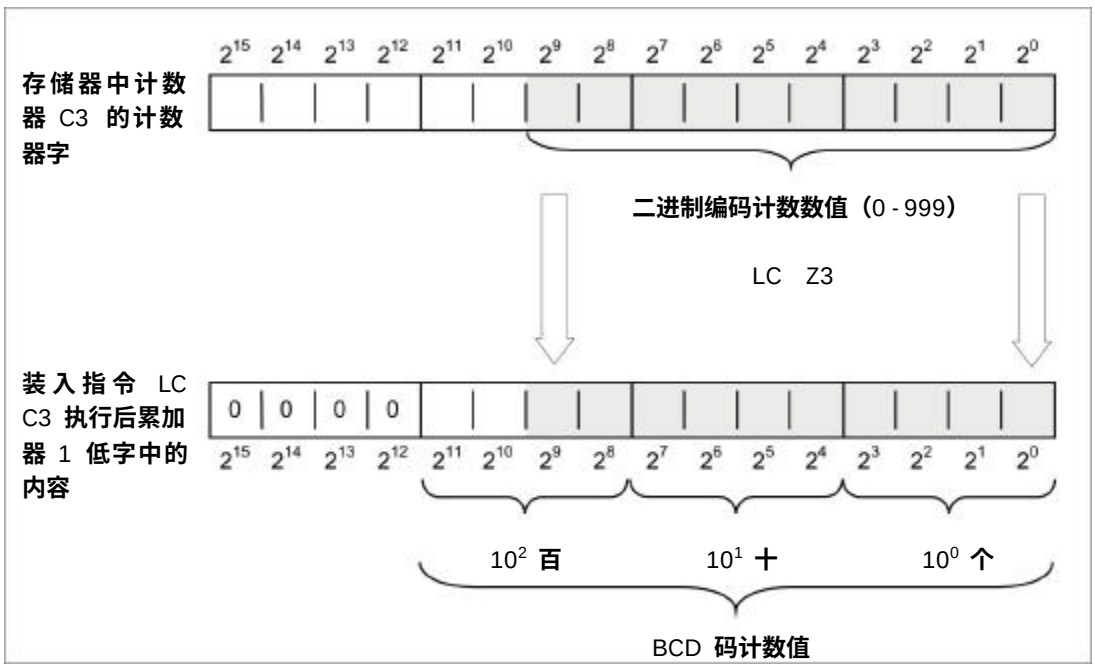
使用该指令，可以在累加器 1 的内容保存到累加器 2 中之后，将寻址计数器的计数作为一个 BCD 数装入累加器 1 中。

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十进制格式（BCD）装入累加器 1 低字。



4.5 R 复位计数器

格式

R <**计数器**>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	要复位计数器，范围与 CPU 有关

说明

使用该指令，可以在 RLO=1 **时，对寻址计数器进行复位。**

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，复位计数器** C3 **为“** 0 **”。**

4.6 S 计数器置位

格式

S <计数器>

地 址	数据类型	存储区	说 明
<计数器>	COUNTER	C	要复位计数器，范围与 CPU 有关

说明

使用该指令，可以在 RLO 从“0”变为“1”时，将计数从累加器 1 低字中装入计数器。累加器 1 中的计数必须是 0 –999 之间的一个 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，置数计数器 C1 为计数数值。

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，范围与** CPU **有关**

说明

使用该指令，可以在 RLO **从“** 0 **”变为“** 1 **”时，使寻址计数器的计数加“** 1 **”，并且计数小于“** 999 **”。当计数到达其上限“** 999 **”时，停止计数。其它** RLO **跳变没有影响；没有设置溢出（**OV**）。**

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，范围与 CPU 有关

说明

使用该指令，可以在 RLO 从“0”变为“1”时，使寻址计数器的计数减“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 跳变没有影响；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	// 计数器预置值。
A	I .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 的上升沿降计数一次。
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。一个程序自身同时可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 **将共享数据块的块号装入累加器 1 中**
- L DILG **将背景数据块的长度装入累加器 1 中**
- L DINO **将背景数据块的块号装入累加器 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，可以打开一个数据块作为共享数据块或背景数据块。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存储字** MW22**。**
OPN DI20	// **打开数据块** DB20 **作为背景数据块。**
L DIB12	// **将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

使用装入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            // 如果长度小于存储双字 MD10 中的数值，则跳转至 ERRO 跳转标号。
```

5.5 L DBNO 将共享数据块的块号装入累加器 1 中

格式

L DBNO

指令说明

使用装入共享数据块块号指令,可以在累加器 1 的内容保存到累加器 2 中之后，将共享数据块的块号装入累加器 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

使用装入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 D120	// 打开数据块 DB20 作为背景数据块。
L DILG	// 装入背景数据块的长度（DB20 的长度）。
L MW10	// 比较数据块的长度是否足够长。
<1	
JC	// 如果长度小于存储字 MW10 中的数值，则跳转至 ERRO 跳转标号。

5.7 L DINO 将背景数据块的块号装入累加器 1 中

格式

L DINO

指令说明

使用装入背景数据块块号指令,可以在累加器 1 的内容保存到累加器 2 中之后,将背景数据块的块号装入累加器 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

你可以使用跳转指令，来控制逻辑流，使能你的程序中断其线性流，重新从不同点开始扫描。你可以使用循环控制指令（LOOP），调用一个程序段多次。

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“：”，并且其后紧接语句。

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。

你可以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**

使用以下跳转指令，可以根据前一指令语句产生的逻辑运算结果（RLO），中断程序逻辑流：

- JC **若 RLO = 1，则跳转**
- JCN **若 RLO = 0，则跳转**
- JCB **若 RLO = 1， 则连同BR一起跳转**
- JNB **若 RLO = 0，则连同BR一起跳转**

使用以下跳转指令，可以根据状态字中的一个位的信号状态，中断程序逻辑流：

- 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**

使用以下跳转指令，可以根据一个计算的结果，中断程序逻辑流：

- 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　　**若正或零，则跳转**
- JMZ　　**若负或零，则跳转**
- JUO　　**若无效数，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>	跳转目的地的符号名

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的地，与状态字的内容无关。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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  I1.0
      A  I1.2
      JCDELE        // 如果 RLO=1，则跳转到跳转标号 DELE。
      L  MB10
      INC    1
      T  MB10
      JUFORW        // 无条件跳转到跳转标号 FORW。
DELE: L  0
      T  MB10
FORW: A  I2.1       // 在跳转到跳转标号 FORW 之后重新进行程序扫描。
```

6.3 JL 跳转到标号

格式

JL <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

使用该指令（通过跳转到列表跳转），可以使能多个编程的跳转操作。跳转目标列表最多有 255 个输入项，从该指令的下一行开始，到该指令地址中参考跳转标号的前一行结束。每一个跳转目的地都由一个无条件跳转指令（JU）组成。跳转目的地的数量（0 – 255）可以从累加器 1 低字的低字节中获得。

只要累加器的内容小于 JL 指令和跳转标号之间的跳转目的地的数量，JL 指令就跳转到 JU 指令之一。如果累加器 1 低字的低字节为“0”，则跳到第一个 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。如果跳转目的地的数量太大，则 JL 指令跳转到目的地列表中最后一个 JU 指令之后的第一个指令。

跳转目的地列表必须由位于 JL 指令地址中参考跳转标号前面的 JU 指令组成。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        // 将跳转目的地编号装入累加器 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，则跳转到目的地。
      JU SEG0       // 如果累加器 1 低字低字节中的内容等于 0，则跳转到目的地。
      JU SEG1       // 如果累加器 1 低字低字节中的内容等于 1，则跳转到目的地。
      JU COMM       // 如果累加器 1 低字低字节中的内容等于 2，则跳转到目的地。
      JU SEG3       // 如果累加器 1 低字低字节中的内容等于 3，则跳转到目的地。
LSTX: JU COMM
SEG0:    *          // 允许的指令
      JU COMM
SEG1:    *
      JU COMM       // 允许的指令
SEG3:    *
      JU COMM
COMM:    *          // 允许的指令。
```

6.4 JC 若 RLO = 1，则跳转

格式

JC <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果逻辑运算结果为“1”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（一个、两个或三个字语句）。

如果逻辑运算结果为“0”，则不执行跳转。RLO 被置为“1”，程序扫描从下一语句继续。

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，则跳转到跳转标号 JOVR。
      L    IW8      // 如果没有执行跳转，则继此继续程序扫描。
      T    MW22
JOVR: A    I 2.1    // 在跳转到跳转标号 JOVR 之后重新进行程序扫描。
```

6.5 JCN 若 RLO = 0，则跳转

格式

JCN <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果逻辑运算结果为“ 0”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（一个、两个或三个字语句）。

如果逻辑运算结果为“ 1”，则不执行跳转。程序扫描从下一语句继续。

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，则跳转到跳转标号 JOVR。
      L    IW8       // 如果没有执行跳转，则继此继续程序扫描。
      T    MW22
JOVR: A    I 2.1     // 在跳转到跳转标号 JOVR 之后重新进行程序扫描。
```

6.6 JCB 若 RLO = 1，则连同BR一起跳转

格式

JCB <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果逻辑运算结果为“1”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（一个、两个或三个字语句）。

如果逻辑运算结果为“0”，则不执行跳转。RLO 被置为“1”，程序扫描从下一语句继续。

RLO 被拷贝到该指令的 BR 中，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，则跳转到跳转标号 JOVR。将 RLO 位的内容复制到 BR 位。
      L     IW8      // 如果没有执行跳转，则继此继续程序扫描。
      T     MW22
JOVR: A     I 2.1    // 在跳转到跳转标号 JOVR 之后重新进行程序扫描。
```

6.7 JNB 若 RLO = 0，则连同BR一起跳转

格式

JNB <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果逻辑运算结果为“ 0”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（一个、两个或三个字语句）。

如果逻辑运算结果为“ 1”，则不执行跳转。RLO 被置为“ 1”，程序扫描从下一语句继续。

RLO 被拷贝到该指令的 BR 中，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，则跳转到跳转标号 JOVR。将 RLO 位的内容复制到 BR 位。
      L    IW8    // 如果没有执行跳转，则继此继续程序扫描。
      T    MW22
JOVR: A    I 2.1  // 在跳转到跳转标号 JOVR 之后重新进行程序扫描。
```

6.8 JBI 若 BR = 1，则跳转

格式

JBI <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果状态位 BR 为“1”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。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“:”，并且其后紧接语句。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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>	跳转目的地的符号名

说明

如果状态位 BR 为“0”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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>	跳转目的地的符号名

说明

如果状态位 OV 为“1”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（一个、两个或三个字语句）。与算术运算指令结合，使用该指令可以检查在每个单独的算术运算指令之后是否有溢出，以保证每个中间结果都在允许范围之内，或使用指令 JOS。

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               // 将存储字 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        // 如果没有执行跳转，则继此继续程序扫描。
      A    M 4.0
      R    M 4.0
      JU   NEXT
OVER: AN   M 4.0       // 在跳转到跳转标号 OVER 之后重新进行程序扫描。
      S    M 4.0
NEXT: NOP 0            // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.11 JOS 若 OS = 1，则跳转

格式

JOS <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果状态位 OS 为“1”，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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     // 如果在计算过程中三个指令之一中有溢出（OS=1，见说明），则跳转。
      T     MW16     // 如果没有执行跳转，则继此继续程序扫描。
      A     M 4.0
      R     M 4.0
      JU    NEXT
OVER: AN    M 4.0    // 在跳转到跳转标号 OVER 之后重新进行程序扫描。
      S     M 4.0
NEXT: NOP 0          // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

注意

在这种情况下，禁止使用JO指令。JO指令只用于在发生溢出时检查前一 -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>	跳转目的地的符号名

说明

如果状态位 CC 1 = 0 **并且** CC 0 = 0**，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的** -32768 **或** +32767 **字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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，则跳转到跳转标号 ZERO。
      L    MW2      // 如果没有执行跳转，则继此继续程序扫描。
      INC  1
      T    MW2
      JU   NEXT
ZERO: L    MW4      // 在跳转到跳转标号 ZERO 之后重新进行程序扫描。
      INC  1
      T    MW4
NEXT: NOP  0        // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.13 JN 若非零，则跳转

格式

JN <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

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），使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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。
      AN    M 4.0    // 如果没有执行跳转，则继此继续程序扫描。
      S     M 4.0
      JU    NEXT
NOZE: AN    M 4.1    // 在跳转到跳转标号 NOZE 之后重新进行程序扫描。
      S     M 4.1
NEXT: NOP 0            // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.14 JP 若正，则跳转

格式

JP <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果状态位 CC 1 = 1 并且 CC 0 = 0，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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               // 将输入字 IW8 的内容减去存储字 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，则继此继续程序扫描。
      S  M 4.0
      JUNEXT
POS:  AN     M 4.1     // 在跳转到跳转标号 POS 之后重新进行程序扫描。
      S      M 4.1
NEXT: NOP    0         // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.15 JM 若负，则跳转

格式

JM <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果状态位 CC 1 = 0 并且 CC 0 = 1，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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                // 将输入字 IW8 的内容减去存储字 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    M           // 如果没有执行跳转，则继此继续程序扫描。
      4.0
      S     M
      4.0
      JU    NEXT
NEG:  AN    M4.1        // 在跳转到跳转标号 NEG 之后重新进行程序扫描。
      S     M4.1
NEXT: NOP   0           // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.16 JPZ 若正或零，则跳转

格式

JPZ <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

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），使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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                     // 将输入字 IW8 的内容减去存储字 MW12 的内容。
      JPZ   REG0             // 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，则继此继续程序扫描。
      S     M 4.0
      JU    NEXT
REG0: AN    M 4.1            // 在跳转到跳转标号 REG0 之后重新进行程序扫描。
      S     M 4.1
NEXT: NOP   0                // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.17 JMZ 若负或零，则跳转

格式

JMZ <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

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），使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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                     // 将输入字 IW8 的内容减去存储字 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，则继此继续程序扫描。
      S     M 4.0
      JU    NEXT
RGE0: AN    M 4.1            // 在跳转到跳转标号 RGE0 之后重新进行程序扫描。
      S     M 4.1
NEXT:  NOP   0                // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.18 JUO 若无效数，则跳转

格式

JUO <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

如果状态位 CC 1 = 1 并且 CC 0 = 1，使用该指令，可以中断线性程序扫描，并跳转到一个跳转目的地。在跳转目的地重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（一个、两个或三个字语句）。

在以下情况下，状态位 CC 1 = 1，CC 0 = 1：

- 出现被零除
- 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                // 将存储双字 MD10 的内容除以输入双字 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，则继此继续程序扫描。
      A     M 4.0
      R     M 4.0
      JU    NEXT
ERRO: AN    M 4.0       // 在跳转到跳转标号 ERRO 之后重新进行程序扫描。
      S     M 4.0
NEXT: NOP 0               // 在跳转到跳转标号 NEXT 之后重新进行程序扫描。
```

6.19 LOOP 循环控制

格式

LOOP <跳转标号>

地 址	说 明
<跳转标号>	跳转目的地的符号名

说明

使用循环控制指令（如果累加器 1 低字中的值不为“0”，则累加器 1 低字中的值减“1”，并跳转），可以简化循环控制编程。在累加器 1 低字中提供有循环计数器。指令可以跳转到指定跳转目的地。只要累加器 1 低字中的值不为“0”，就跳转。并在跳转目的地处重新进行线性程序扫描。跳转目的地通过一个跳转标号来指定。向前跳转和向后跳转均可。只能在一个块内执行跳转，即跳转指令和跳转目的地必须位于同一块内。在该块内跳转目的地必须是唯一的。最大跳转距离为程序代码的 -32768 或 +32767 字。可以跳过语句的实际最大数量取决于程序中所使用语句的混合情况（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 中的内容传送到存储双字 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         // 将存储双字 MD20 的当前内容乘以存储字节 MB10 的当前内容。
      T    MD20      // 将相乘结果传送到存储双字 MD20。
      L    MW10      // 将循环计数器的内容装入累加器 1 中。
      LOOP NEXT      // 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      // 循环结束之后重新进行程序扫描。
      L    200
      >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。其结果保存在累加器 1 中。累加器 1 的原有内容被移入累加器 2 中。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 的内容拷入累加器 2 中，将累加器 4 的内容拷入累加器 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 **作为整数（16 位），将累加器1和累加器2中的内容相加**
- -I **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 **作为整数（16 位），将累加器1和累加器2中的内容相乘**
- /I **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**
- + **加上一个整数常数（16 位，32 位）**
- +D **作为双整数（32 位），将累加器1和累加器2中的内容相加**
- -D **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 **作为双整数（32 位），将累加器1和累加器2中的内容相乘**
- /D **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**
- MOD **双整数除法的余数（32位）**

请参见“ 判断整数算术运算指令后状态字的位 ”。

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**。**

下表所示为使用了整数（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
上溢（加法） 16 **位：结果** = -65536 32 **位：结果** = -4 294 967 296	0	0	1	1
上溢（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 作为整数(16位)，将累加器1和累加器2中的内容相加

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 低字中的内容作为 16 位整数编译。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 都设定为指令结果的一个功能。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 的内容拷入累加器 2 中，将累加器 4 的内容拷入累加器 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	// **将输入字** 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 作为整数(16位)，将累加器2的内容减累加器1的内容

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 低字中的内容作为 16 位整数编译。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 都设定为指令结果的一个功能。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 的内容拷入累加器 2 中，将累加器 4 的内容拷入累加器 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 ≤ 差 < 0	0	1	0	-
32767 ≥ 差 > 0	1	0	0	-
65535 ≥ 差 > 32767	0	1	1	1
-65535 ≤ 差 < -32768	1	0	1	1

举例

```
STL                 解  释
L    IW10           // 将输入字 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。将存储字 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 作为整数(16位)，将累加器1和累加器2中的内容相乘

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 低字中的内容作为 16 位整数编译。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 都设定为指令结果的一个功能。

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 的内容拷入累加器 2 中，将累加器 4 的内容拷入累加器 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	// 将输入字 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。将存储字 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 DBW25。