Koyo

Value & Technology

可编程序控制器 D4-454 系列
编程手册

[第一版]

光洋电子(无锡)有限公司

目录

第 1 章 简介

D4-454 指令集可以执行多种不同类型的操作，本手册向您介绍了指令集中常用指令的使用方法。指令介绍中，梯形图中括号中的内容就是各指令的名称，可以通过手持编程器直接将指令写入程序。如果知道指令种类(布尔、比较接点等)，可以利用章节标题直接找到所在页数。

D4-454的编程软件是KPP SOFT，本手册的许多指令在KPP SOFT中不能使用键盘输入，而是隐含在里面。但是，当程序编写和编译完成，可以在语句视图中看到这些指令。

本手册使用的提示标记：

左边的“记事本”图标表明紧随其后的段落将是一条特别的**注意**事项。

左边的“感叹号”图标表示**警告信息**。此信息可以防止伤害、财产损失，甚至人员伤亡。

如果你有有关本手册的情况需要与我们联系，请首先确定手册的版本号！

资料名称：《D4-454 编程手册》

资料编号	编制日期	内容说明
KEW-M3512	2018 年 12 月	原稿第一版试用

第 2 章 逻辑指令

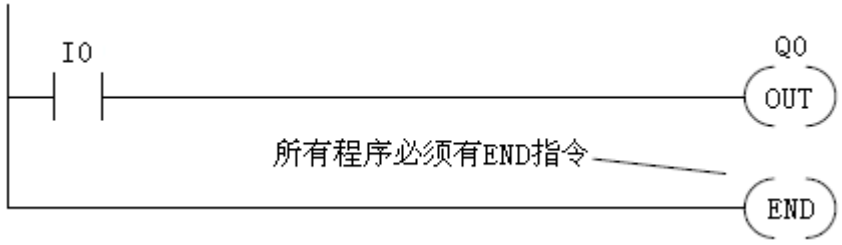
2.1 逻辑指令的使用方法

为什么大多数PLC厂家提供的扫描时间都是1K逻辑程序的扫描时间呢？这是因为大多数程序使用大量的逻辑指令。这些指令通常是非常简单的指令，用于在各种串联和并联回路中添加输入和输出接点。现在各PLC厂家基本都使用梯形图输入进行编程，我们的KPP SOFT软件也类似，因此不需要记住所有的指令。

下面介绍了如何使用逻辑指令来创建简单的梯形图程序。

2.1.1 END 指令

所有D4-454程序需要以END指令结束，以此来通知CPU程序已结束。通常情况下，END指令后的指令不会被执行，子程序、中断程序等除外，相关内容后面会有详细介绍。



2.1.2 命令行

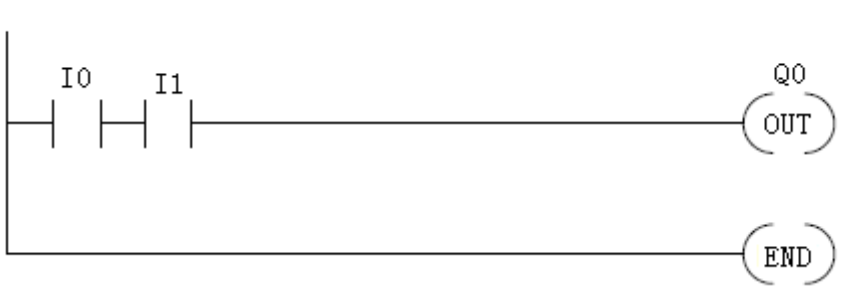
下面的例子是包含一个触点和一个线圈的命令行。触点是 LD 指令，线圈是 OUT 指令。

2.1.3 常闭触点

常闭接点也很常用，常闭接点指令是 LDN，见下面的例子。

2.1.4 触点串联

使用 AND 指令可将两个及以上的触点串联，见下面的例子。

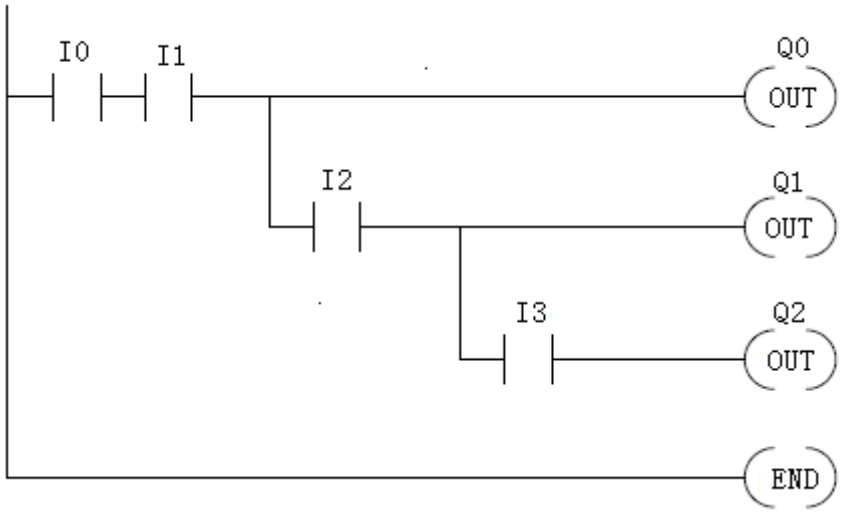


语句视图

```
LD I0
AND I1
OUT Q0
END
```

2.1.5 中间线输出

有时需要使用中间线输出得到其它触点条件的额外输出，见下面的例子，用 AND 指令实现更多条件输出。

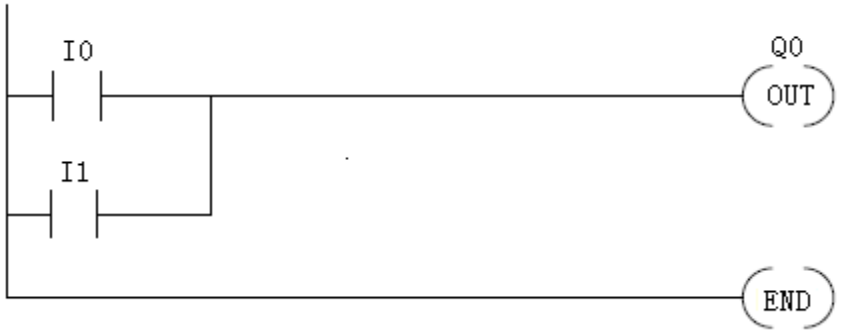


语句视图

```
LD I0
AND I1
OUT Q0
AND I2
OUT Q1
AND I3
OUT Q2
END
```

2.1.6 触点并联

使用 OR 指令可将触点并联，见下面的例子。

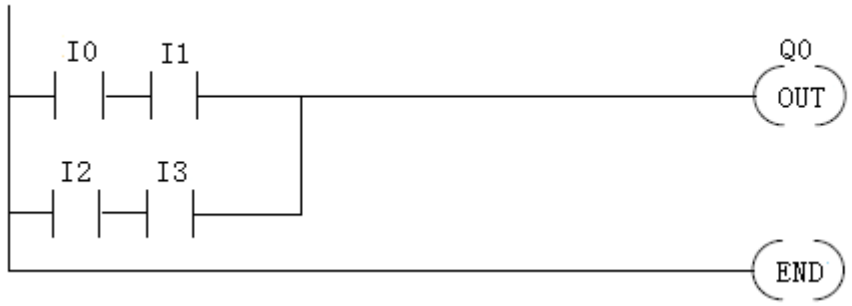


语句视图

```
LD I0
OR I1
OUT Q0
END
```

2.1.7 串联支路并联

使用 ORLD 指令可将串联的支路并联，见下面的例子。

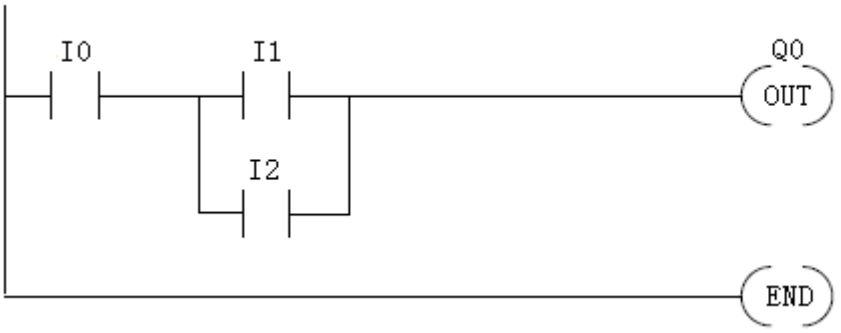


语句视图

```
LD I0
AND I1
LD I2
AND I3
ORLD
OUT Q0
END
```

2.1.8 串联中加入并联支路

使用 ANDLD 指令可在串联回路中加入并联支路，见下面的例子。

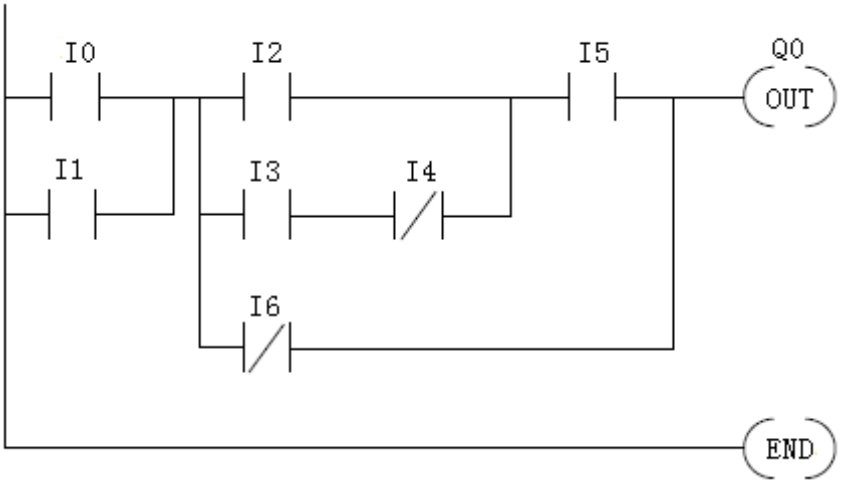


语句视图

```
LD I0
LD I1
OR I2
ANDLD
OUT Q0
END
```

2.1.9 网络组合

可以将各种串联和并联支路组合来解决各种应用问题，下面是一个简单的例子。



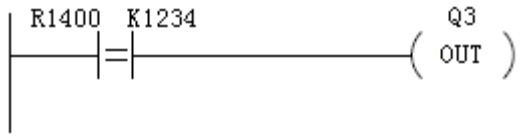
语句视图

```
LD I0
OR I1
LD I2
LD I3
ANDN I4
ORLD
AND I5
ORN I6
ANDLD
OUT Q0
```

2.1.10 比较触点

比较触点是使用一个逻辑触点对两个 4 位 BCD/十六进制数进行比较，比较触点有：比较一致触点、比较不一致触点、大于等于触点和小于等于触点。

下面的例子中，当寄存器 R1400 中的数据等于 BCD 常数 1234 时，Q3 被接通。

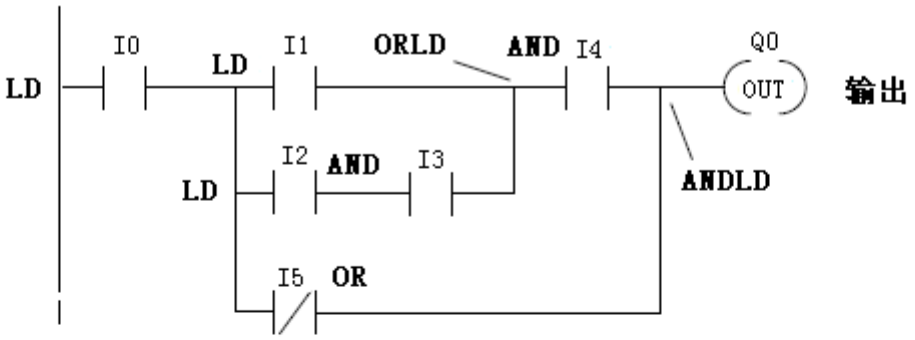


2.1.11 逻辑堆栈

一行程序的指令数是有限制的，这是因为 D4-454 CPU 使用一个 8 级逻辑堆栈来进行逻辑计算，逻辑堆栈是一个逻辑指令的临时存放区域。每输入一个 LD 指令，这个指令就被存放在堆栈的顶端，堆栈中的其它级被压入下一级中。ANDLD 和 ORLD 指令将各级的指令组合起来。由于逻辑堆栈只有 8 级，如果 CPU 检测到程序使用了超出 8 级的逻辑堆栈，就会产生错误。

可能你会说：“我使用 KPP SOFT 编程，这样我就没必要知道逻辑堆栈是如何工作的。”这样是不对的，KPP SOFT 中，虽然可用图形标志创建程序，但是 CPU 对堆栈的限制也是一样的。如果超出这个限制，程序编译的时候将会产生错误。

下面的例子演示了逻辑堆栈是如何工作的。



LD I0

1	LD I0
2	
3	
4	

LD I1

1	LD I1
2	LD I0
3	
4	

LD I2

1	LD I2
2	LD I1
3	LD I0
4	

AND I3

1	I2 AND I3
2	LD I1
3	LD I0
4	

ORLD

1	I1 OR (I2 AND I3)
2	LD I0
3	

AND I4

1	I4 AND [I1 OR (I2 AND I3)]
2	LD I0
3	

OR I5

1	NOT I5 OR I4 AND [I1 OR(I2 AND I3)]
2	LD I0
3	

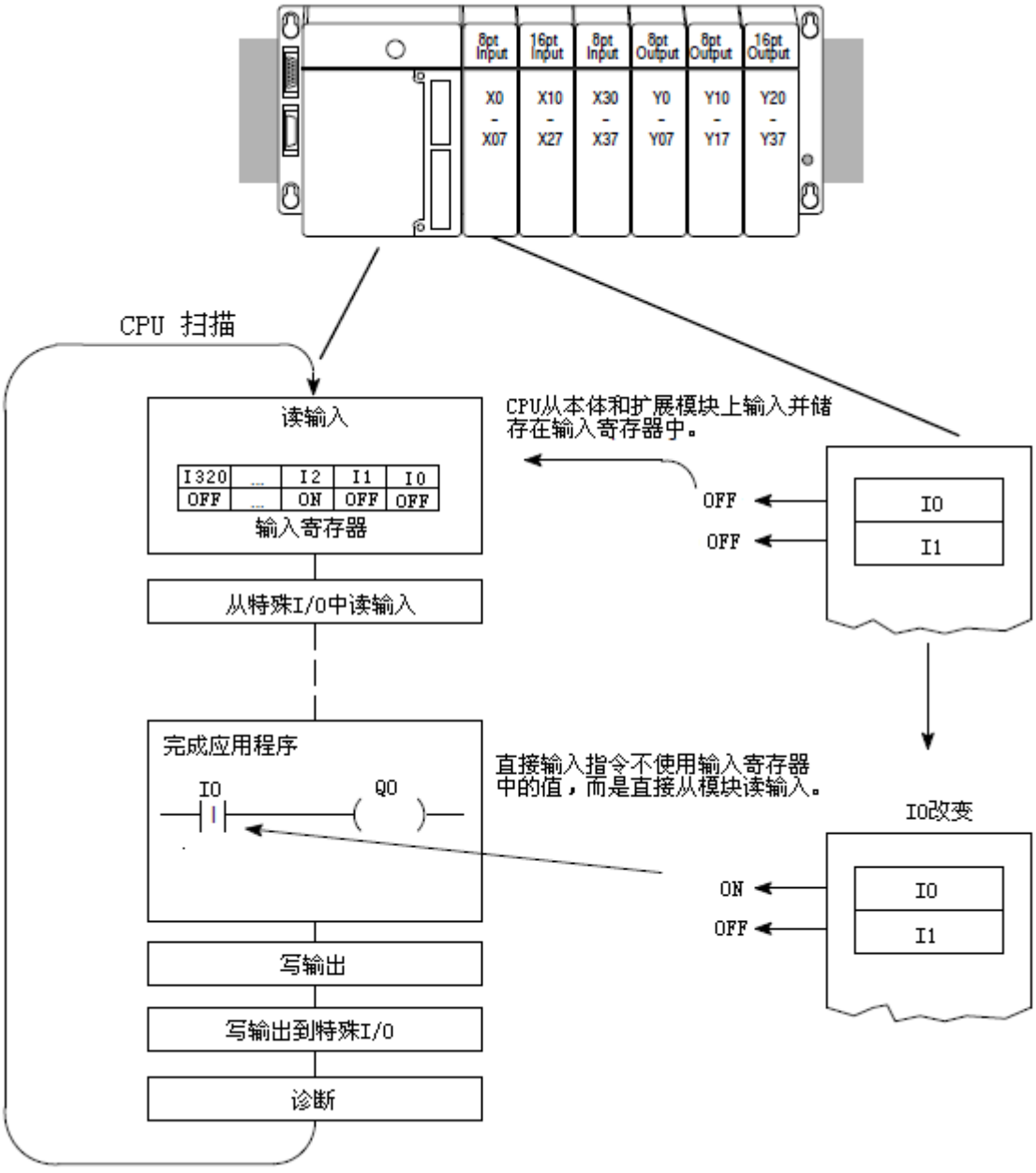
ANDLD

1	I0 AND (NOT I5 OR I4) AND [I1 OR (I2 AND I3)]
2	
3	

2.1.12 直接指令

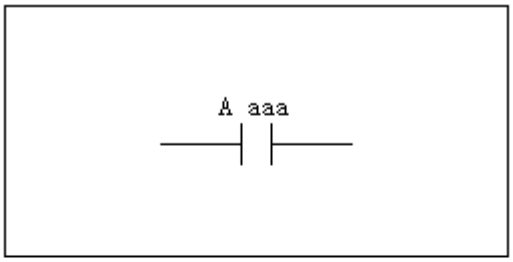
D4-454 CPU 完成一个操作循环通常只需要几毫秒，但是有些应用中，几毫秒都不能等待。D4-454 CPU 提供了特殊的逻辑指令——直接输入和输出指令，在程序执行时允许立即读输入和写输出。直接指令会使程序执行时间加长，因为 CPU 读写输入输出时程序执行被中断。

注意：虽然直接输入指令读取模块的最新状态值，但是只有一个指令是使用这个最新状态值，寄存器的数据并没有被更新，因此，其它常规指令仍然使用寄存器中的状态值，直接输入指令会再次采集到模块的最新状态值。直接输出指令将输出写入模块并且更新寄存器的数据。



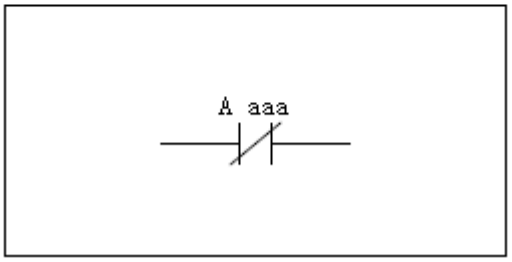
2.2 逻辑运算开始常开触点（LD）

LD 指令是一个常开触点，用于开始一行程序或是增加一个支路。触点的状态同映像寄存器或存储地址中的相应位状态保持一致。



2.3 逻辑运算开始常闭触点（LDN）

LDN 指令是一个常闭触点，用于开始一行程序或是增加一个支路。触点的状态同映像寄存器或存储地址中的相应位状态相反。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
特殊继电器	SP	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

在下面的 LD 指令例子中，当 I1 为 ON 时，输出 Q2 被接通。

语句视图

```
LD I1
OUT Q2
```

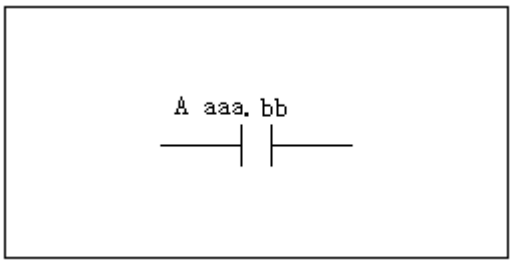
在下面的 LDN 指令例子中，当 I1 为 OFF 时，输出 Q2 被接通。

语句视图

```
LDN I1
OUT Q2
```

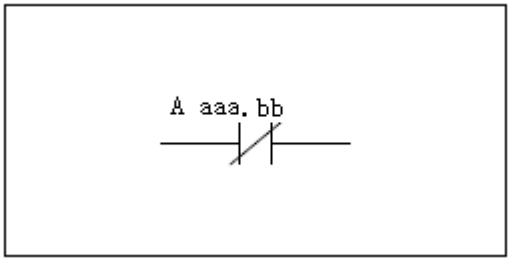
2.4 位逻辑运算开始常开触点（BLD）

BLD 指令是一个常开触点，用于开始一行程序或是增加一个支路。触点的状态同存储地址中的相应位状态保持一致。



2.5 位逻辑运算开始常闭触点（BLDN）

BLDN 指令是一个常闭触点，用于开始一行程序或是增加一个支路。触点的状态同存储地址中的相应位状态相反。



操作数类型	D4-454 范围	
A	aaa	bb
寄存器 B	所有（附录 1）	BCD，0-15
指针 PB	所有（附录 1）	BCD，0-15

在下面的 BLD 指令例子中，当寄存器 R1400 的位 12 为“1”时，输出 Q2 被接通。

语句视图

```
BLD B1400.12
OUT Q2
```

在下面的 BLDN 指令例子中，当寄存器 R1400 的位 12 为“0”时，输出 Q2 被接通。

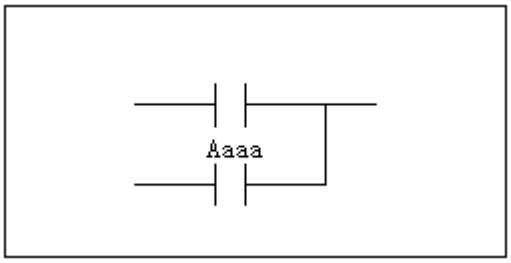
语句视图

```
BLDN B1400.12
OUT Q2
```

注意：使用 KPP SOFT 软件编程时，BLD 和 BLDN 的图标分别是 和 。

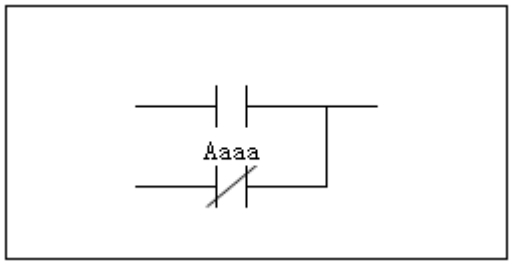
2.6 逻辑或运算常开触点（OR）

OR 指令将一个常开触点与命令行中另一个触点并联，触点的状态同指定映像寄存器点或功能存储器的状态相同。



2.7 逻辑或运算常闭触点（ORN）

ORN 将一个常闭触点与命令行中另一个触点并联，触点的状态同指定映像寄存器点或功能存储器的状态相反。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
特殊继电器	SP	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的 OR 指令例子中，当输入 I1 或 I2 为 ON 时，输出 Q5 被接通。

语句视图

```
LD  I1
OR  I2
OUT Q5
```

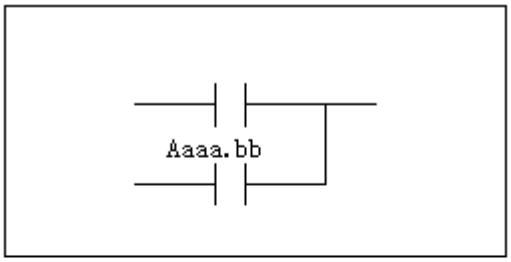
下面的 ORN 指令例子中，当输入 I1 为 ON 或 I2 为 OFF 时，输出 Q5 被接通。

语句视图

```
LD  I1
ORN I2
OUT Q5
```

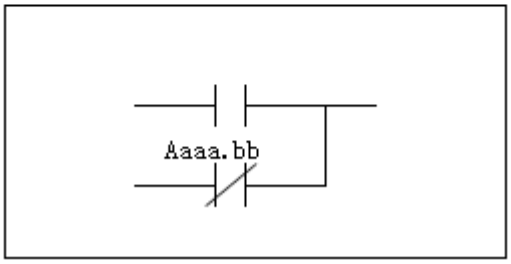
2.8 位逻辑或运算常开触点（BOR）

BOR 指令将一个常开触点与命令行中另一个触点并联，常开触点的状态同存储器的指定位状态相同。



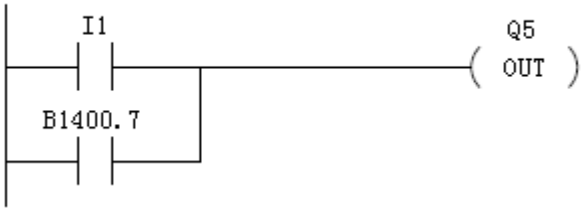
2.9 位逻辑或运算常闭触点（BORN）

BORN 指令将一个常闭触点与命令行中另一个触点并联，常闭触点的状态同存储器的指定位状态相反。



操作数类型		D4-454 范围	
	A	aaa	bb
寄存器	B	所有（附录 1）	BCD，0-15
指针	PB	所有（附录 1）	BCD，0-15

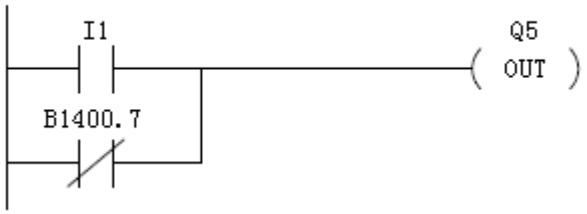
下面的 BOR 指令例子中，当输入 I1 或 R1400 的位 7 为 ON 时，输出 Q5 被接通。



语句视图

```
LD  I1
BOR B1400.7
OUT Q5
```

下面的 BORN 指令例子中，当输入 I1 为 ON 或 R1400 的位 7 为 OFF 时，输出 Q5 被接通。

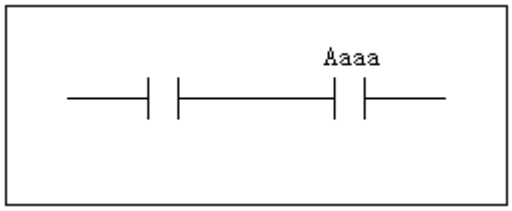


语句视图

```
LD  I1
BORN B1400.7
OUT Q5
```

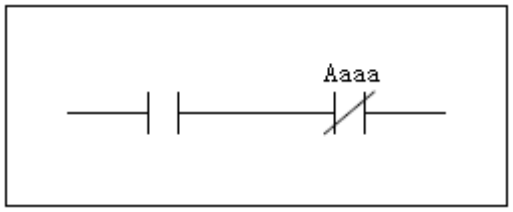
2.10 逻辑与运算常开触点（AND）

AND 指令将一个常开触点同命令行另一个触点串联，触点的状态同指定映像寄存器或存储器的状态相同。



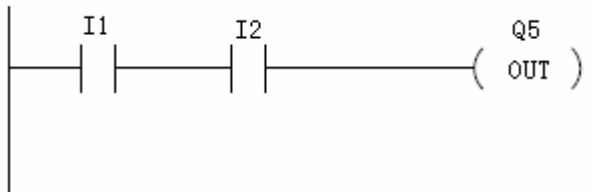
2.11 逻辑与运算常闭触点（ANDN）

ANDN 指令将一个常闭触点同命令行另一个触点串联，触点的状态同指定映像寄存器或存储器的状态相反。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
特殊继电器	SP	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

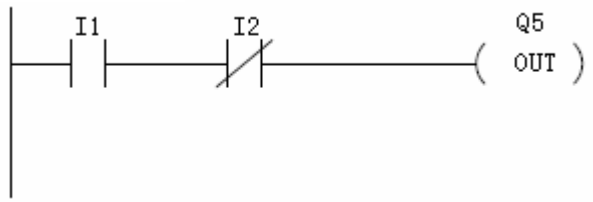
下面的 AND 指令例子中，当输入 I1 和 I2 均为 ON 时，输出 Q5 被接通。



语句视图

```
LD  I1
AND I2
OUT Q5
```

下面的 ANDN 指令例子中，当输入 I1 为 ON 并且 I2 为 OFF 时，输出 Q5 被接通。

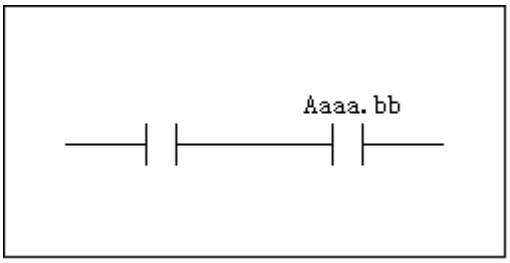


语句视图

```
LD  I1
ANDN I2
OUT Q5
```

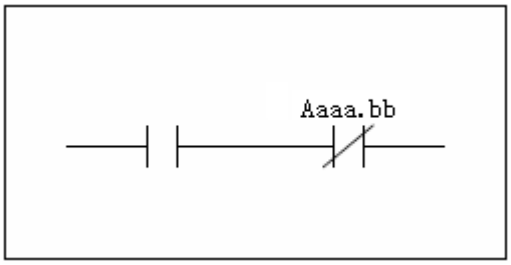
2.12 位逻辑与运算常开触点（BAND）

BAND 指令将一个常开触点同命令行另一个触点串联，触点的状态同功能存储器的指定位状态相同。



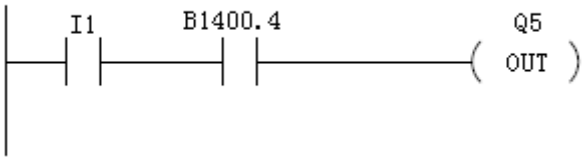
2.13 位逻辑与运算常闭触点（BANDN）

BANDN 指令将一个常闭触点同命令行另一个触点串联，触点的状态同功能存储器的指定位状态相反。



操作数类型		D4-454 范围	
	A	aaa	bb
寄存器	B	所有（附录 1）	BCD，0-15
指针	PB	所有（附录 1）	BCD，0-15

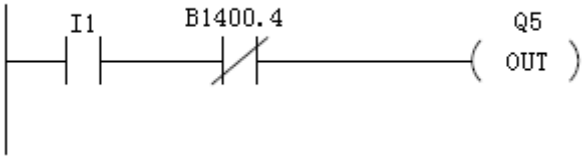
下面的 BAND 指令例子中，当输入 I1 和 R1400 的位 4 均为 ON 时，输出 Q5 被接通。



语句视图

```
LD I1
BAND B1400.4
OUT Q5
```

下面的 BANDN 指令例子中，当输入 I1 为 ON 并且 R1400 的位 4 为 OFF 时，输出 Q5 被接通。

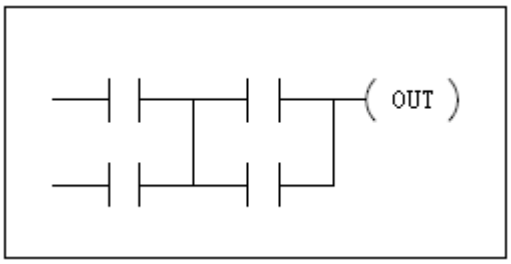


语句视图

```
LD I1
BANDN B1400.4
OUT Q5
```

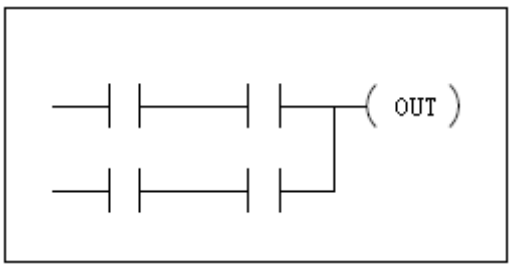
2.14 逻辑组串联指令（ANDLD）

ANDLD 指令将两个逻辑组串联,两个逻辑组都必须以逻辑运算开始指令开头。

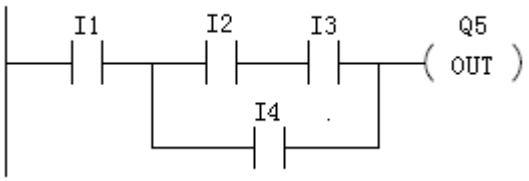


2.15 逻辑组并联指令（ORLD）

ORLD 指令将两个逻辑组并联，两个逻辑组都必须以逻辑运算开始指令开头。



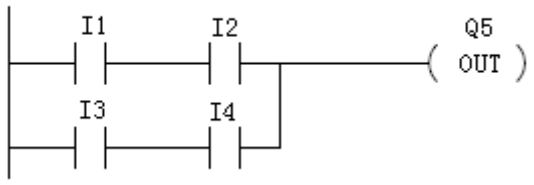
下面的 ANDLD 指令例子中，I2、I3 和 I4 组成的逻辑组同 I1 串联。



语句视图

```
LD I1
LD I2
AND I3
OR I4
ANDLD
OUT Q5
```

下面的 ORLD 指令例子中，I1、I2 组成的逻辑组同 I3、I4 组成的逻辑组并联。

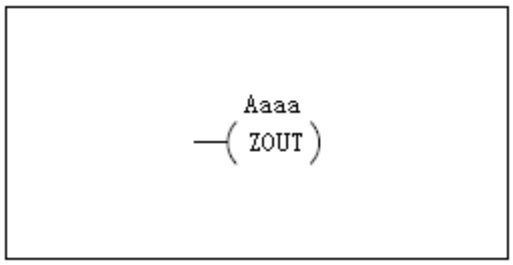


语句视图

```
LD I1
AND I2
LD I3
AND I4
ORLD
OUT Q5
```

2.16 输出指令（后面优先）（ZOUT）

输出指令将程序结果输出到功能存储器，多个输出指令不可使用相同的功能存储器地址，因为只有最后一个指令控制物理输出点。



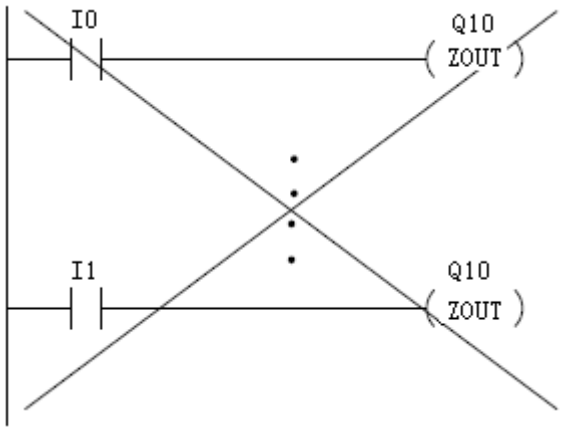
操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的输出指令例子中，当 I1 为 ON 时，输出 Q2 和 Q5 被接通。

语句视图

```
LD  I1
ZOUT  Q2
ZOUT  Q5
```

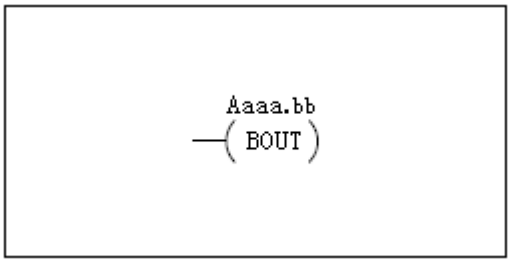
下面的输出指令例子中，有两个输出指令使用了相同的地址（Q10），Q10 最终是由下面一条程序控制，I1 将覆盖 I0 的控制结果，因此编程时要避免多个输出指令使用同一个地址。



注意： 1. OUT 指令多重使用时，OR 动作。ZOUT 指令多重使用时，后面的优先动作。
2. 如果同一线圈没有多重使用，OUT 指令、ZOUT 指令动作相同。
3. 在通常的梯形图程序中，同一线圈不多重使用，所以没有必要区别 OUT 和 ZOUT。

2.17 位输出指令（BOUT）

位输出指令将程序结果输出到功能存储器的指定位，多个输出指令不可使用相同的功能存储器位地址，因为只有最后一个指令控制物理输出点。



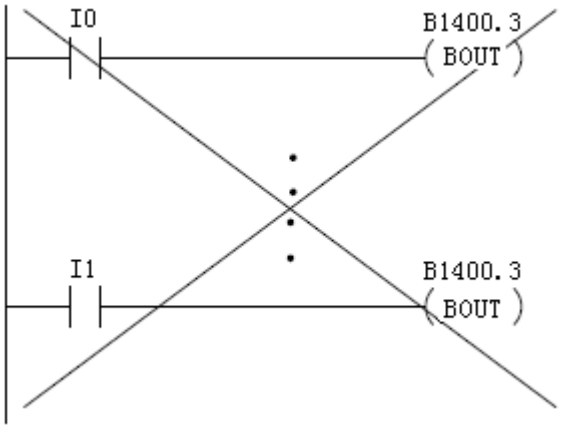
操作数类型		D4-454 范围	
	A	aaa	bb
寄存器	B	所有（附录 1）	BCD，0-15
指针	PB	所有（附录 1）	BCD，0-15

下面的位输出指令例子中，当 I1 为 ON 时，R1400 的位 3 和 R1401 的位 6 被置为 ON。

语句视图

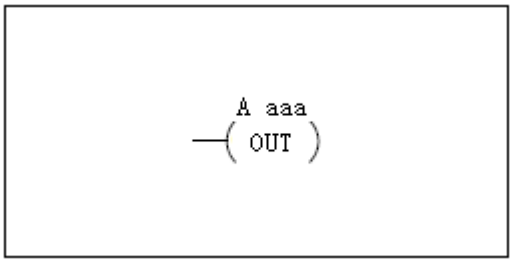
```
LD  I1
BOUT  B1400.3
BOUT  B1401.6
```

下面的输出指令例子中，有两个输出指令使用了相同的地址（R1400 的位 3），R1400 的位 3 最终是由下面一条程序控制，I1 将取代 I0 的控制结果，因此编程时要避免多个输出指令使用同一个地址。



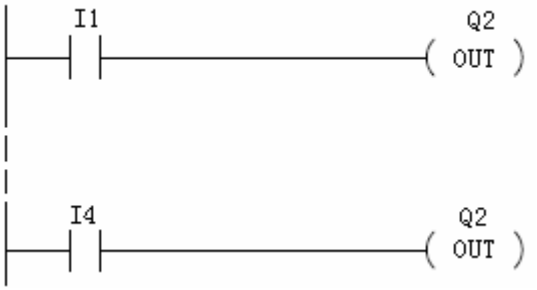
2.18 动作输出指令（OUT）

OUT 指令是或逻辑动作输出指令，用于多段程序控制单个输出的程序，任何一段程序为 ON 时，输出也为 ON。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的例子中，当 I1 或 I4 为 ON 时，Q2 被接通。



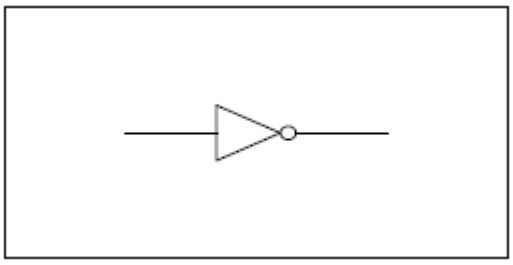
语句视图

```
LD  I1
OUT Q2
LD  I4
OUT Q2
```

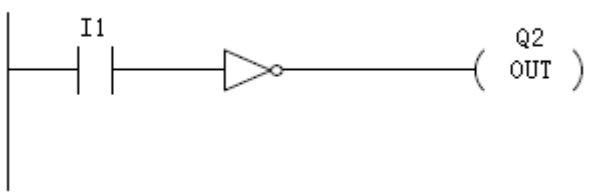
注意： 1. OUT 指令多重使用时，或逻辑动作。ZOUT 指令多重使用时，后面的优先动作。
2. 如果同一线圈没有多重使用，OUT 指令、ZOUT 指令动作相同。
3. 在通常的梯形图程序中，同一线圈不多重使用，所以没有必要区别 OUT 和 ZOUT。

2.19 非指令（NOT）

非指令将当前回路的状态取反。



下面的例子中，当 I1 为 OFF 时，Q2 被接通。这是因为 NOT 指令改变了回路的状态。

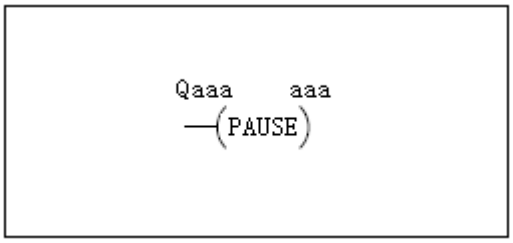


语句视图

```
LD  I1
NOT
OUT Q2
```

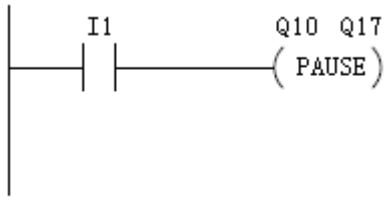
2.20 暂停指令（PAUSE）

暂停指令用于将一定范围的输出禁用，禁用的同时，程序可继续执行并更新映像寄存器，但是被禁用的输出在输出模块被置为 OFF。



操作数类型		D4-454 范围
	A	aaa
输出	Q	0-1777

下面的程序例子中，当 I1 为 ON，Q10-Q17 被置为 OFF，程序的执行不起作用。

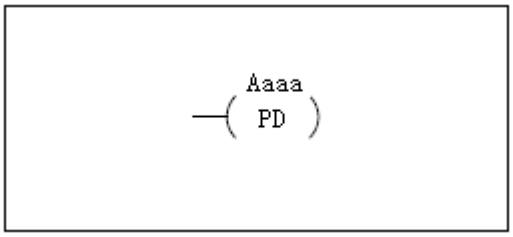


语句视图

```
LD I1
PAUSE Q10 Q17
```

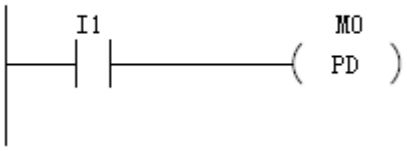
2.21 一次扫描输出指令（PD）

当开关量输入产生一个 OFF→ON 的脉冲时，在脉冲的上升沿输出将接通一个扫描周期，此后，其仍然保持 OFF 直到下一个 OFF→ON 脉冲的产生。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777

下面的例子中，每次 I1 上产生一个 OFF→ON 的变化，M0 就接通一个扫描周期。



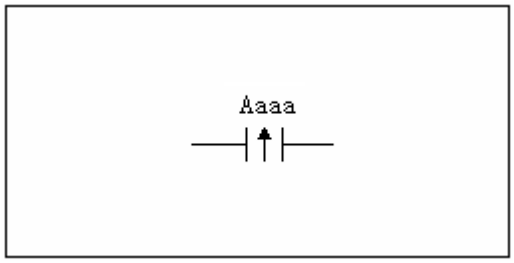
语句视图

```
LD I1
PD M0
```

注意，如果在 PD 指令前加入一个 NOT 指令，可在 ON→OFF 变化时产生一个扫描周期的脉冲。

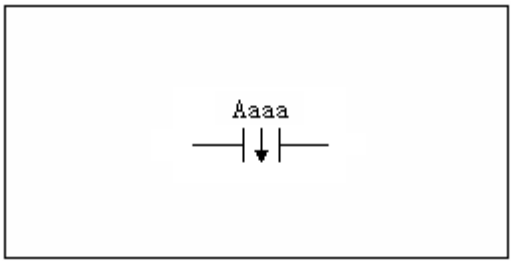
2.22 逻辑运算开始上升沿接点（LDPD）

LDPD 指令可直接连接在母线上，也可以连接在触点组上使用，是一个常开触点。当指定的功能存储器位发生 OFF→ON 的变化时，PD 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 OFF→ON 变化的发生。如果映像寄存器处于停电保持区，由编程模式→运行模式也会接通。



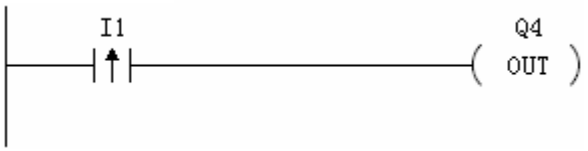
2.23 逻辑运算开始下降沿接点（LDND）

LDND 指令可直接连接在母线上，也可以连接在触点组上使用，是一个常开触点。当指定的功能存储器位发生 ON→OFF 的变化时，ND 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 ON→OFF 变化的发生。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

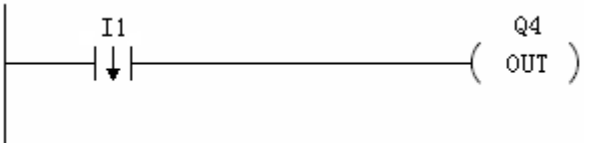
下面的程序例子中，每次 I1 上发生 OFF→ON 的变化时，Q4 将接通一个周期。



语句视图

```
LDPD I1
OUT Q4
```

下面的程序例子中，每次 I1 上发生 ON→OFF 的变化时，Q4 将接通一个周期。

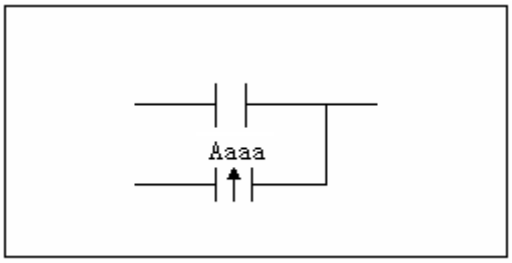


语句视图

```
LDND I1
OUT Q4
```

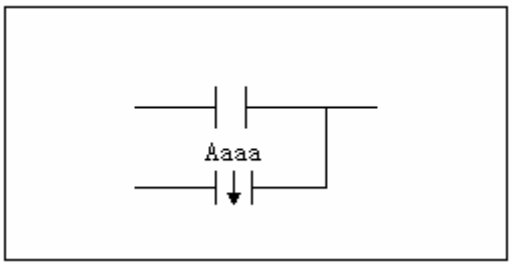
2.24 逻辑或运算上升沿接点（ORPD）

ORPD 指令将一个 PD 触点同一个触点并联。当指定的功能存储器位发生 OFF→ON 的变化时，PD 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 OFF→ON 变化的发生。



2.25 逻辑或运算下降沿接点（ORND）

ORND 指令将一个 ND 触点同一个触点并联。当指定的功能存储器位发生 ON→OFF 的变化时，ND 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 ON→OFF 变化的发生。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的程序例子中，当 I1 为 ON，Q5 被接通；当 I2 上发生 OFF→ON 的变化时，Q5 被接通一个周期。

语句视图

```
LD I1
ORPD I2
OUT Q5
```

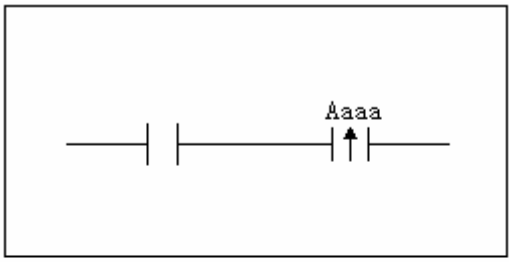
下面的程序例子中，每次 I1 为 ON，Q5 被接通；当 I2 上发生 ON→OFF 的变化时，Q5 被接通一个周期。

语句视图

```
LD I1
ORND I2
OUT Q5
```

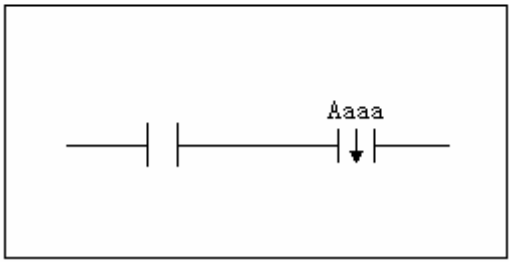
2.26 逻辑与运算上升沿接点（ANDPD）

ANDPD 指令将一个 PD 触点同一个触点串联。当指定的功能存储器位发生 OFF→ON 的变化时，PD 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 OFF→ON 变化的发生。



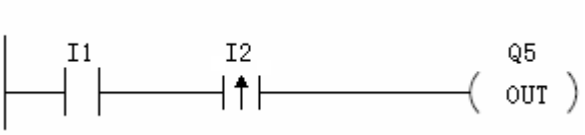
2.27 逻辑与运算开始下降沿接点（ANDND）

ANDND 指令将一个 ND 触点同一个触点串联。当指定的功能存储器位发生 ON→OFF 的变化时，ND 触点将闭合一个扫描周期。此后，此触点将保持断开状态直到下一个 ON→OFF 变化的发生。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

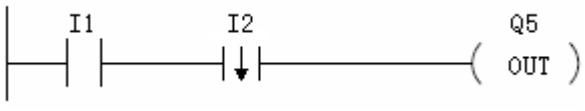
下面的程序例子中，当 I1 为 ON 并且 I2 上产生 OFF→ON 的变化时，Q5 被接通一个周期。



语句视图

```
LD  I1
ANDPD  I2
OUT  Q5
```

下面的程序例子中，当 I1 为 ON 并且 I2 上产生 ON→OFF 的变化时，Q5 被接通一个周期。



语句视图

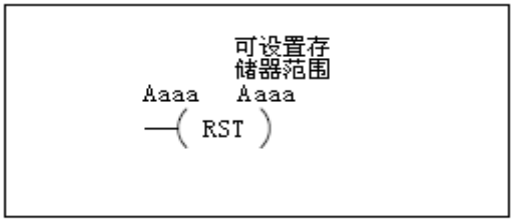
```
LD  I1
ANDND  I2
OUT  Q5
```

2.28 线圈置位指令（SET）

SET 指令将指定的一个或连续一段映像寄存器点/存储器置位，这些点一旦被置位就一直处于 ON 状态，直到 RST 指令将其复位。控制 SET 指令的控制条件不需要一直保持 ON。

2.29 线圈复位指令（RST）

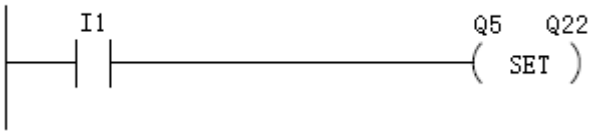
RST 指令将指定的一个或连续一段映像寄存器点/存储器复位，这些点一旦被复位就一直处于 OFF 状态，直到 SET 指令将其置位。控制 RST 指令的控制条件不需要一直保持 ON。



操作数类型		D4-454 范围
	A	aaa
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

注意：不可将分配给输入模块的输入点（I）置位。

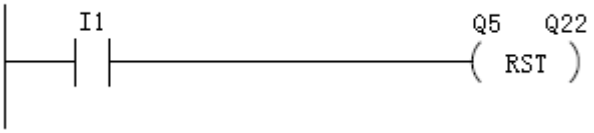
下面的程序例子中，当 I1 变为 ON 时，输出 Q5-Q22 被置位。



语句视图

```
LD I1
SET Q5 Q22
```

下面的程序例子中，当 I1 变为 ON 时，输出 Q5-Q22 被复位。

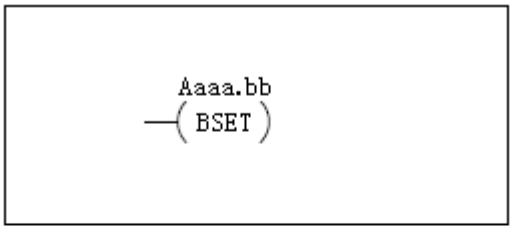


语句视图

```
LD I1
RST Q5 Q22
```

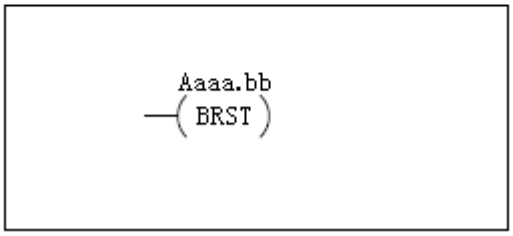
2.30 BIT 置位指令（BSET）

BSET 指令将指定寄存器的指定位置位，指定位一旦被置位就一直处于 ON 状态，直到 BRST 指令将其复位。控制 BSET 指令的控制条件不需要一直保持 ON。



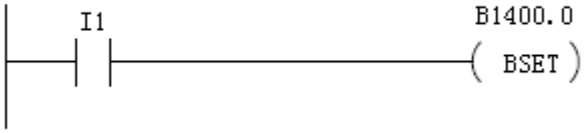
2.31 BIT 复位指令（BRST）

BRST 指令将指定寄存器的指定位复位，指定位一旦被复位就一直处于 OFF 状态，直到 BSET 指令将其置位。控制 BRST 指令的控制条件不需要一直保持 ON。



操作数类型		D4-454 范围	
	A	aaa	bb
寄存器	B	所有（附录 1）	BCD，0-15
指针	PB	所有（附录 1）	BCD，0-15

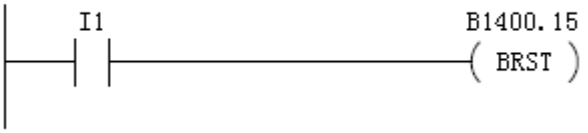
下面的程序例子中，当 I1 变为 ON 时，R1400 的位 0 被置位。



语句视图

```
LD  I1
BSET  B1400.0
```

下面的程序例子中，当 I1 变为 ON 时，R1400 的位 15 被复位。



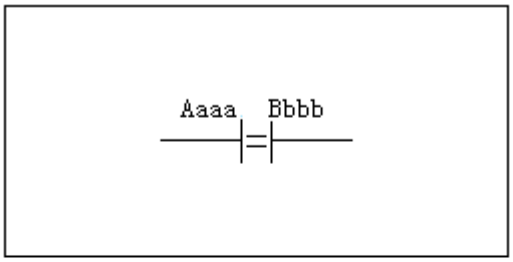
语句视图

```
LD  I1
BRST  B1400.15
```

第 3 章 比较指令

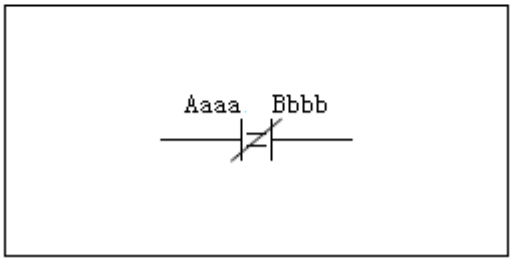
3.1 逻辑运算开始比较一致触点（LDEQ）

LDEQ 指令开始一个回路或添加一个支路，是常开比较触点，当 A aaa=B bbb 时，触点接通。



3.2 逻辑运算开始比较不一致触点（LDNEQ）

LDNEQ 指令开始一个回路或添加一个支路，是常闭比较触点，当 Aaaa≠Bbbb 时，触点接通。



操作数类型		D4-454 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	0-FFFF

下面的例子程序中，当 R1400 中的数据等于 4933 时，Q3 被接通。

语句视图

```
LDEQ R1400 K4933
OUT Q3
```

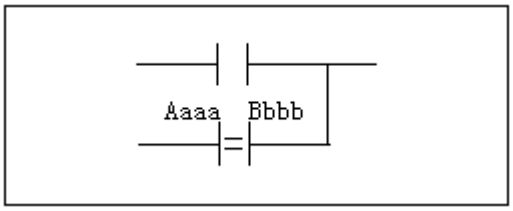
下面的例子程序中，当 R1400 中的数据不等于 5060 时，Q3 被接通。

语句视图

```
LDNEQ R1400 K5060
OUT Q3
```

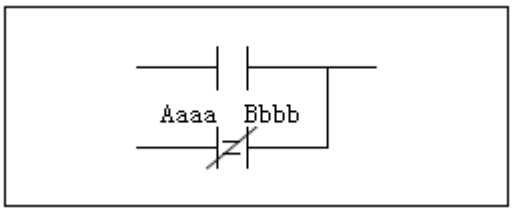
3.3 逻辑或运算比较一致触点（OREQ）

OREQ 指令将一个 EQ 触点并联到另一个触点，当 Aaaa=Bbbb 时，EQ 触点接通。



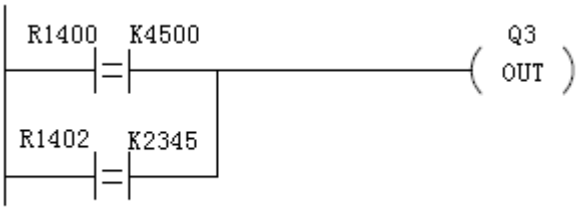
3.4 逻辑或运算比较不一致触点（ORNEQ）

ORNEQ 指令将一个 NEQ 触点同一个触点并联，当 Aaaa≠Bbbb 时，NEQ 触点接通。



操作数类型		D4-454 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	0-FFFF

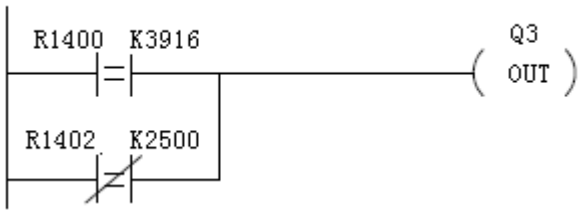
下面的例子中，当 R1400=4500 或 R1402=2345 时，Q3 被接通。



语句视图

```
LDEQ R1400 K4500
OREQ R1402 K2345
OUT Q3
```

下面的例子中，当 R1400=3916 或 R1402≠2500 时，Q3 被接通。

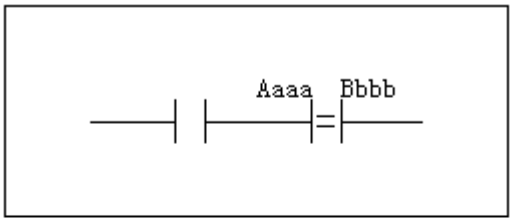


语句视图

```
LDEQ R1400 K3916
ORNEQ R1402 K2500
OUT Q3
```

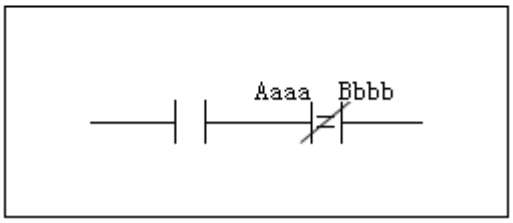
3.5 逻辑与运算比较一致触点（ANDEQ）

ANDEQ 指令将一个 EQ 触点同一个触点串联，当 Aaaa=Bbbb 时，EQ 触点接通。



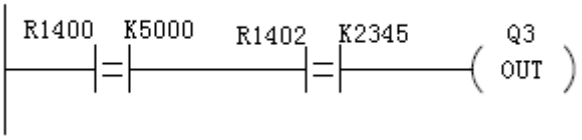
3.6 逻辑与运算比较不一致接点（ANDNEQ）

ANDNEQ 指令将一个 NEQ 触点同一个触点串联，当 Aaaa≠Bbbb 时，NEQ 触点接通。



操作数类型		D4-454 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	0-FFFF

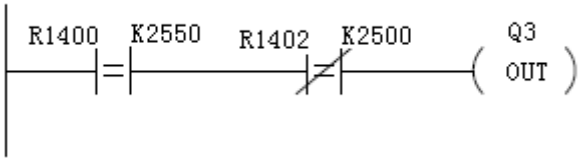
下面的例子中，当 R1400=5000 并且 R1402=2345 时，Q3 被接通。



语句视图

```
LDEQ R1400 K5000
ANDEQ R1402 K2345
OUT Q3
```

下面的例子中，当 R1400=2550 并且 R1402≠2500 时，Q3 被接通。

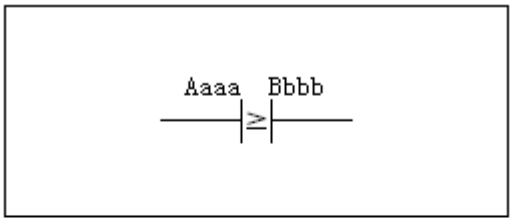


语句视图

```
LDEQ R1400 K2550
ANDNEQ R1402 K2500
OUT Q3
```

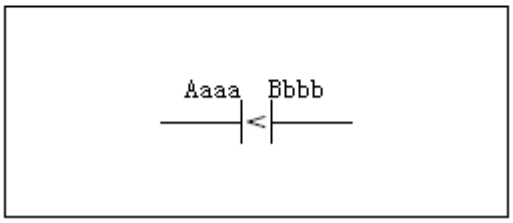
3.7 比较大于等于逻辑运算开始触点（LDGE）

LDGE 指令用一个 GE 触点开始一个新回路或一个新支路，当 Aaaa≥Bbbb 时，触点接通。



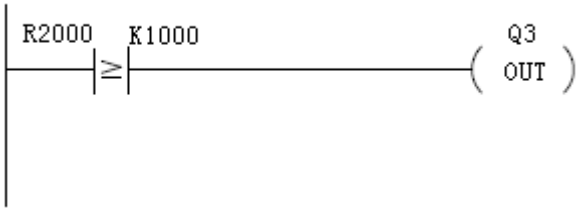
3.8 比较小于逻辑运算开始触点（LDNGE）

LDNGE 指令用一个 NGE 触点开始一个新回路或一个新支路，当 Aaaa＜Bbbb 时，触点接通。



操作数类型		D4-454 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	0-FFFF

下面的例子中，当 R2000≥1000 时，Q3 被接通。



语句视图

```
LDGE R2000 K1000
OUT Q3
```

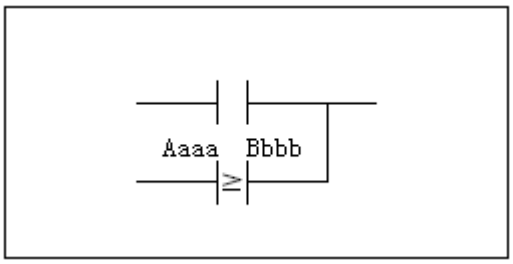
下面的例子中，当 R2000＜4050 时，Q3 被接通。

语句视图

```
LDNGE R2000 K4050
OUT Q3
```

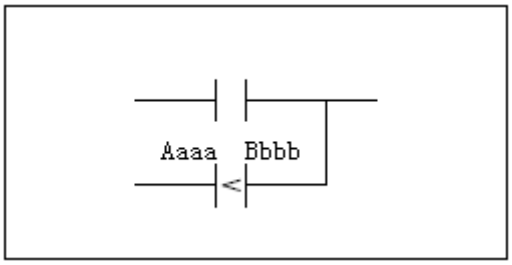
3.9 比较大于等于逻辑或运算触点（ORGE）

ORGE 指令将一个 GE 触点同另一个触点并联，当 Aaaa≥Bbbb 时，触点接通。



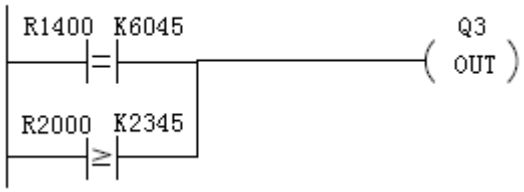
3.10 比较小于逻辑或运算触点（ORNGE）

ORNGE 指令将一个 NGE 触点同另一个触点并联，当 Aaaa＜Bbbb 时，触点接通。



操作数类型		D4-454 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	0-FFFF

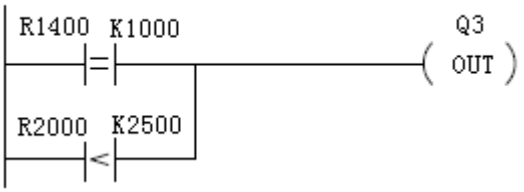
下面的例子中，当 R1400=6045 或 R2000≥2345 时，Q3 被接通。



语句视图

```
LDEQ R1400 K6045
ORGE R2000 K2345
OUT Q3
```

下面的例子中，当 R1400=1000 或 R2000＜2500 时，Q3 被接通。

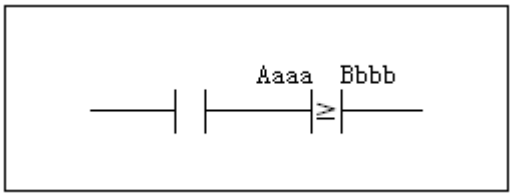


语句视图

```
LDEQ R1400 K1000
ORNGE R2000 K2500
OUT Q3
```

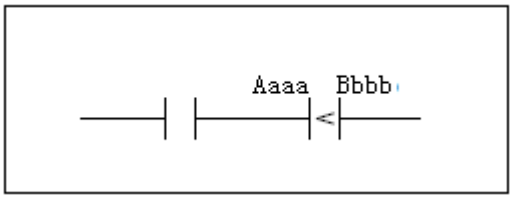
3.11 比较大于等于逻辑与运算接点（ANDGE）

ANDGE 指令将一个 GE 触点同另一个触点串联，当 Aaaa≥Bbbb 时，触点接通。



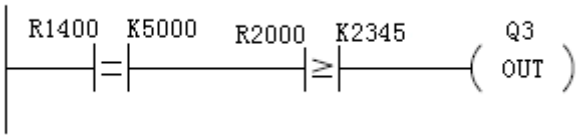
3.12 比较小于逻辑与运算接点（ANDNGE）

ANDNGE 指令将一个 NGE 触点同另一个触点串联，当 Aaaa＜Bbbb 时，触点接通。



操作数类型		D4-454 范围	
	A/B	aaa	bbb
寄存器	R	所有(附录 1)	所有(附录 1)
指针	P	所有(附录 1)	所有(附录 1)
常数	K	-	0-FFFF

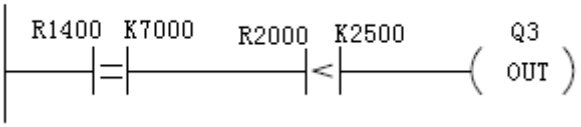
下面的例子中，当 R1400=5000 并且 R2000≥2345 时，Q3 被接通。



语句视图

```
LDEQ R1400 K5000
ANDGE R2000 K2345
OUT Q3
```

下面的例子中，当 R1400=7000 并且 R2000＜2500 时，Q3 被接通。



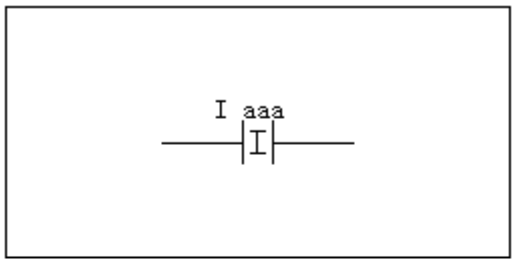
语句视图

```
LDEQ R1400 K7000
ANDNGE R2000 K2500
OUT Q3
```

第 4 章 直接指令

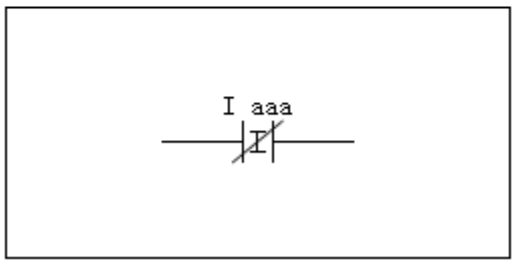
4.1 逻辑运算开始常开直接触点（LDDI）

LDDI 指令开始一个新回路或一个新支路，指令执行时，触点的状态同指定的模块输入点状态相同，对应映像寄存器状态未更新。



4.2 逻辑运算开始常闭直接触点（LDNDI）

LDNDI 指令开始一个新回路或一个新支路，指令执行时，触点的状态同指定的模块输入点状态相反，对应映像寄存器状态未更新。



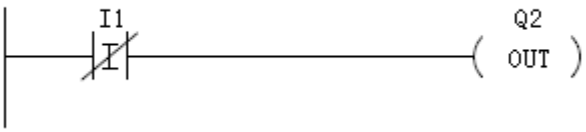
操作数类型	D4-454 范围
	aaa
输入 I	0-1777

下面的例子中，当 I1 为 ON 时，Q2 被接通。

语句视图

```
LDDI I1
OUT Q2
```

下面的例子中，当 I1 为 OFF 时，Q2 被接通。

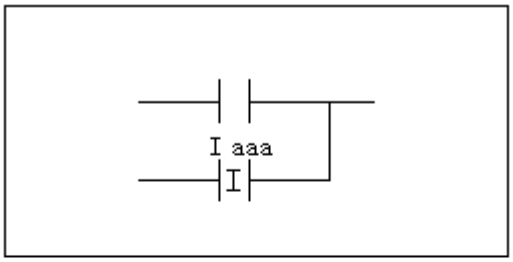


语句视图

```
LDNDI I1
OUT Q2
```

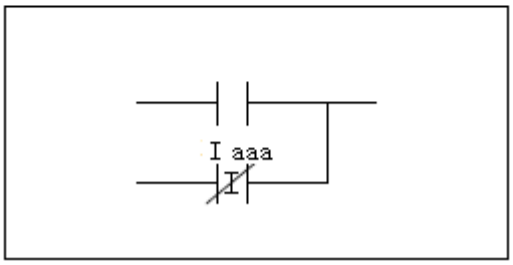
4.3 逻辑或运算常开直接触点（ORDI）

ORDI 指令将两个触点并联，指令执行时，触点状态同指定的模块输入点状态相同,对应映像寄存器状态未更新。



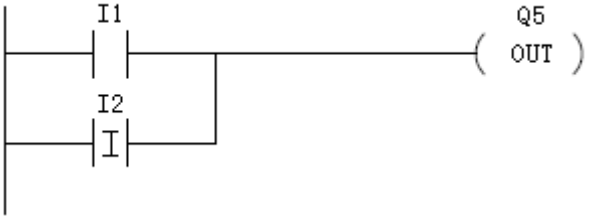
4.4 逻辑或运算常闭直接触点（ORNDI）

ORNDI 指令将两个触点并联，指令执行时，触点状态同指定的模块输入点状态相反,对应映像寄存器状态未更新。



操作数类型	D4-454 范围
	aaa
输入 I	0-1777

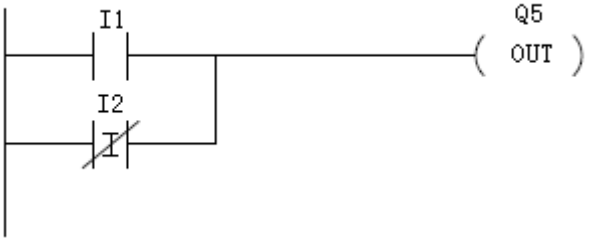
下面的例子中，当 I1 或 I2 为 ON 时，Q5 被接通。



语句视图

```
LD I1
ORDI I2
OUT Q5
```

下面的例子中，当 I1 为 ON 或是 I2 为 OFF 时，Q5 被接通。

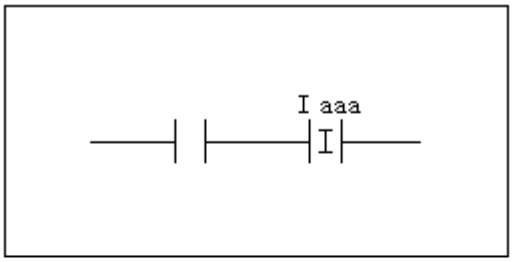


语句视图

```
LD I1
ORNDI I2
OUT Q5
```

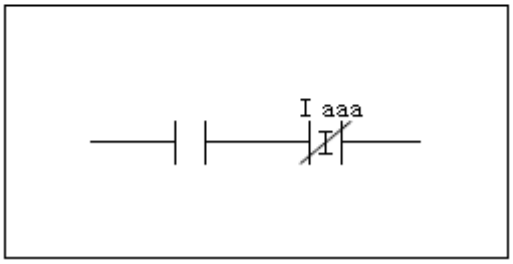
4.5 逻辑与运算常开直接触点（ANDDI）

ANDDI 指令将两个触点串联，接点状态同指定的模块输入点状态相同，对应映像寄存器状态未刷新。



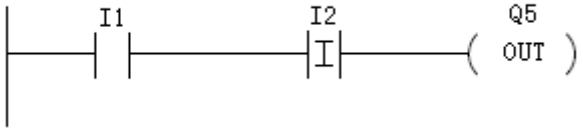
4.6 逻辑与运算常闭直接触点（ANDNDI）

ANDNDI 指令将两个触点串联，触点状态同指定的模块输入点状态相反，对应映像寄存器状态未刷新。



操作数类型	D4-454 范围
	aaa
输入 I	0-1777

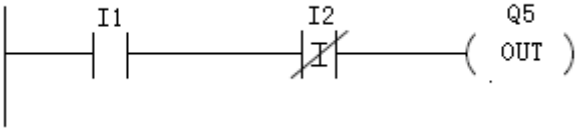
下面的例子中，当 I1 和 I2 均为 ON 时，Q5 被接通。



语句视图

```
LD I1
ANDDI I2
OUT Q5
```

下面的例子中，当 I1 为 ON 并且 I2 为 OFF 时，Q5 被接通。

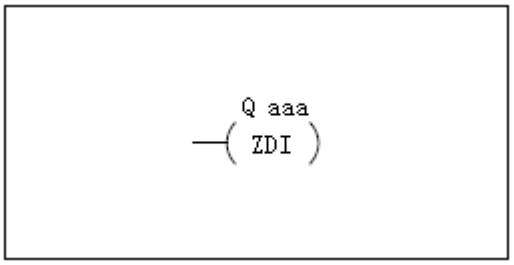


语句视图

```
LD I1
ANDNDI I2
OUT Q5
```

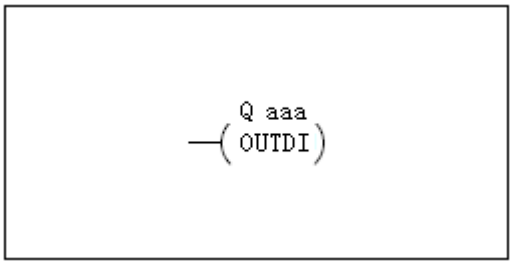
4.7 直接输出指令（ZDI）

直接输出指令表现回路状态，并将回路状态输出到指定的模块输出点，指令执行时，对应映像寄存器状态也被更新。当有多个直接输出指令对同一个模块输出点进行输出时，那这个输出点的状态在一个扫描周期中可能变化多次，见 OUTDI 指令。



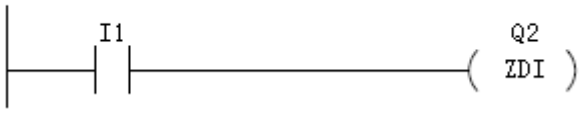
4.8 OR 直接输出指令（OUTDI）

OUTDI 指令用于多个输出回路控制同一个输出点。由于多个 OUTDI 输出到同一个输出点，因此，输出结果是这些回路的逻辑或。如果一个回路是 ON，这个输出就是 ON。



操作数类型	D4-454 范围
	aaa
输出 Q	0-1777

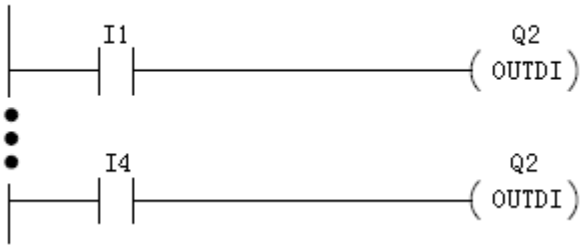
下面的例子中，当 I1 为 ON 时，Q2 被接通。



语句视图

```
LD  I1
ZDI Q2
```

下面的例子中，当 I1 或 I4 为 ON 时，Q2 被接通。



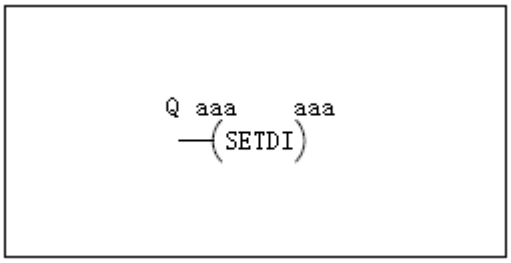
语句视图

```
LD  I1
OUTDI Q2

LD  I4
OUTDI Q2
```

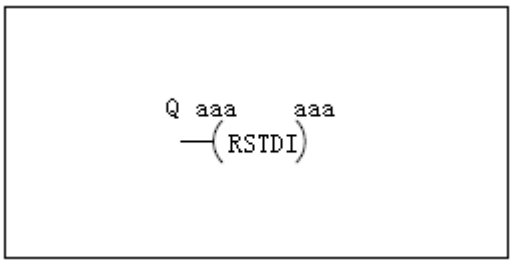
4.9 线圈直接置位指令（SETDI）

SETDI 指令，当其执行时，立即将指定的一个或一定范围的输出点置位。对应的映像寄存器也被刷新。输出被置位后就保持 ON 状态，直到 RSTDI 指令将其复位。



4.10 线圈直接复位指令（RSTDI）

RSTDI 指令，当其执行时，立即将指定的一个或一定范围的输出点复位。对应的映像寄存器也被刷新。输出被复位后就保持 OFF 状态。



操作数类型	D4-454 范围
	aaa
输出 Q	0-1777

下面的例子中，当 I1 为 ON 时，输出 Q5-Q22 被置位。

语句视图

```
LD I1
SETDI Q5 Q22
```

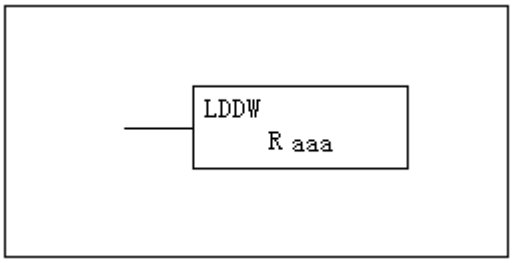
下面的例子中，当 I1 为 ON 时，输出 Q5-Q22 被复位。

语句视图

```
LD I1
RSTDI Q5 Q22
```

4.11 直接 16 位读入指令（LDDW）

LDDW 指令将一个 16 位 R 寄存器数据读入累加器。本体基架上的所有输入点地址都是有效地址。此值是 LDDW 指令执行时，输入点的当前值，此指令可替代 LDDF 指令。

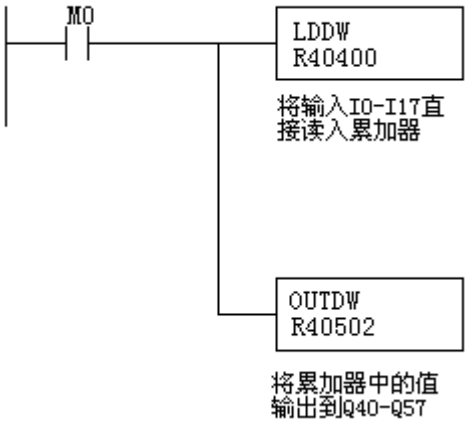


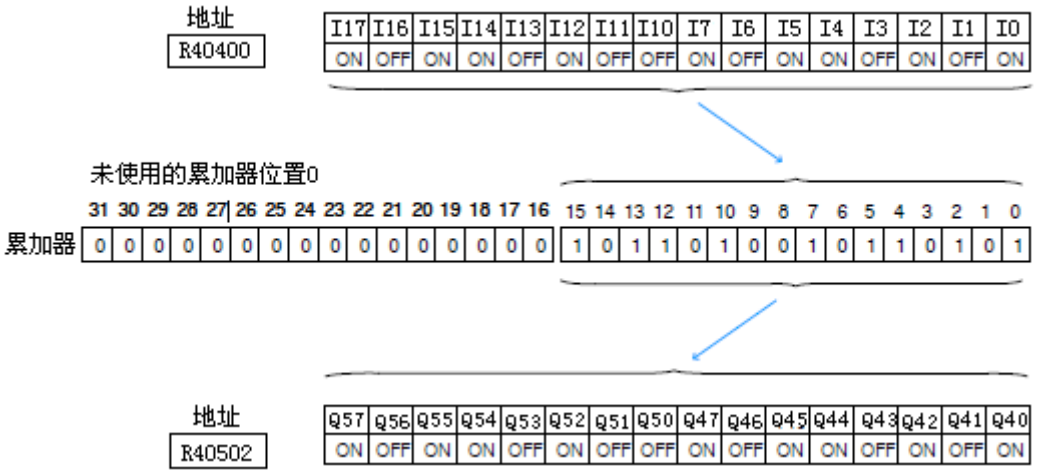
操作数类型	D4-454 范围
	aaaaa
寄存器 R	40400-40477

下面的例子中，当 M0 为 ON 时，输入 I0-I17 被 LDDW 指令读入累加器，OUTDW 指令将累加器中的数据直接输出到输出点，本例中是 Q40-Q57。这种方法用于将输入快速输出到输出点（无需等待 CPU 扫描周期的完成）。

语句视图

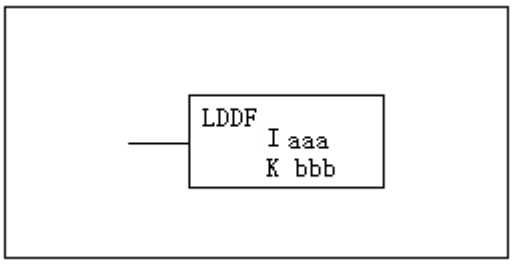
```
LD  M0
LDDW  R40400
OUTDW  R40502
```





4.12 直接读入任意位指令（LDDF）

LDDF 指令将 1-32 位的二进制数据读入累加器，此数据是 LDDF 指令执行时输入模块的当前值。累加器的未使用位被设置为 0。

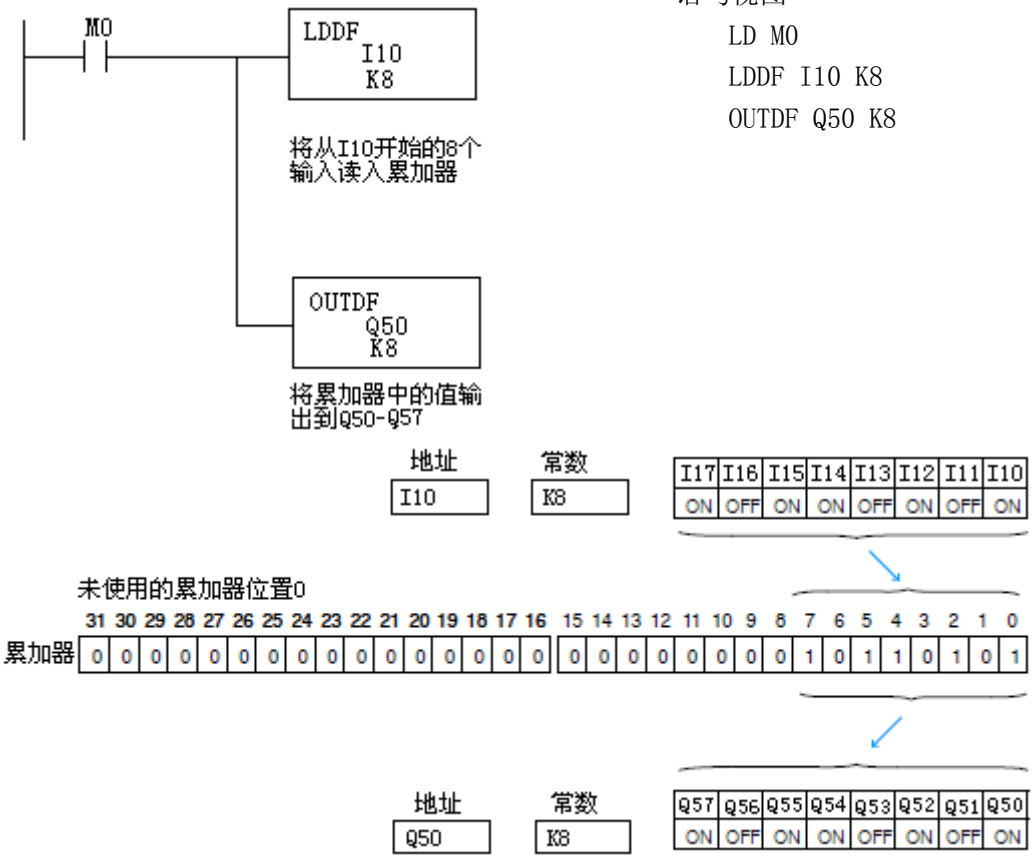


操作数类型		D4-454 范围	
		aaa	bbb
输入	I	0-1777	-
常数	K	-	1-32

下面的例子中，当 M0 为 ON 时，输入 I10-I17 被 LDDF 指令读入累加器，OUTDF 指令将累加器中的数据输出到 Q50-Q57。这种方法用于将输入点快速输出到输出点（无需等待 CPU 扫描周期的完成）。

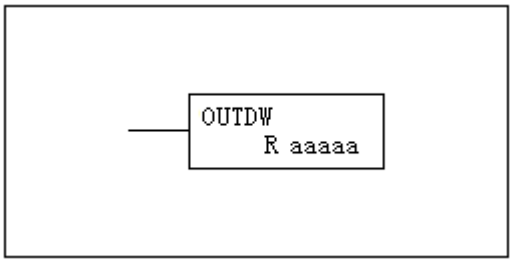
语句视图

```
LD M0
LDDF I10 K8
OUTDF Q50 K8
```



4.13 直接 16 位输出指令（OUTDW）

OUTDW 指令，当其执行时，将一个 16 位二进制数据输出到 R 寄存器中。本体框架上的所有输出点地址都是有效地址。此指令可替代 OUTDF 指令。

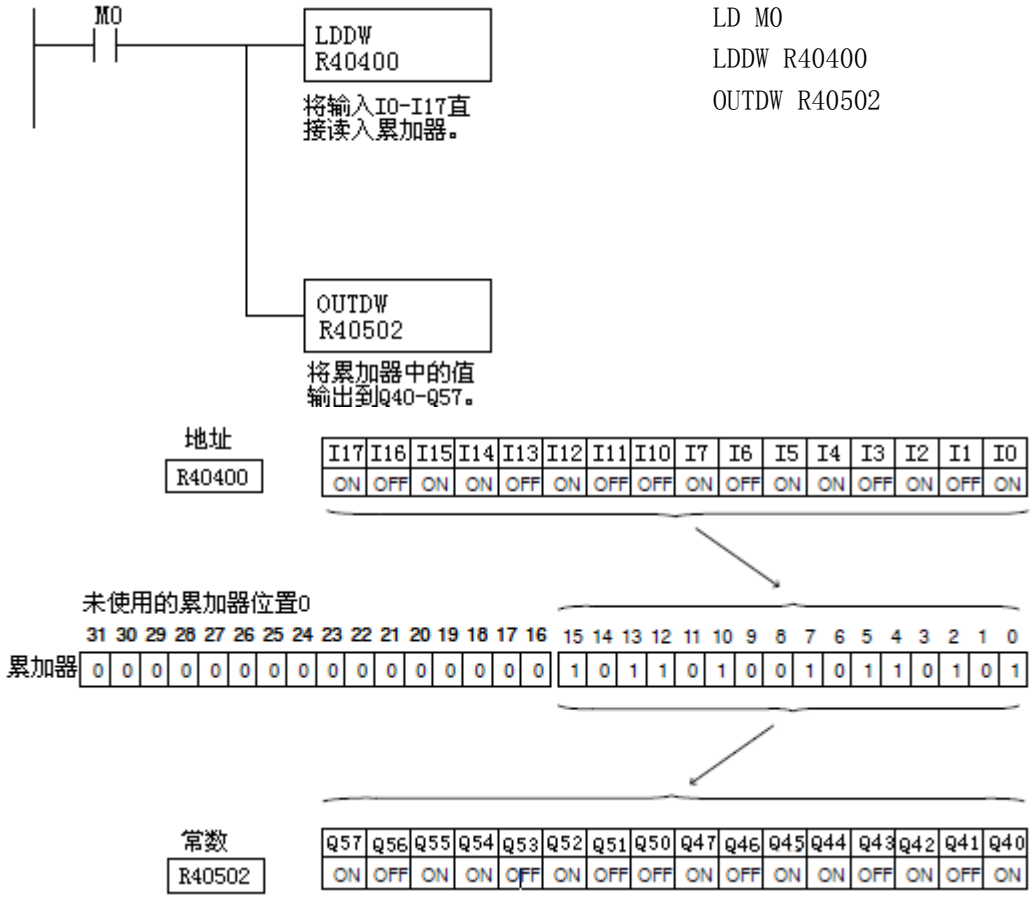


操作数类型	D4-454 范围
	aaaaa
寄存器 R	40500-40577

下面的例子中，当 M0 为 ON 时，I0-I17 被 LDDW 指令读入累加器，OUTDW 指令直接将累加器的数据读出到 Q40-Q57。这种方法用于将输入点快速输出到输出点（无需等待 CPU 扫描周期的完成）。

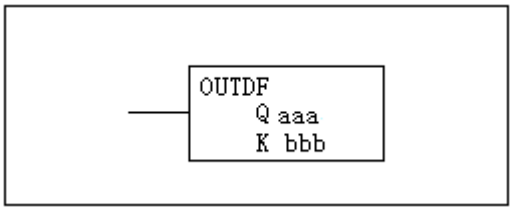
语句视图

```
LD M0
LDDW R40400
OUTDW R40502
```



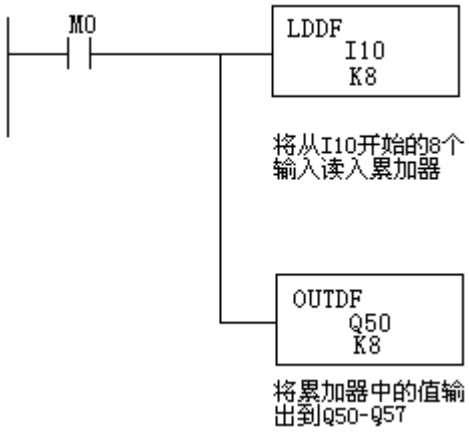
4.14 直接输出任意位指令（OUTDF）

OUTDF 指令，当指令执行时，将 1-32 位二进制数据从累加器中输出到指定输出点中。未使用的累加器位被设置为 0。



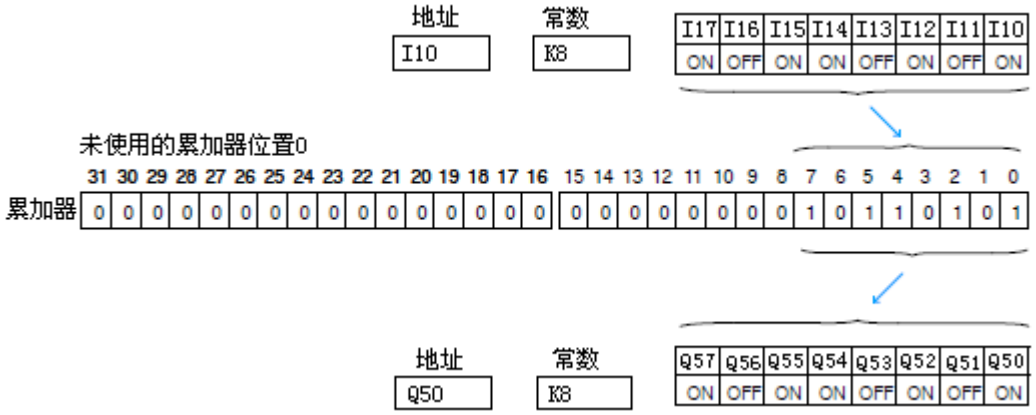
操作数类型		D4-454 范围	
		aaa	bbb
输出	Q	0-1777	-
常数	K	-	1-32

下面的例子中，当 M0 为 ON 时，输入 I10-I17 被 LDDF 指令读入累加器，OUTDF 指令将累加器中的数据输出到 Q50-Q57。这种方法用于将输入点快速输出到输出点（无需等待 CPU 扫描周期的完成）。



语句视图

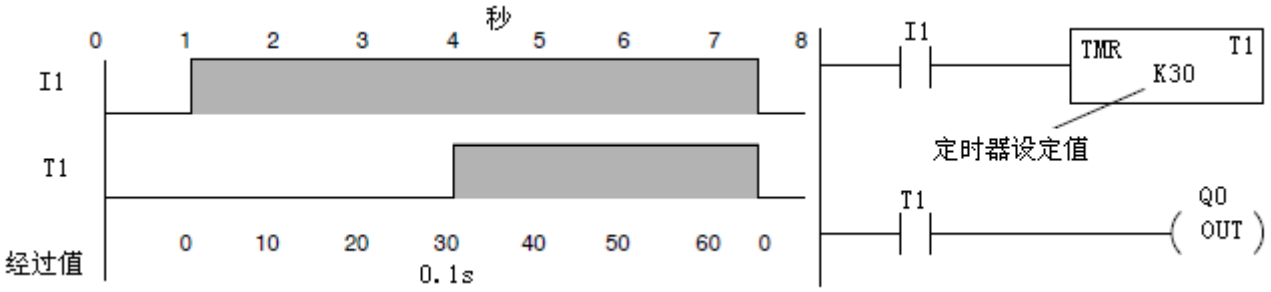
```
LD M0
LDDF I10 K8
OUTDF Q50 K8
```



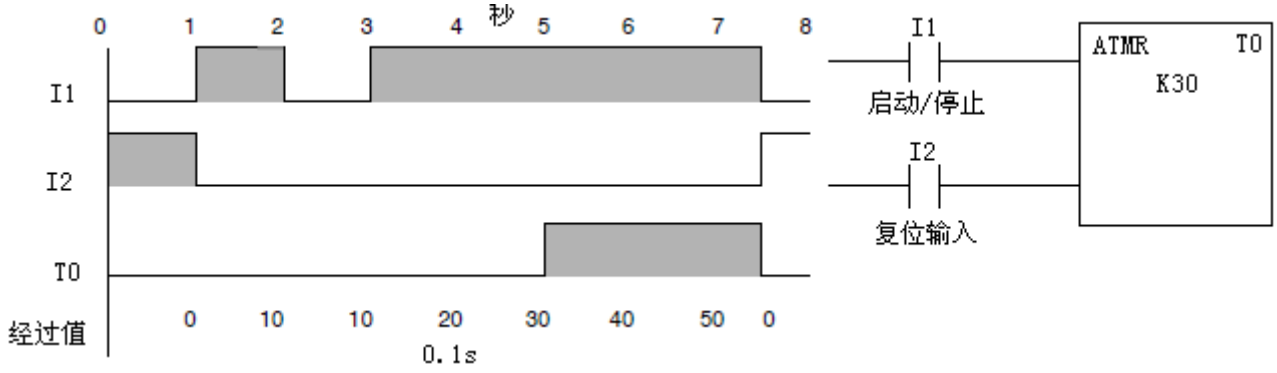
第 5 章 定时器、计数器、移位寄存器指令

5.1 定时器的使用方法

定时器用于给事件规定一个期望的时间。单输入定时器只要输入是 ON 就计时，当输入由 ON 变为 OFF，定时器经过值被复位为 0。有 0.1 秒和 0.01 秒定时器，定时最大值分别为 999.9 秒和 99.99 秒。当定时器经过值等于或大于设定值时，定时器会接通一个开关量。下图显示了定时器输入、经过值、设定值和定时器接通点之间的关系。



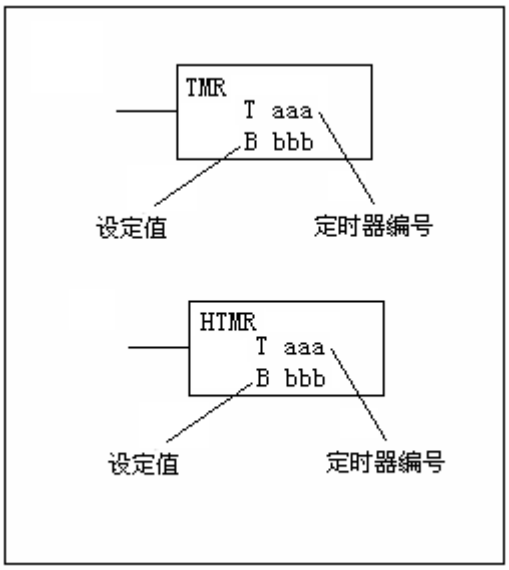
有些应用需要累积定时器，累积定时器可以计时、停止、然后再从先前停止的地方开始计时。累积定时器同普通定时器相似，但是需要两个输入。启动/停止输入用于启动和停止定时器。计时器停止时，计时经过值保持不变。当计时重新开始时，计时经过值从原来停止的地方累加。当复位输入变为 ON 时，计时经过值被复位为 0。有 0.1 秒和 0.01 秒累积定时器，定时最大值分别为 9999999.9 秒和 999999.99 秒。下图显示了定时器输入、复位条件、经过值、设定值和定时器接通点之间的关系。



注意：定时器设置时不使用小数点，小数点隐含在数值中。四种定时器的预置值与经过值为 BCD 格式。

5.2 0.1 秒定时器（TMR）和 0.01 秒定时器（HTMR）

0.1 秒定时器 TMR 为单输入，以 0.1 秒为单位增计时，定时范围 0-999.9s。0.01 秒定时器 HTMR 为单输入，以 0.01 秒为单位增计时，定时范围 0-99.99s。当输入为 ON 时，这些定时器增计时。当输入为 OFF 时，定时器经过值复位。两种定时器使用单字 BCD 值来表示预置值与经过值，隐含小数点。



定时器指令上接通点和经过值没有标示

指令详述：

定时器编号（T aaa）：指定定时器编号

设定值（B bbb）：常数（K）或寄存器地址。

经过值：计时器经过值以 BCD 格式，存储在相对应的 R 寄存器或 TA 存储地址中。例如，定时器 T3 的经过值存放在 R3 中。

定时器接通点：经过值等于或大于设定值时，定时器接通点接通，例如，定时器 2 接通时 T2 为 ON。

注意：仅当程序或操作单元需要改变设定值时，才将设定值放入寄存器中。

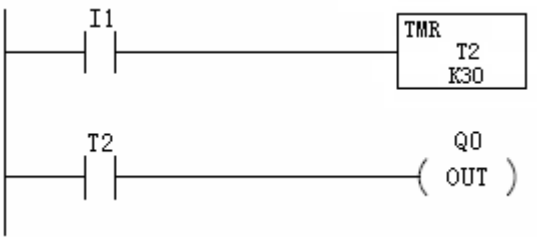
操作数类型		D4-454 范围	
	A/B	aaa	bbb
定时器	T	0-377	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-9999

注意：定时器使用不同的存储区，例如：T2 表示定时器 2 的连接触点，TA2 表示定时器 2 的经过值。

定时器有两种用法，一种是使用定时器接通点触发某一事件，一种是使用比较指令在同一个定时器的不同时间点进行比较来触发事件。下面有这两种方法的编程举例。

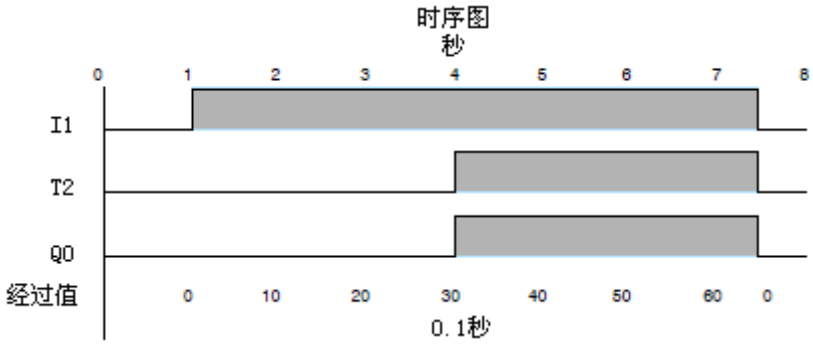
5.2.1 使用定时器接通点的程序举例

下面的例子中，定时器设定值是 3 秒，当定时器计时到 3 秒时，定时器接通点 T2 变为 ON。当 I1 变为 OFF 时，定时器经过值复位，接通点 T2 变为 OFF。



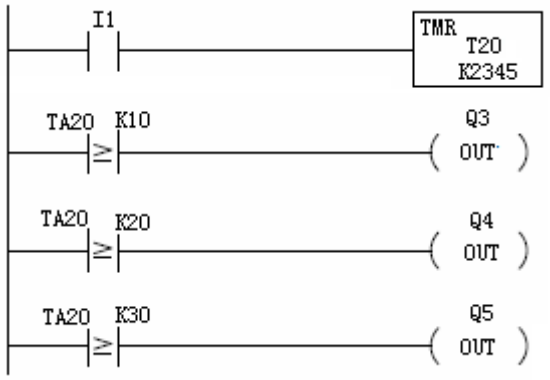
语句视图

```
LD I1
TMR T2 K30
LD T2
OUT Q0
```



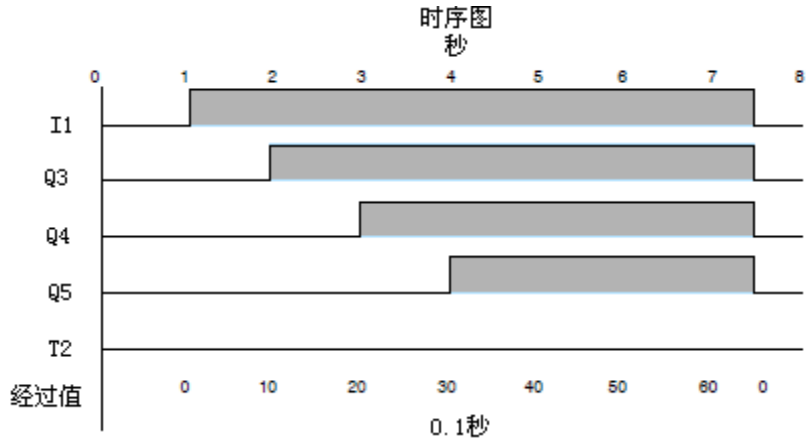
5.2.2 使用比较指令的程序举例

下面的例子中，定时器设定值是 234.5 秒，比较触点用来以 1 秒的间隔接通 Q3、Q4 和 Q5。当 I1 变为 OFF 时，定时器经过值被复位为 0，比较触点将 Q3、Q4 和 Q5 变为 OFF。



语句视图

```
LD I1
TMR T20 K2345
LDGE TA20 K10
OUT Q3
LDGE TA20 K20
OUT Q4
LDGE TA20 K30
OUT Q5
```



5.3 0.1 秒累积定时器（ATMR）和 0.01 秒累积定时器(AHTMR)

0.1 秒累积定时器 ATMR 以 0.1 秒为单位进行增计时，定时范围为 0-9999999.9。ATMR 占用两个 R 寄存器。

0.01 秒累积定时器 AHTMR 以 0.01 秒为单位进行增计时，定时范围为 0-999999.99。AHTMR 占用两个 R 寄存器。

这两种定时器有两个输入：启动和复位。当启动输入为 ON 时计时开始；当启动输入为 OFF 时计时停止，但是经过值保持而不复位。当复位输入为 ON 时经过值复位；当复位输入为 OFF 时计时允许。

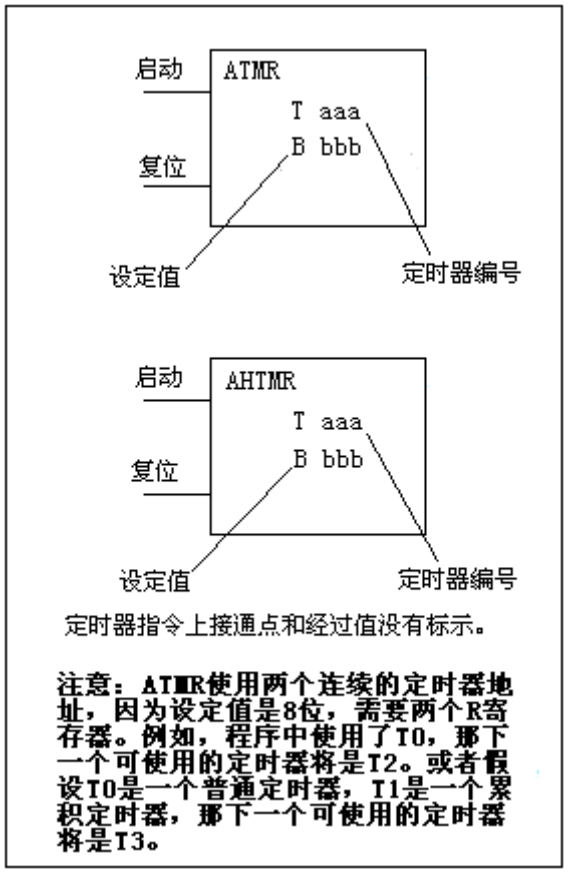
指令详述：

定时器编号（T aaa）：指定定时器编号

设定值（B bbb）：常数（K）或两个连续的寄存器地址（R）。

经过值：计时器经过值是一个双字，存储在相对应的 R 寄存器或 TA 存储地址中。例如，定时器 T3 的经过值存放在 R3 和 R4 中。

定时器接通点：经过值等于或大于设定值时，定时器接通点接通，例如，定时器 2 接通时 T2 为 ON。



注意：累积定时器占用两个连续的 R 寄存器，因此，累积定时器的编号不可连续，例如，使用了定时器 ATMR T1，那么下一个有效的定时器应该是 T3 而不是 T2。

注意：仅当程序需要改变设定值时，才将设定值放入寄存器中。

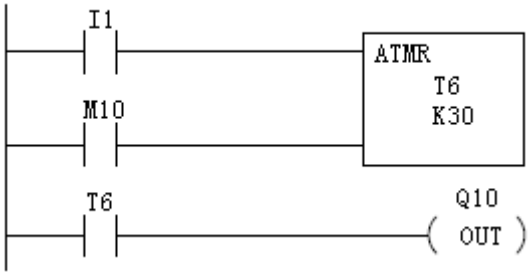
操作数类型		D4-454 范围	
	A/B	aaa	bbb
定时器	T	0-377	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-99999999

注意：定时器使用不同的存储区，例如：T2 表示定时器 2 的连接触点，TA2 表示定时器 2 的经过值。

定时器有两种用法，一种是使用定时器接通点触发某一事件，一种是使用比较指令在同一个定时器的不同时间点进行比较来触发事件。下面有这两种方法的编程举例。

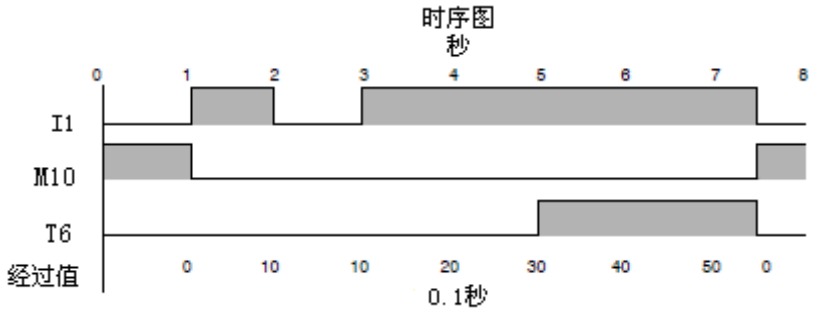
5.3.1 使用累积定时器接通点的程序举例

下面的例子中，累积定时器设定值是 3 秒，当定时器计时到 3 秒时，定时器接通点 T6 变为 ON。注意本例中，累积定时器先计时 1 秒，然后停 1 秒，然后再继续计时，当 M10 变为 ON 时，定时器经过值复位，接通点 T2 变为 OFF。



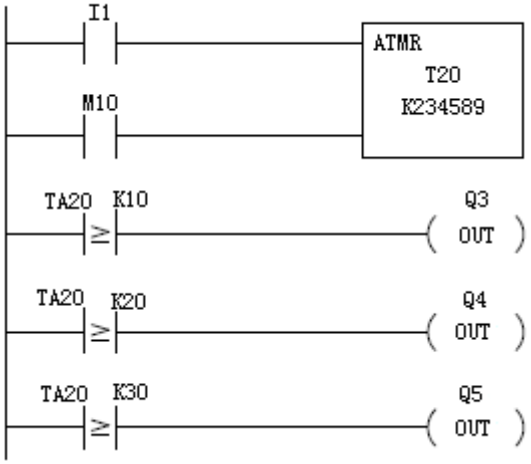
语句视图

```
LD I1
LD M10
ATMR T6 K30
LD T6
OUT Q10
```



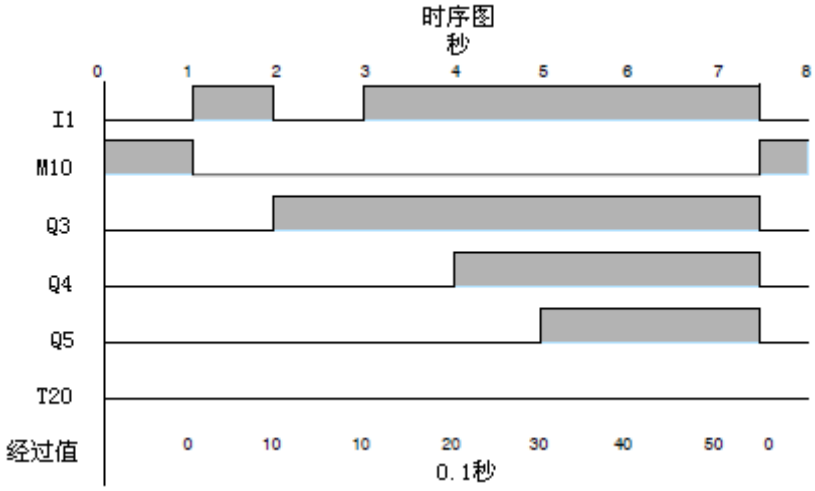
5.3.2 使用比较指令的程序举例

下面的例子中，累积定时器设定值是 23458.9 秒，比较触点以 1 秒的间隔来接通 Q3、Q4 和 Q5。当 M10 变为 ON 时，定时器经过值被复位为 0，比较触点将 Q3、Q4 和 Q5 变为 OFF。



语句视图

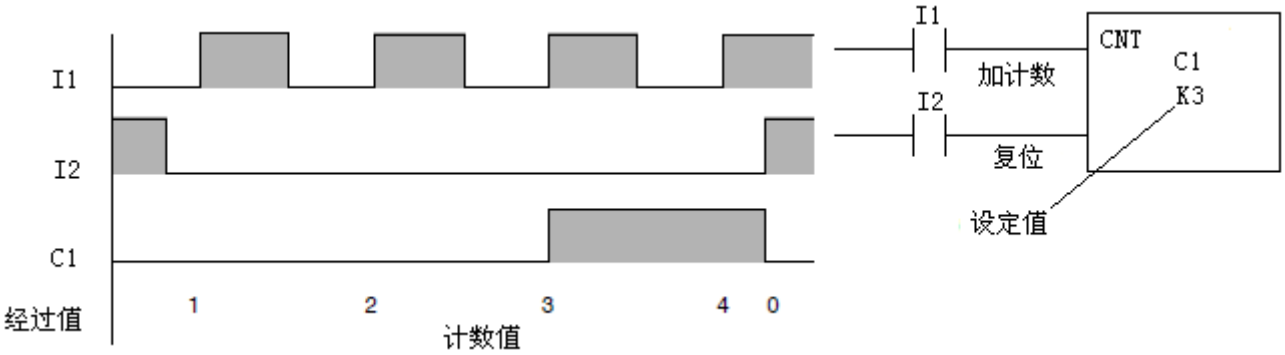
```
LD I1
LD M10
ATMR T20 K234589
LDGE TA20 K10
OUT Q3
LDGE TA20 K20
OUT Q4
LDGE TA20 K30
OUT Q5
```



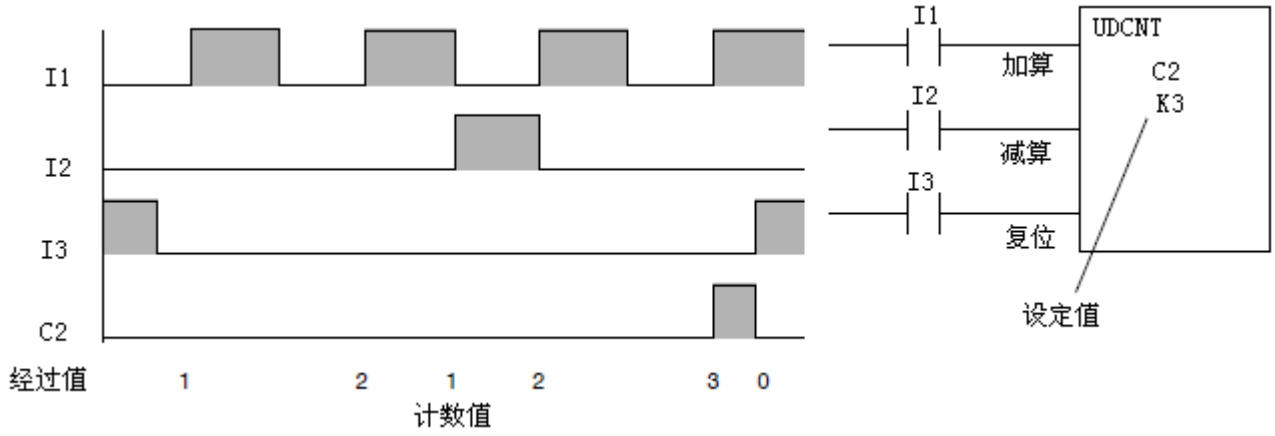
5.4 计数器的使用方法

计数器用于计数事件。计数器有加计数器、加减计数器和级式计数器（用于级式语言编程）。

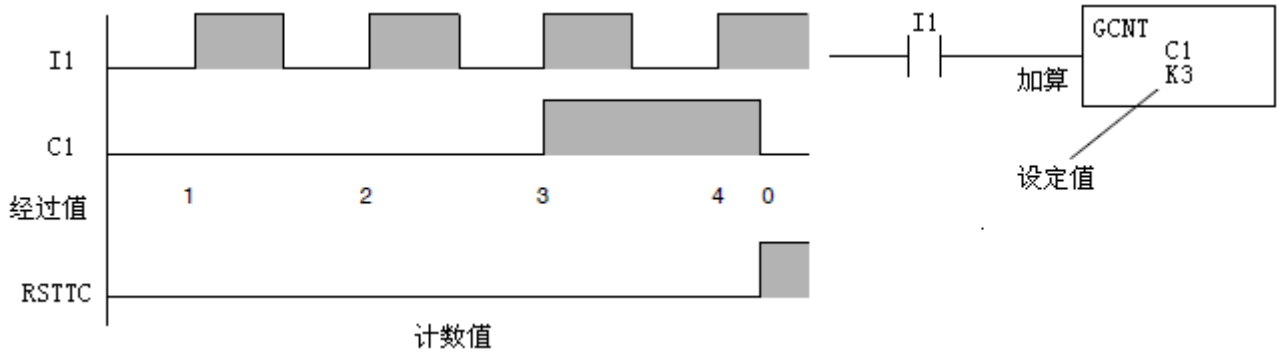
加算计数器(带复位端)有两个输入，一个计数输入一个复位输入。最大计数值是 9999。下面的时序图显示了计数输入、计数复位、对应开关量、经过值和设定值之间的关系。CNT 计数器的预置值与经过值都是单字 BCD 数。



加减计数器有三个输入：加算输入、减算输入和复位输入。最大计数值是 99999999。下面的时序图显示了计数输入、计数复位、对应开关量、经过值和设定值之间的关系。UDCNT 计数器的预置值与经过值都是双字 BCD 数。

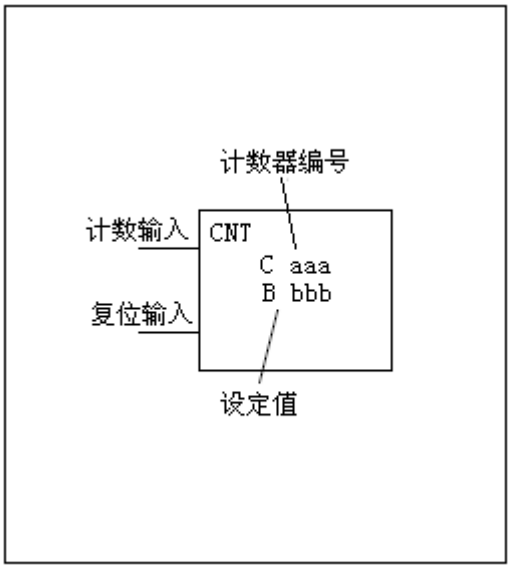


加算计数器（不带复位端）有一个计数输入，由 RSTTC 指令复位。这个计数器用于级式语言编程，最大计数值是 9999，下面的时序图演示了计数输入、对应开关量、经过值、设定值和复位指令之间的关系。



5.5 加算计数器(带复位端)（CNT）

CNT 指令有两个输入，当计数输入由 OFF 变为 ON 时开始加计数；当复位输入为 ON 时，计数值复位到 0。当经过值等于设定值时，对应开关量变为 ON，计数值继续增加到最大值 9999。计数最大值保持到计数器被复位。



计数器指令上接通点和经过值没有标示

指令详述：

计数器编号（C aaa）：指定计数器编号

设定值（B bbb）：常数（K）或 R 寄存器地址。

经过值：计数器经过值存储在相对应的 R 寄存器或 CA 存储地址中。R 寄存器地址是计数器编号+1000。例如，计数器 C3 的经过值存放在 R1003 中。

计数器接通点：经过值等于或大于设定值时，计数器接通点接通，例如，计数器 2 接通时 C2 为 ON。

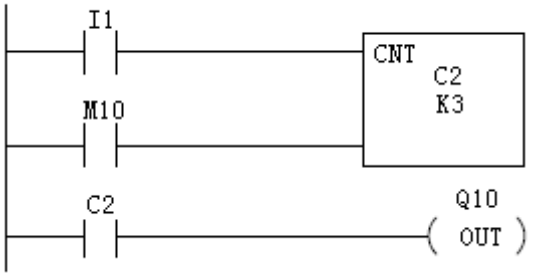
注意：仅当程序需要改变设定值时，才将设定值放入寄存器中。

操作数类型		D4-454 范围	
	A/B	aaa	bbb
计数器	C	0-377	-
设定值寄存器	R	-	所有（附录 1）
指针(仅设定值)	P	-	所有（附录 1）
常数(仅设定值)	K	-	0-9999

注意：计数器使用不同的存储区，例如：C2 表示计数器 2 的连接触点，CA2 表示计数器 2 的经过值。

5.5.1 使用计数器接通点的程序举例

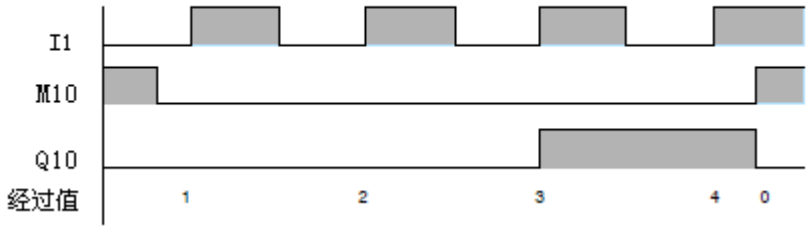
下面的例子中，每当 I1 由 OFF 变为 ON 时，计数器 C2 就加 1。当计数值等于设定值 3 时，计数器接通点 C2 变为 ON，Q10 被接通。当复位信号 M10 变为 ON 时，计数器接通点变为 OFF，经过值复位为 0。计数器 C2 的经过值保存在寄存器 R1002 中。



语句视图

```
LD I1
LD M10
CNT C2 K3
LD C2
OUT Q10
```

时序图

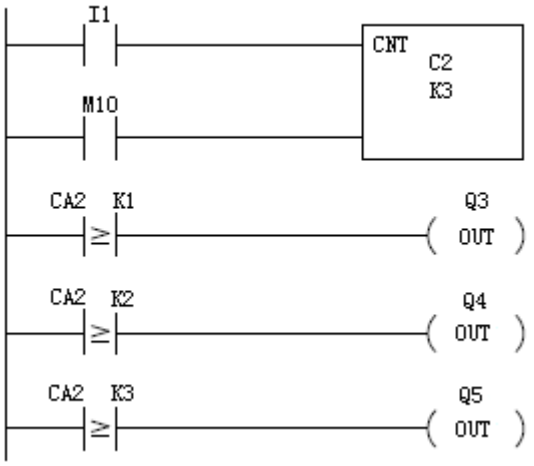


5.5.2 使用比较指令的程序举例

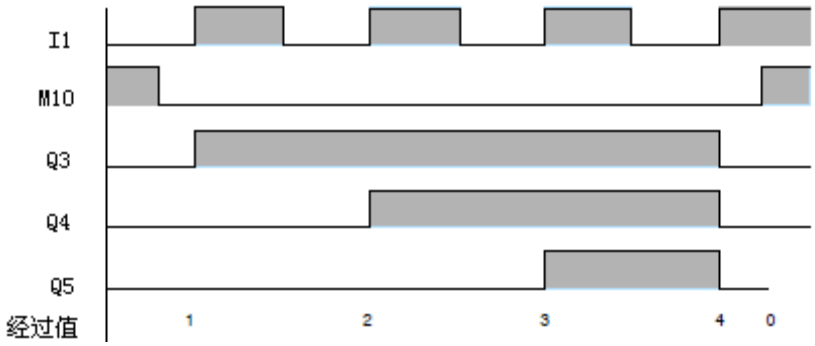
下面的例子中，每当 I1 由 OFF 转变为 ON，计数器 C2 将加 1。比较触点用于在不同的计数值将 Q3、Q4 和 Q5 接通。当计数器被复位时，比较触点将变为 OFF。当复位输入 M10 变为 ON 时，计数器接通点将变为 OFF，计数经过值被复位为 0。

语句视图

```
LD I1
LD M10
CNT C2 K3
LDGE CA2 K1
OUT Q3
LDGE CA2 K2
OUT Q4
LDGE CA2 K3
OUT Q5
```

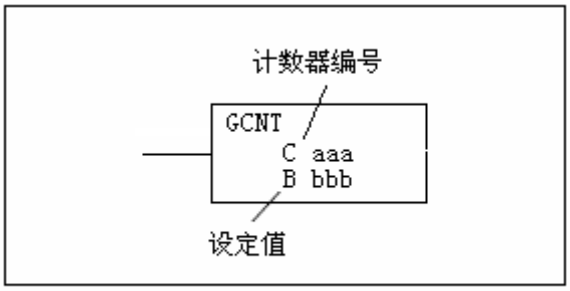


时序图



5.6 加算计数器(不带复位端)（GCNT）

GCNT 指令只有一个输入，当这个输入由 OFF 变为 ON 时开始加计数。这个计数器区别于其它计数器，因为它将保持经过值直到 RSTTC 指令将其复位。GCNT 指令设计用于级式语言编程，也可以用于一般梯形图程序中。当经过值等于设定值时，对应开关量变为 ON，计数值继续增加到最大值 9999。计数最大值保持到计数器被复位。



计数器指令上接通点和经过值没有标示。

指令详述：

计数器编号（C aaa）：指定计数器编号

设定值（B bbb）：常数（K）或 R 寄存器地址。

经过值：计数器经过值存储在相对应的 R 寄存器或 CA 存储地址中。R 寄存器地址是计数器编号+1000。例如，计数器 C3 的经过值存放在 R1003 中。

计数器接通点：经过值等于或大于设定值时，计数器接通点接通，例如，计数器 2 接通时 C2 为 ON。

注意：当将 GCNT 用于级中时，在 GCNT 指令的输入发生一个 0→1 的转换时，其所在的级至少被激活了一个扫描周期，否则，相当于输入没有发生 0→1 的转换，计数器将不计数。

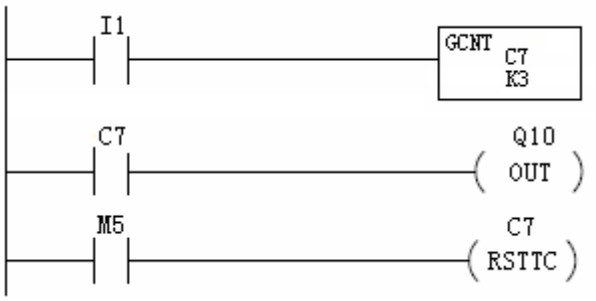
注意：仅当程序或操作单元需要改变设定值时，才将设定值放入寄存器中。

操作数类型		D4-454 范围	
	A/B	aaa	bbb
计数器	C	0-377	-
设定值寄存器	R	-	所有（附录 1）
指针(仅设定值)	P	-	所有（附录 1）
常数(仅设定值)	K	-	0-9999

注意：计数器使用不同的存储区，例如：C2 表示计数器 2 的连接触点，CA2 表示计数器 2 的经过值。

5.6.1 使用计数器接通点的程序举例

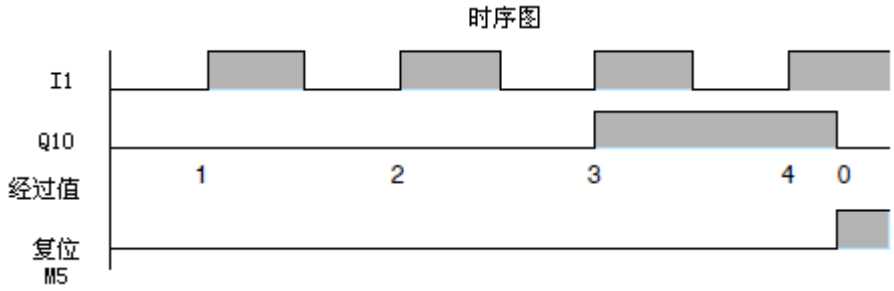
下面的例子中，每当 I1 由 OFF 变为 ON 时，计数器 C7 就加 1。当计数值等于设定值 3 时，计数器接通点 C7 变为 ON，Q10 被接通。计数器接通点一旦接通就保持 ON 状态直到使用 RSTTC 指令将其复位。计数器被复位后，经过值被复位为 0。计数器 C7 的经过值保存在寄存器 R1007 中。



语句视图

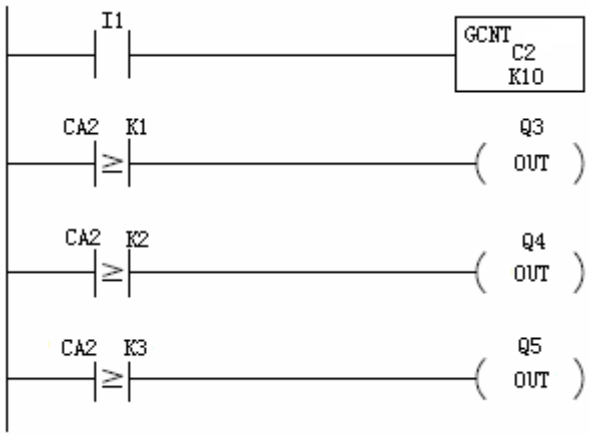
```
LD I1
GCNT C7 K3
LD C7
OUT Q10
LD M5
RSTTC C7
```

时序图



5.6.2 使用比较指令的程序举例

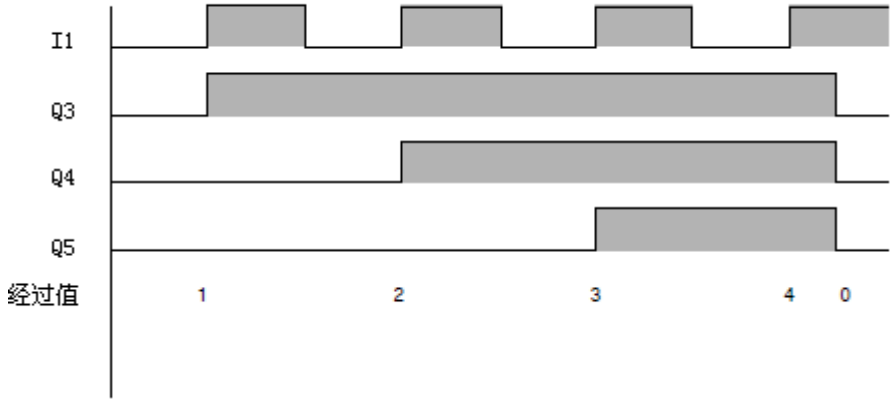
下面的例子中，每当 I1 由 OFF 变为 ON，计数器 C2 将加 1。比较触点用于在不同的计数值将 Q3、Q4 和 Q5 接通。当计数器被复位时，比较触点将变为 OFF，计数经过值被复位为 0，计数经过值被存放在 R1002 中。



语句视图

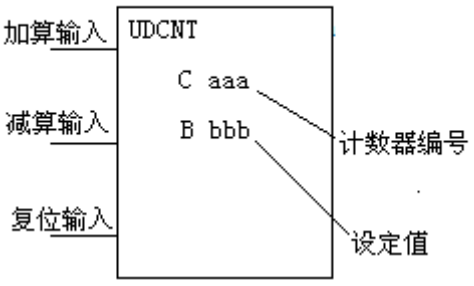
```
LD I1
GCNT C2 K10
LDGE CA2 K1
OUT Q3
LDGE CA2 K2
OUT Q4
LDGE CA2 K3
OUT Q5
```

时序图



5.7 加减计数器（UDCNT）

UDCNT 指令，当加算输入由 OFF 变为 ON 时开始加计数；当减算输入由 OFF 变为 ON 时开始减计数；当复位输入为 ON 时，计数值复位为 0。计数范围是 0-99999999。不使用的计数输入要保持 OFF 状态以使计数功能有效。



定时器指令上接通点和经过值没有标示

注意：UDCNT指令使用两个连续的计数器地址，因为设定值是8位，需要两个R寄存器。例如，程序中使用C0，那下一个可使用的定时器将是C2。或者假设C0是一个普通计数器，C1是一个加减计数器，则下一个可使用个计数器将是C3。

指令详述：

计数器编号（C aaa）：计数器编号

设定值（B bbb）：常数（K）或两个连续的 R 寄存器地址。

经过值：计数器经过值是一个双字值，存储在相对应的 R 寄存器或 CA 存储地址中。R 寄存器地址是计数器编号+1000。例如，计数器 C5 的经过值存放在 R1005 和 R1006 中。

计数器接通点：经过值等于或大于设定值时，计数器接通点接通，例如，计数器 C2 接通时 C2 为 ON。

注意：加减计数器 UDCNT 占用两个连续的 R 寄存器，因此，加减计数器的编号不可连续，例如，使用了计数器 UDCNT C1，那么下一个有效的计数器应该是 C3 而不是 C2。

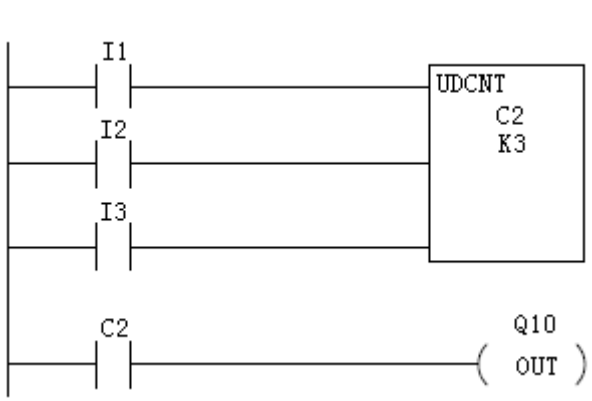
注意：仅当程序或操作单元需要改变设定值时，才将设定值放入寄存器中。

操作数类型		D4-454 范围	
	A/B	aaa	bbb
计数器	C	0-376	-
设定值寄存器	R	-	所有（附录 1）
指针（仅设定值）	P	-	所有（附录 1）
常数（仅设定值）	K	-	0-99999999

注意：计数器使用不同的存储区，例如：C2 表示计数器 2 的连接触点，CA2 表示计数器 2 的经过值。

5.7.1 使用计数器接通点的程序举例

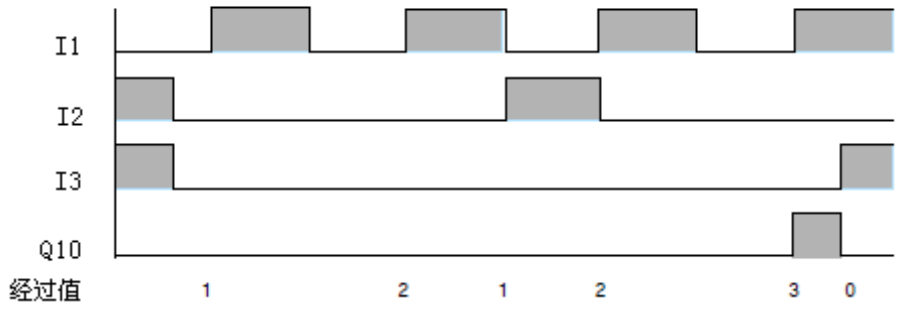
下面的例子中，如果 I2 和 I3 为 OFF，当 I1 由 OFF 变为 ON 时，计数器 C2 就加 1。如果 I1 和 I3 为 OFF，当 I2 由 OFF 变为 ON 时，计数器 C2 就减 1。当计数值等于设定值 3 时，计数器接通点 C2 变为 ON，Q10 被接通。当复位信号 I3 变为 ON 时，计数器接通点变为 OFF，经过值复位为 0。



语句视图

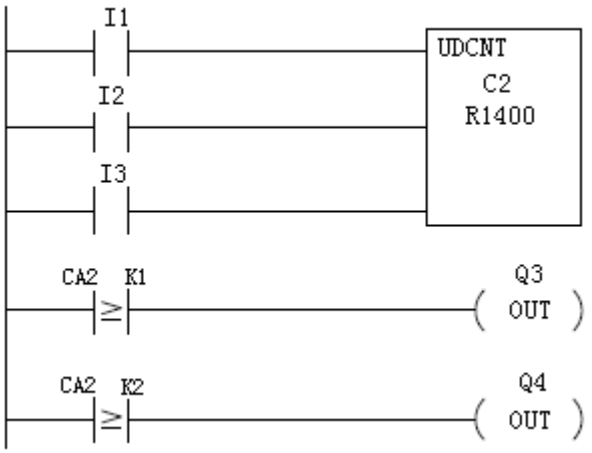
```
LD I1
LD I2
LD I3
UDCNT C2 K3
LD C2
OUT Q10
```

时序图



5.7.2 使用比较指令的程序举例

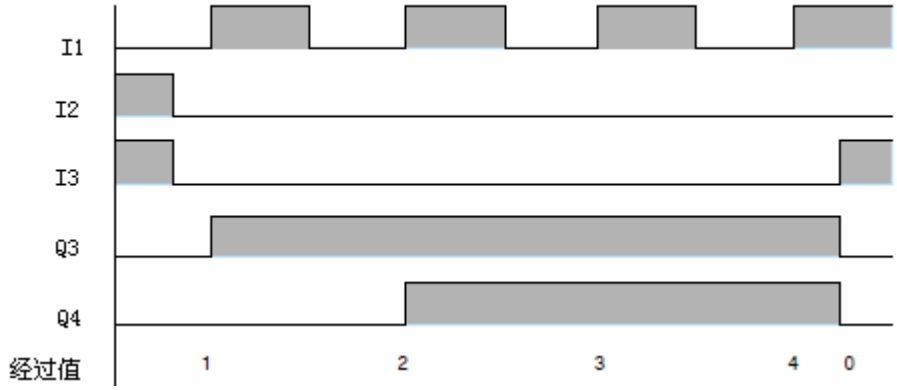
下面的例子中，每当 I1 由 OFF 变为 ON，计数器 C2 将加 1。比较触点用于在不同的计数值将 Q3 和 Q4 接通。当复位输入 I3 变为 ON 时，计数器接通点将变为 OFF，计数经过值被复位为 0。



语句视图

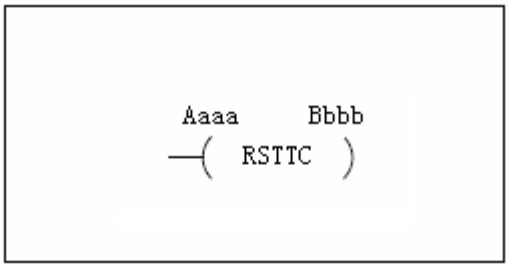
```
LD I1
LD I2
LD I3
UDCNT C2 R1400
LDGE CA2 K1
OUT Q3
LDGE CA2 K2
OUT Q4
```

时序图



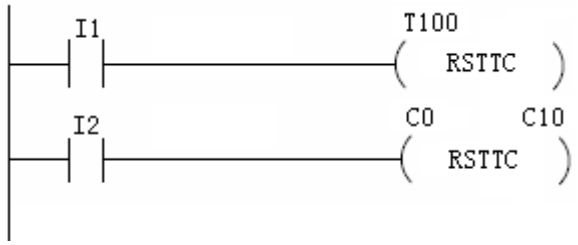
5.8 定时器、计数器复位指令（RSTTC）

RSTTC 指令是使定时器/计数器的经过值复位的指令。其可以是单步指令，也可以是两步指令（对一个范围复位）。执行此指令后，定时器/计数器的经过值复位为“0”。



操作数类型		D4-454 范围	
	A/B	aaa	bbb
定时器	T	0-377	0-377
计数器	C	0-377	0-377

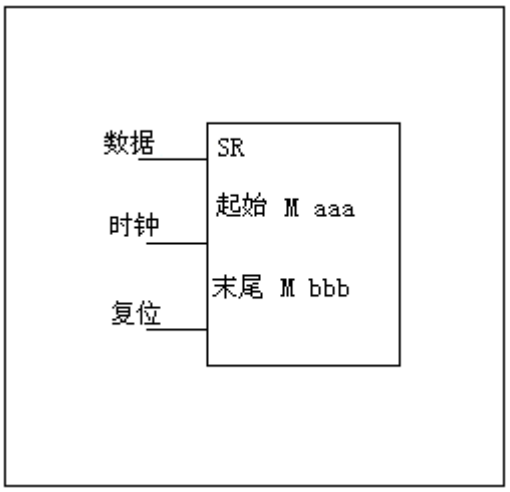
下面的例子中，当 I1 为 ON 时，定时器 T100 的经过值为复位为 0；当 I2 为 ON 时，计数器 C0-C10 的经过值被复位为 0.。



语句视图

```
LD I1
RSTTC T100
LD I2
RSTTC C0 C10
```

5.9 移位寄存器指令（SR）



SR 指令将预先确定范围的内部线圈移位。内部线圈范围以 8 点为单位，起始点和末尾点必须是 8 点的两端，也就是说如果起始定位号是 M×××0，则末尾定义号为 M×××7，移位方向是 M×××0→M×××7；如果起始定位号是 M×××7，则末尾定义号为 M×××0，移位方向是 M×××7→M×××0。

SR 指令的三个输入：

数据：决定了要进入存储器的值（是 1 或是 0）。

时钟：当时钟输入由 OFF→ON 时，将数据输入的状态送入起始位，同时移位寄存器内各位的状态均向下移 1 位。

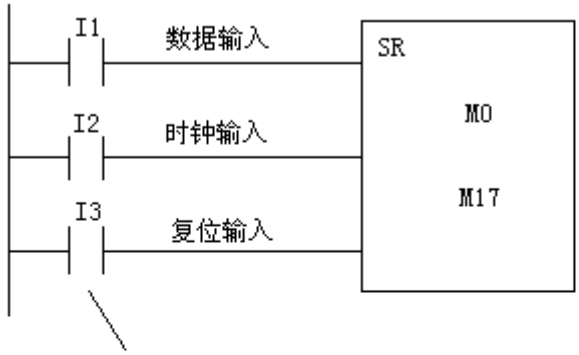
复位：将移位寄存器复位。

当时钟输入由 OFF→ON 时，将数据输入的状态送入起始位，同时移位寄存器内各位的状态向下移 1 位，移位方向取决于起始和末尾地址。例如，假设起始定义号为 M0，末尾定义号为 M17，则这 16 位由低地址位向高地址位移动；假设起始定义号为 M17，末尾定义号为 M0，则这 16 位由高地址位向低地址位移动。最大移位范围取决于可用内部线圈地址范围，最小移位范围是 8 点内部线圈位。

操作数类型		D4-454 范围	
		aaa	bbb
内部线圈	M	0-3777	0-3777

语句视图

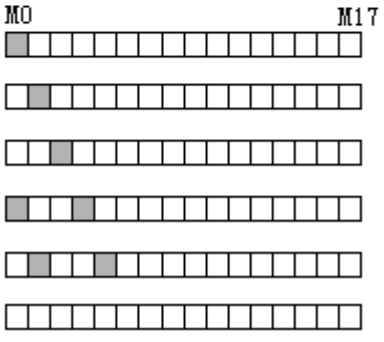
```
LD  I1
LD  I2
LD  I3
SR  M0  M17
```



正常扫描时的各输入

数据	时钟	复位
1	⌐¬	0
0	⌐¬	0
0	⌐¬	0
1	⌐¬	0
0	⌐¬	0
-	-	1

移位寄存器位



第 6 章 累加器/数据堆栈和输出指令

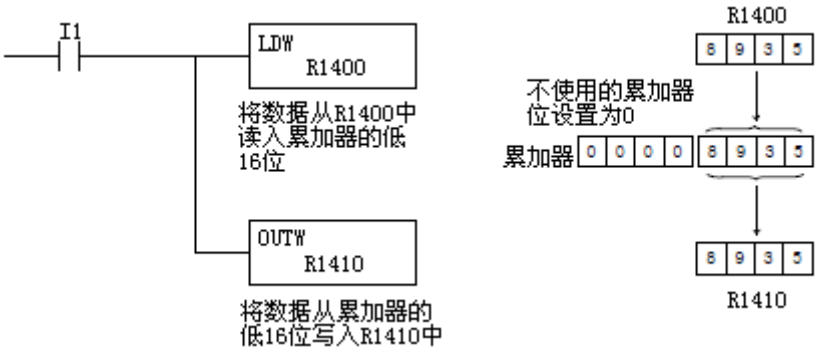
6.1 累加器、堆栈的使用方法

6.1.1 累加器的使用方法

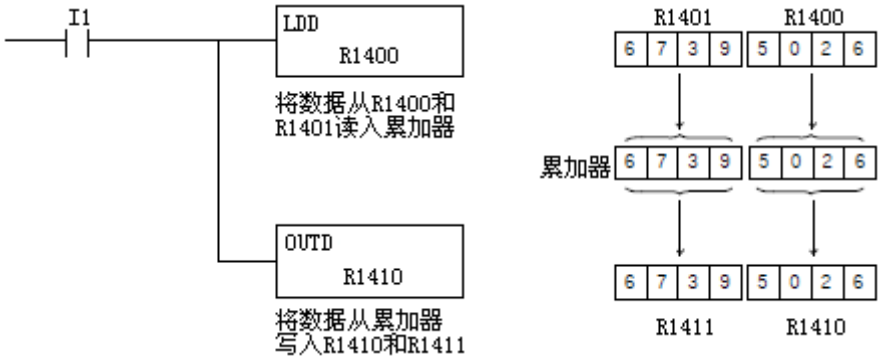
累加器是一种特殊寄存器，由 32 位组成，临时存储数据处理结果。例如，必须使用累加器来完成加、减、除等数学运算。因为累加器是 32 位，可以存放一个 8 位 BCD 数，或是一个 32 位二进制补码数。每个 CPU 扫描周期结束累加器都被复位为 0。

数据读入累加器

读入和写入指令用于将数据从 R 寄存器读入累加器或是将数据从累加器写入 R 寄存器。下面的例子将数据从 R1400 写入 R1410。

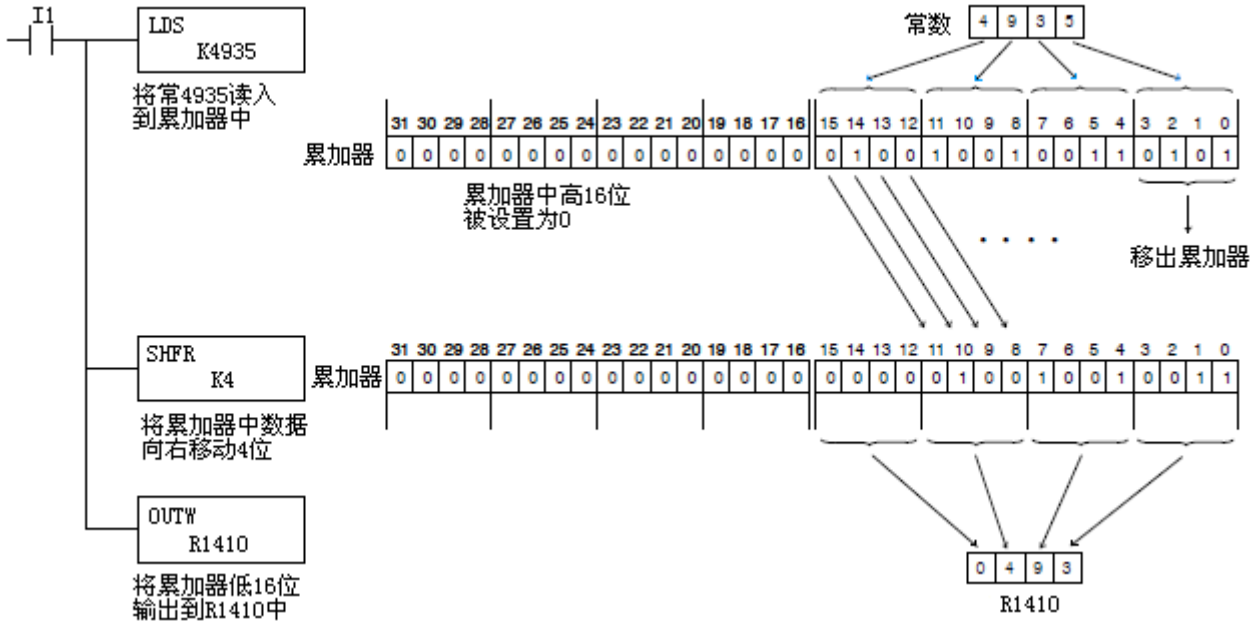


由于累加器是 32 位，R 寄存器是 16 位，因此 LDD 和 OUTD 指令（及其类似指令）可将两个连续的 R 寄存器数据或一个 8 位 BCD 数读入累加器或从累加器中写入到 R 寄存器中。例如，如果想把 R1400 和 R1401 中的数据读入到 R1410 和 R1411 中，最简单有效的方法如下：

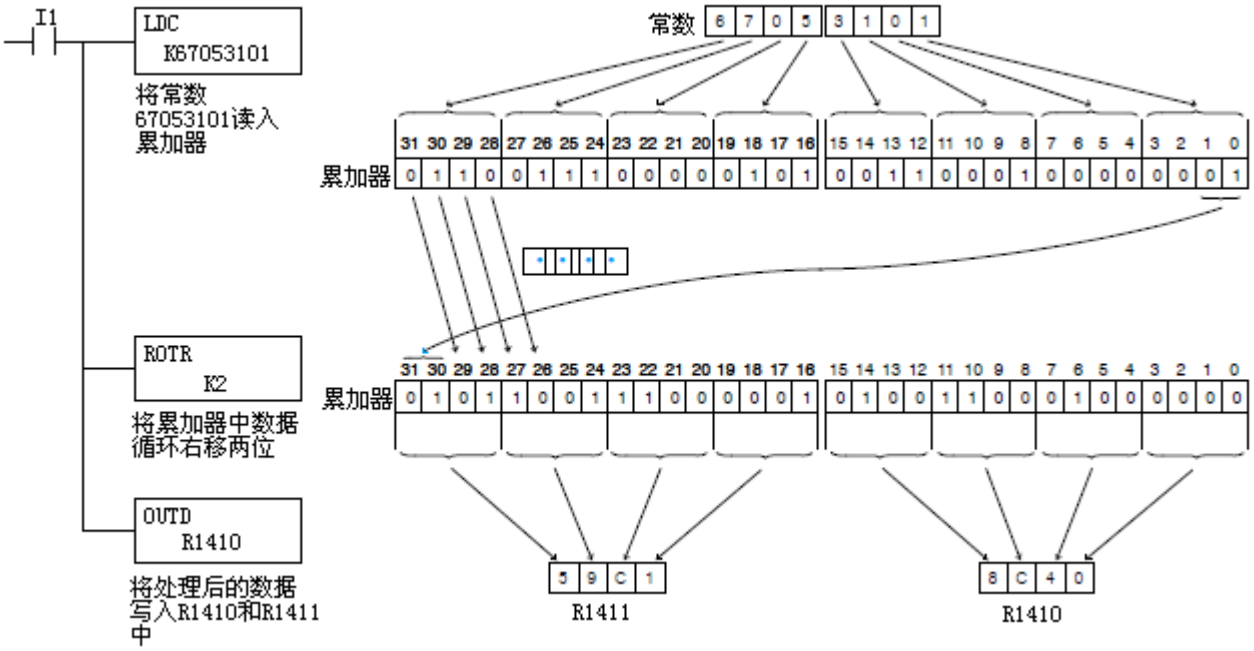


改变累加器数据

处理数据的指令也要用到累加器。数据在累加器中处理，处理过的数据暂时存放在累加器中，下面的例子中，将 BCD 常数 4935 读入累加器中，右移 4 位后将结果输出到 R1410 中。

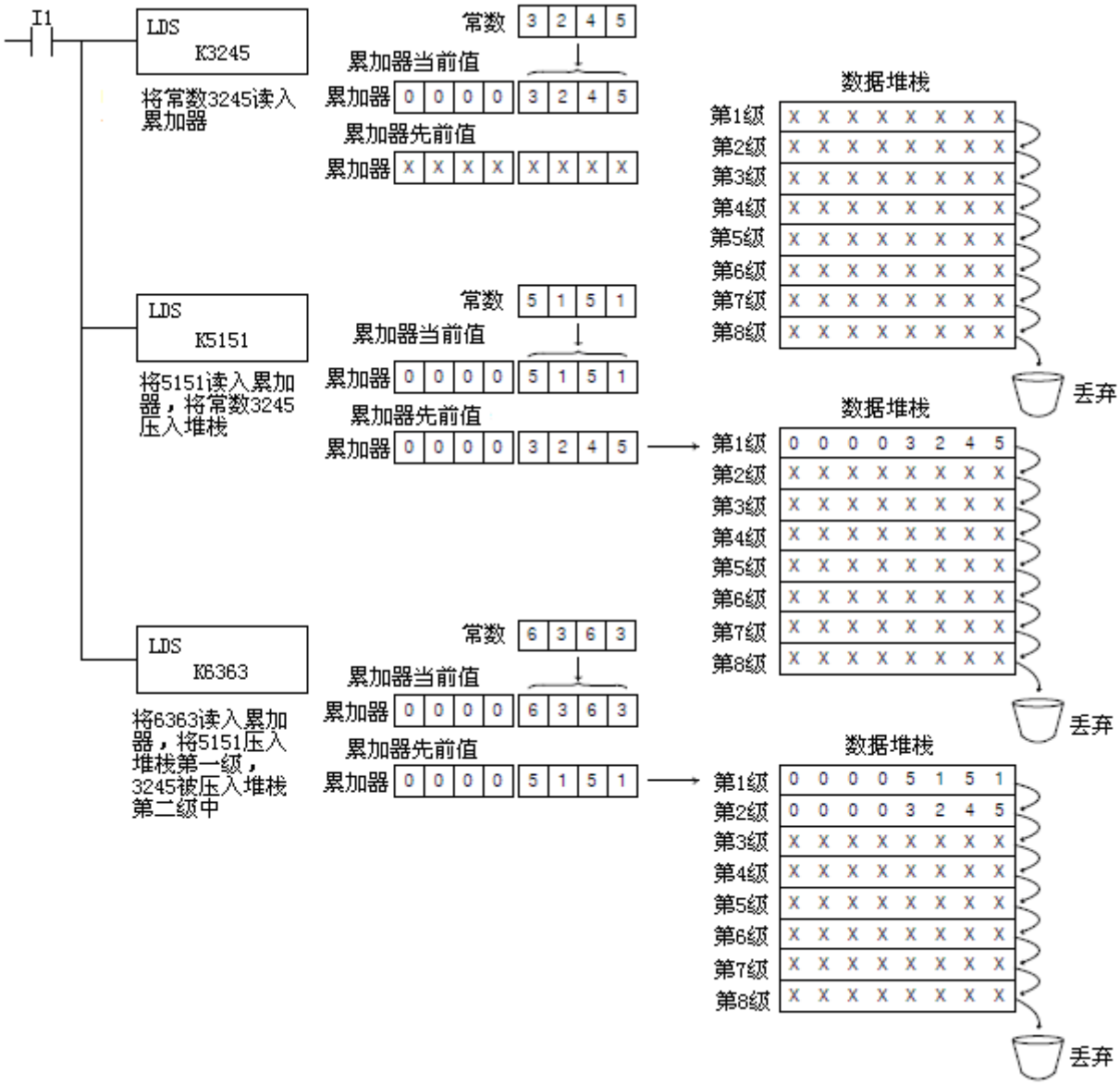


有些处理数据的指令会使用到累加器的 32 位。这些指令处理累加器中的两个连续的 R 寄存器数据或一个 8 位 BCD 数。下面的例子中，I1 为 ON 时，将 BCD 数 67053101 循环右移两位后输出到 R1410 和 R1411 中。

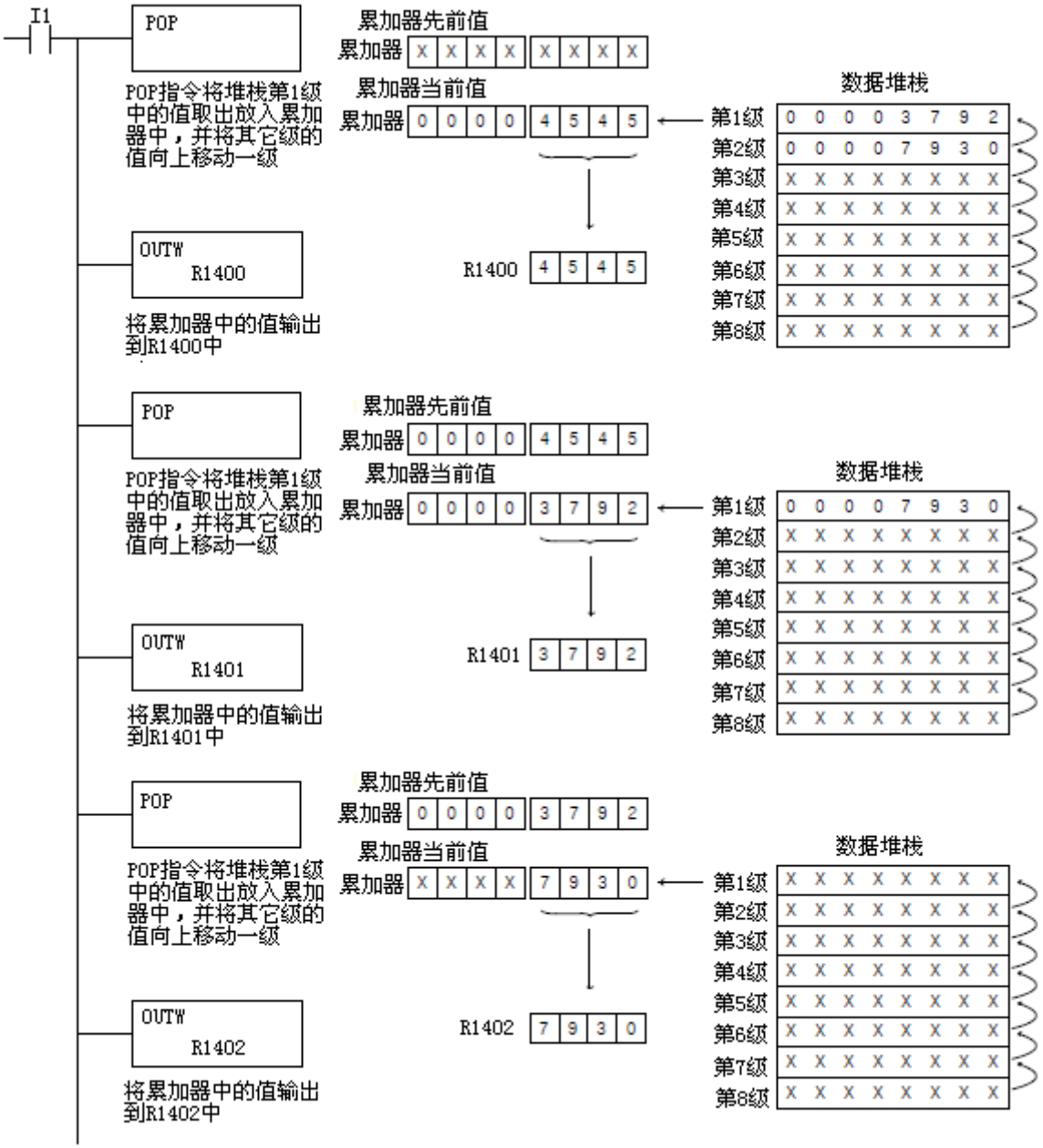


6.1.2 数据堆栈的使用方法

数据堆栈用于需要一个以上的参数来执行一个功能或用户自定义功能的指令。数据堆栈用于有多个读入指令执行，但是没有使用输出指令的场合。第一个读入指令将一个数据读入累加器，之后每个不使用输出指令的读入指令都将数据读入累加器，而累加器中的原有数据被压入数据堆栈。输出指令使其前面一个读入指令失效，当下一个读入指令执行时，不会将累加器中的数据压入数据堆栈中。每次将一个数据压入数据堆栈，堆栈中原有的数据就被压入下一级。数据堆栈有 8 级（8 个 32 位寄存器）。如果第 8 级中有数据，当一个新数据被压入堆栈后，第 8 级中的数据被压出堆栈，并且不能恢复。



POP 指令将堆栈第 1 级的数据取出放入累加器中，累加器中记忆的内容在执行 POP 指令之后丢失，堆栈中 2-8 级的内容依次上移一级。


I1
POP
POP指令将堆栈第1级中的值取出放入累加器中，并将其它级的值向上移动一级
累加器先前值
累加器 X X X X X X X X
累加器当前值
累加器 0 0 0 0 4 5 4 5
数据堆栈
第1级 0 0 0 0 3 7 9 2
第2级 0 0 0 0 7 9 3 0
第3级 X X X X X X X X
第4级 X X X X X X X X
第5级 X X X X X X X X
第6级 X X X X X X X X
第7级 X X X X X X X X
第8级 X X X X X X X X
OUTW R1400
R1400 4 5 4 5
将累加器中的值输出到R1400中
POP
POP指令将堆栈第1级中的值取出放入累加器中，并将其它级的值向上移动一级
累加器先前值
累加器 0 0 0 0 4 5 4 5
累加器当前值
累加器 0 0 0 0 3 7 9 2
数据堆栈
第1级 0 0 0 0 7 9 3 0
第2级 X X X X X X X X
第3级 X X X X X X X X
第4级 X X X X X X X X
第5级 X X X X X X X X
第6级 X X X X X X X X
第7级 X X X X X X X X
第8级 X X X X X X X X
OUTW R1401
R1401 3 7 9 2
将累加器中的值输出到R1401中
POP
POP指令将堆栈第1级中的值取出放入累加器中，并将其它级的值向上移动一级
累加器先前值
累加器 0 0 0 0 3 7 9 2
累加器当前值
累加器 X X X X 7 9 3 0
数据堆栈
第1级 X X X X X X X X
第2级 X X X X X X X X
第3级 X X X X X X X X
第4级 X X X X X X X X
第5级 X X X X X X X X
第6级 X X X X X X X X
第7级 X X X X X X X X
第8级 X X X X X X X X
OUTW R1402
R1402 7 9 3 0
将累加器中的值输出到R1402中

6.1.3 累加器、数据堆栈中的数据地址

有时需要读取数据堆栈的数据，但是尚未将其从堆栈中弹出。累加器和数据堆栈中有相应的累加器地址，可以通过程序访问。

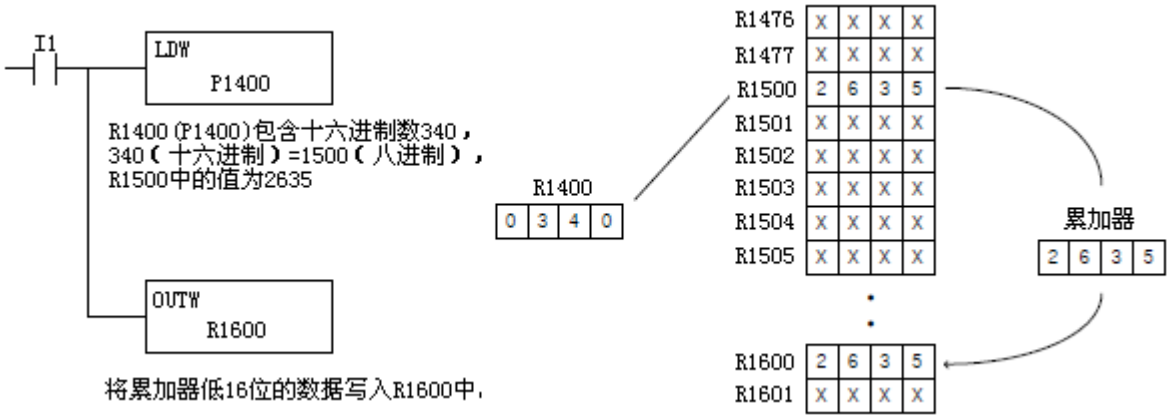
不可向这些地址写入数据，但是可以读取它们的数据，或将它们用于比较指令等。

6.1.4 指针的使用方法

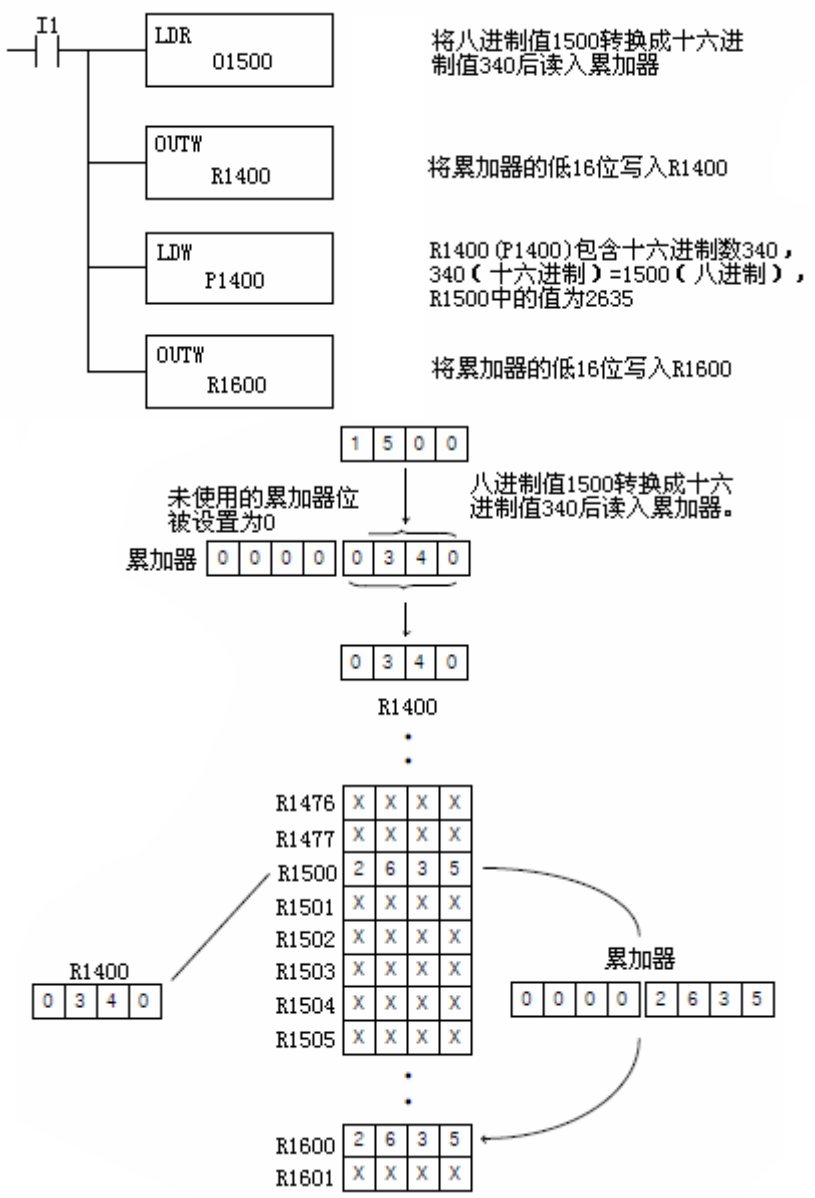
D4-454 CPU 的许多指令允许将 R 寄存器指针作为操作数。指针在梯形图编程中很有用，但是如果没有指针（也被称为间接寻址）的使用经验的话可能比较难理解。指针允许指令通过指针值获取 R 寄存器地址中的数据。

注意：D4-454 中 R 寄存器地址是八进制的，但是 R 寄存器中的地址值是十六进制值。使用 LDR 读入指令将地址值读入指针，指令自动完成八进制到十六进制的转换。

下面的例子中，在读入指令中使用指针作为操作数。R 寄存器地址 1400 是指针地址，R1400 中的数据 340 是十六进制值，其对应的八进制值是 1500。CPU 将 R1500 中的数据（2635）读入累加器的低位中。

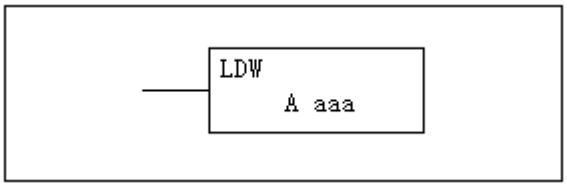


下面的例子同上面的例子类似，LDR 指令自动将八进制地址转换成十六进制数。



6.2 读入指令（LDW）

LDW 指令是一个 16 位指令，将指定 R 寄存器中的数据读入累加器的低 16 位中，高 16 位被设置为 0。

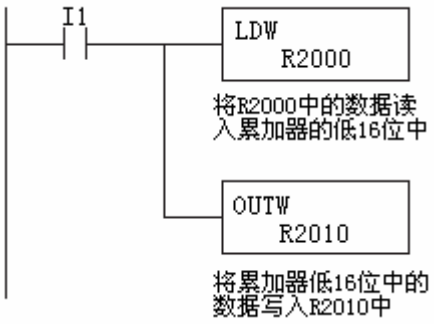


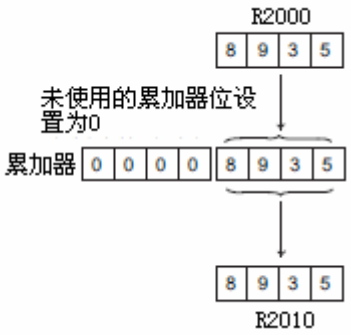
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP76	读入累加器的数据为 0 时 ON

注意：如果两个读入指令连续，则第一个读入指令的值将被压入数据堆栈的第 1 级中。

下面的例子中，当 I1 为 ON 时，R2000 中的数据被读入累加器中然后写入到 R2010 中。



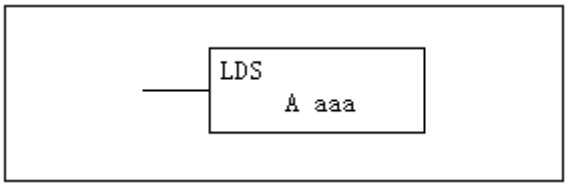


语句视图

```
LD I1
LDW R2000
OUTW R2010
```

6.3 读入指令（LDS）

LDS 指令是一个 16 位指令，将一个 4 位常数读入累加器的低 16 位中，高 16 位被设置为 0。



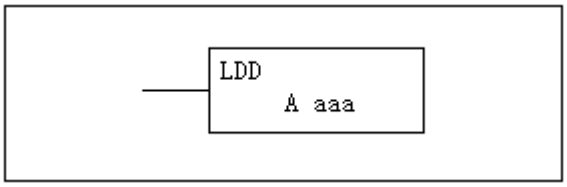
操作数类型		D4-454 范围
	A	aaa
常数	K	0-FFFF

受影响的标志线圈	描述
SP76	读入累加器的数据为 0 时 ON

注意：如果两个读入指令连续，则第一个读入指令的值将被压入数据堆栈的第 1 级中。

6.4 读入指令（LDD）

LDD 指令是一个 32 位指令，将指定的两个连续的寄存器中的数据读入累加器。

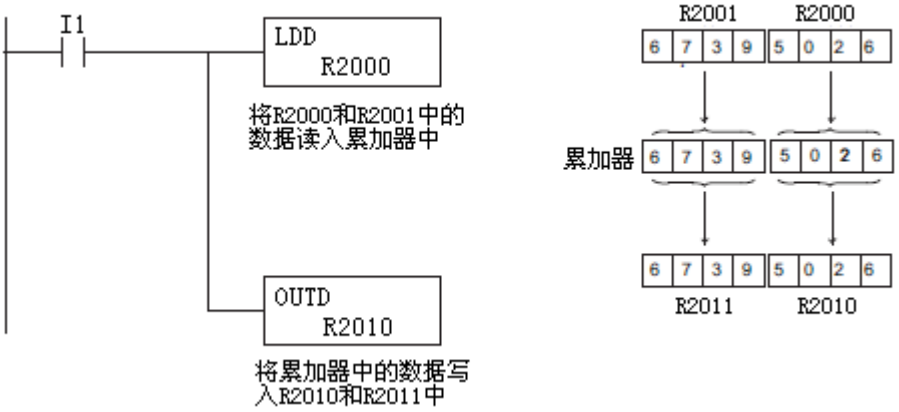


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP70	读入累加器的数据为负时 ON
SP76	读入累加器的数据为 0 时 ON

注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 ON 时，R2000 和 R2001 中的值将被装入累加器中然后输出到 R2010 和 R2011 中。

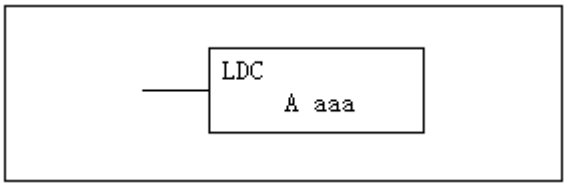


语句视图

```
LD I1
LDD R2000
OUTD R2010
```

6.5 读入指令（LDC）

LDC 指令是一个 32 位指令，将 8 位常数读入累加器。



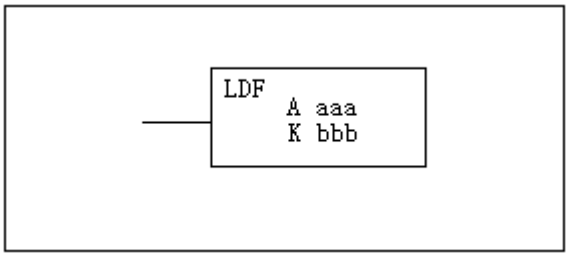
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFFFFFF

受影响的标志线圈	描述
SP70	读入累加器的数据为负时 ON
SP76	读入累加器的数据为 0 时 ON

注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

6.6 任意位读入指令（LDF）

LDF 指令用于将 1-32 个连续位读入累加器，指令需要一个起始地址（A aaa）和一个位数（K bbb），不使用的累加器位被设置为 0。

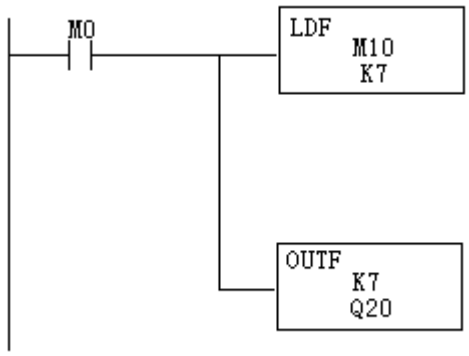


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器位	T	0-377	--
计数器位	C	0-377	--
特殊继电器	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

受影响的标志线圈	描述
SP70	读入累加器的数据为负时 ON
SP76	读入累加器的数据为 0 时 ON

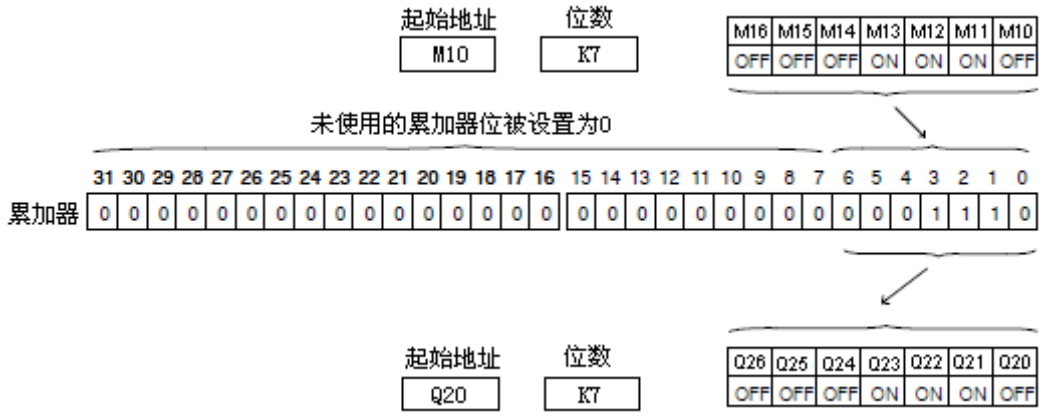
注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 M0 为 ON 时，LDF 指令将 M10-M16（7 位）读入累加器中，然后 OUTF 指令将累加器的低 7 位写入到 Q20-Q26 中。



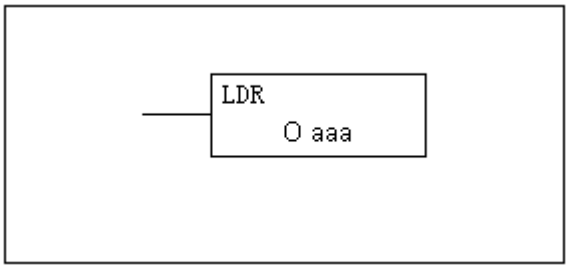
语句视图

```
LD M0
LDF M10 K7
OUTF Q20 K7
```



6.7 寄存器地址读入指令（LDR）

LDR 指令是一个 16 位指令，它将八进制值或地址转换为十六进制值或地址并将其写入累加器。当需要地址参数时，该指令很有用，因为 D4-454 系统的所有地址都是八进制的。

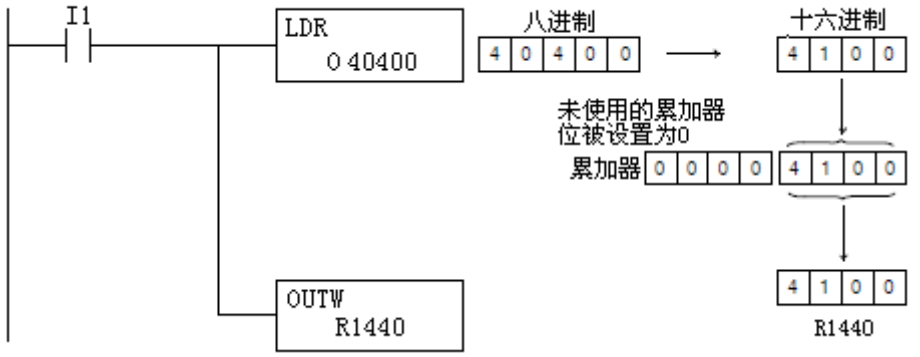


操作数类型		D4-454 范围
	A	aaa
八进制地址	O	所有（附录 1）

受影响的标志线圈	描述
SP76	读入累加器的数据为 0 时 ON

注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 ON 时，八进制数据 40400 将被转换为十六进制数据 4100 并被读入累加器中。累加器的低 16 位被写入 R1440 中。

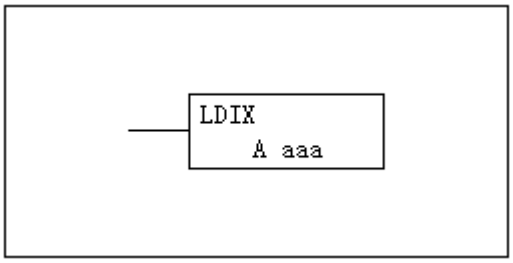


语句视图

```
LD  I1
LDR 040400
OUTW R1440
```

6.8 索引 16 位读入指令（LDIX）

LDIX 指令是一个 16 位指令，它指定一个源地址，这个源地址加上数据堆栈第 1 级中的偏移量得到一个新的寄存器地址，将这个新寄存器地址中的数据读入累加器的低 16 位中，累加器的高 16 位被设置为 0。此指令将堆栈第 1 级中的数据当作十六进制数。



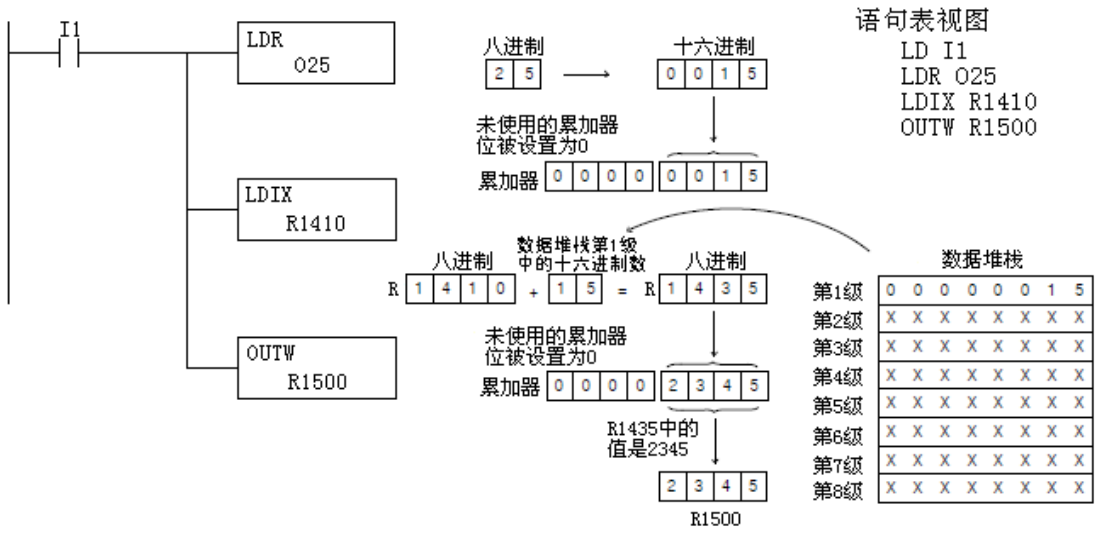
小技巧：LDR 指令可将八进制数转换为十六进制数并将其读入累加器。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）
指针 P	所有（附录 1）

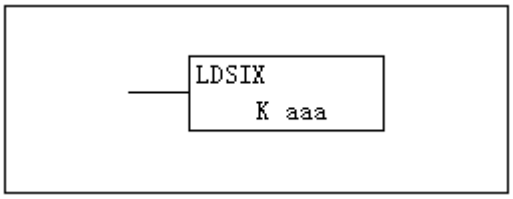
受影响的标志线圈	描述
SP76	读入累加器的数据为 0 时 ON

注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 ON 时，八进制数据 25 的十六进制数据 15 被装入累加器，当 LDIX 指令执行时，偏移量被压入数据堆栈的第 1 级中。R 寄存器地址 1410 加上第 1 级中的数据的八进制值 25 得到寄存地地址值 1435，将 R1435 中的数据读入累加器的低 16 位中，R1435 中的数据为 2345。OUTW 指令将累加器低 16 位的数据写入到 R1500 中。



6.9 程序块索引读入指令（LDSIX）



LDSIX 指令是一个 16 位指令，它指定了存放数字或 ASCII 常数的数据标记区的标号。

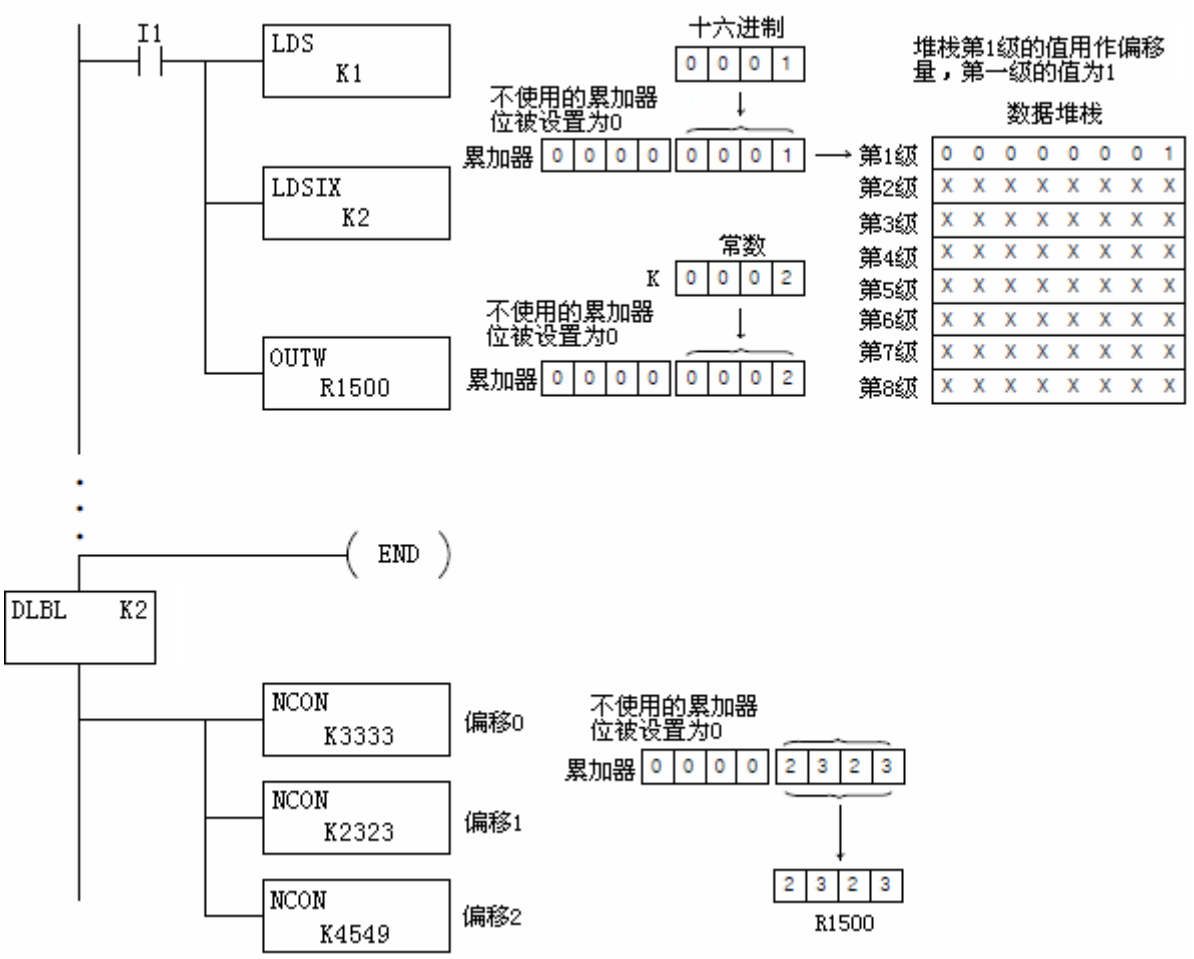
LDSIX 指令将数据堆栈第 1 级中的数据作为偏移量，这个偏移量指定了数据标记区中哪个数据将被写入累加器中。此指令将堆栈第 1 级中的数据当作十六进制数。

小技巧：LDR 指令可将八进制数转换为十六进制数并将其读入累加器。

操作数类型		D4-454 范围
	A	aaa
常数	K	1-FFFF

注意：如果有两个连续的读入指令，第一个读入指令的数据将被压入数据堆栈中。

下面的例子中，当 I1 为 ON 时，LDS 指令将偏移量常数 1 读入累加器，当 LDSIX 指令执行时，偏移量被压入数据堆栈的第 1 级中。LDSIX 指令指定了数据标记区的标号（DLBL K2），并将相应偏移值的数据读入累加器中。

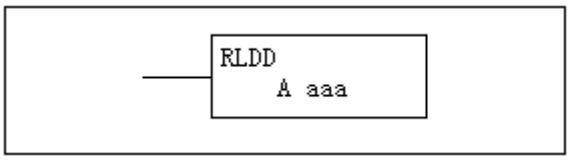


语句视图

```
LD I1
LDS K1
LDSIX K2
OUTW R1500
…
END
DLBL K2
NCON K3333
NCON K2323
NCON K4549
```

6.10 实数读入指令（RLDD）

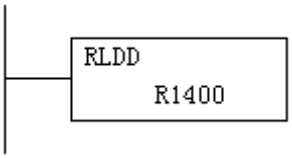
RLDD 指令将存放在两个连续寄存器中的实数读入累加器。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

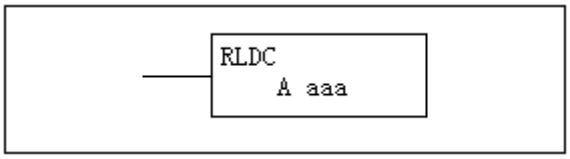
受影响的标志线圈	描述
SP70	读入累加器的数据为负时 ON
SP76	读入累加器的数据为 0 时 ON

使用 RLDD 指令可将一个实数存入 R1400 和 R1401 中，如右图所示，然后可将其进行实数运算，或将其转化成一个二进制数。



6.11 实数读入指令（RLDC）

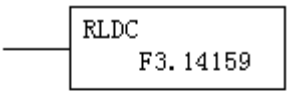
RLDC 指令将一个浮点常数读入累加器。



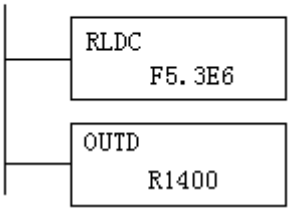
操作数类型		D4-454 范围
	A	aaa
浮点常数	F	-3.402823E+038～+3.402823E+038

受影响的标志线圈	描述
SP70	读入累加器的数据为负时 ON
SP76	读入累加器的数据为 0 时 ON

KPP 允许直接输入实数，如右图所示，以“F”开头，后面的数就是实数，例如右图中的“π”。要想输入负数，在“F”后加上“－”号即可。



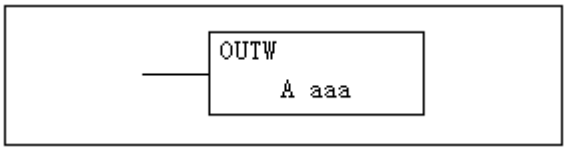
对于特别大和特别小的数，可以使用指数计数法，如右图中的 5.3E6 就是 5300000,OUTD 指令将其存放在 R1400 和 R1401 中。



这些数据都是 IEEE 32 位浮点数格式，因此不管这些数很大或是很小，都占用两个 R 寄存器。如果用十六进制、二进制或是 BCD 格式查看储存的实数，显示的数据会很难理解。跟其它类型的数据一样，必须清楚实数的存储位置，以便使用正确的指令将其读出。

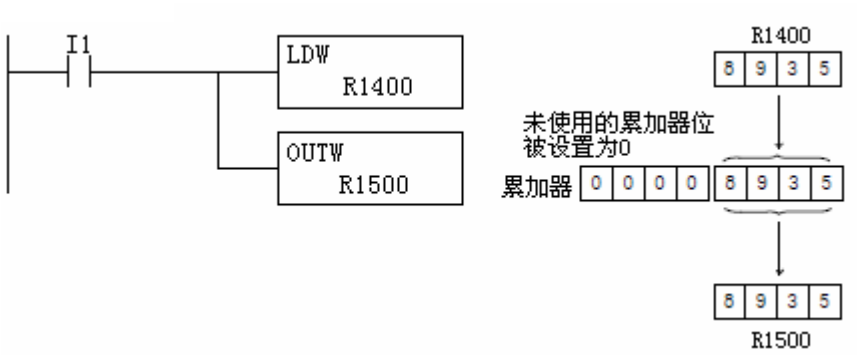
6.12 16 位写入指令（OUTW）

OUTW 指令将累加器的低 16 位写入到 R 寄存器中。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器的低 16 位，OUTW 指令将累加器的低 16 位写入 R1500 中。

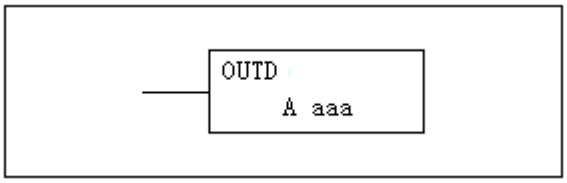


语句视图

```
LD  I1
LDW R1400
OUTW R1500
```

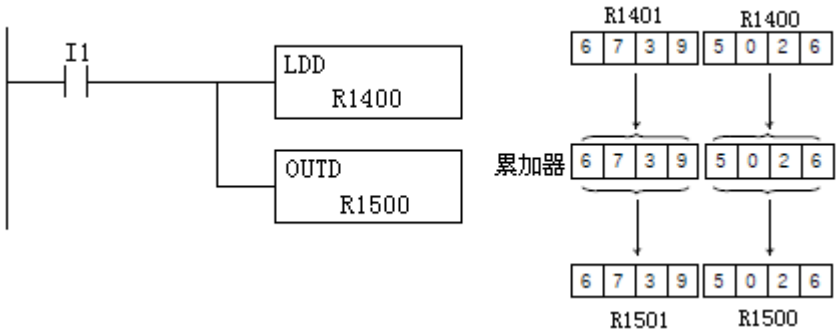
6.13 32 位写入指令（OUTD）

OUTD 指令是一个 32 位指令，用于将累加器的数据写入到两个连续的 R 寄存器中，第一个 R 寄存器的地址为 A aaa。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的 32 位数据读入累加器，OUTD 指令将累加器的 32 位数据写入到 R1500 和 R1501 中。

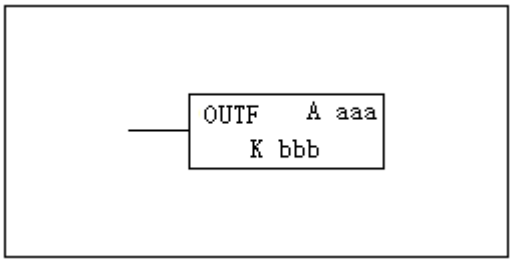


语句视图

```
LD  I1
LDD R1400
OUTD R1500
```

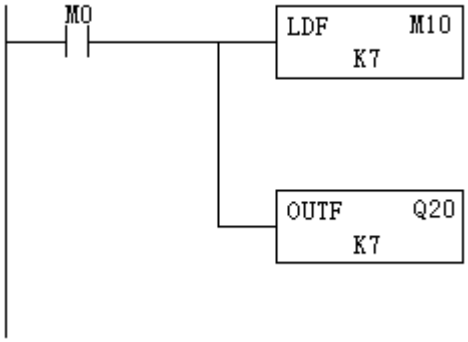
6.14 任意位写入指令（OUTF）

OUTF 指令将累加器的 1-32 位写入到指定的地址中。指令需要目标地址的起始地址和输入的位数。



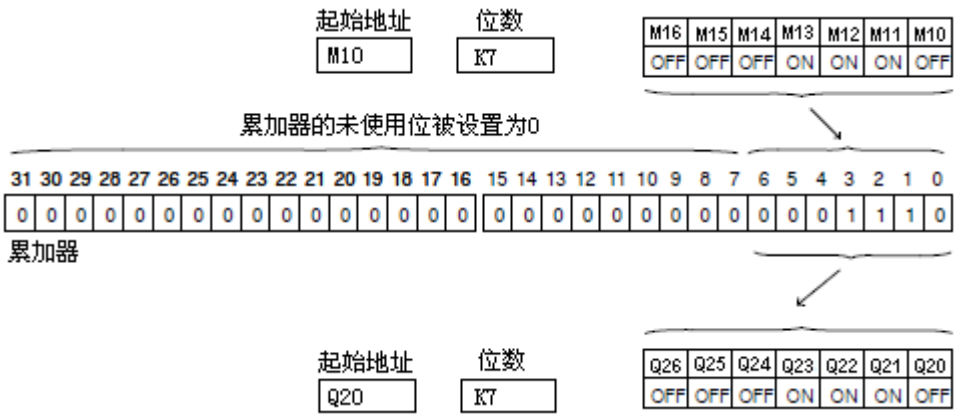
操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

下面的例子中，当 M0 为 ON 时，LDF 指令将 M10-M16（7 位）读入累加器中，OUTF 指令将累加器的低 7 位写入到 Q20-Q26 中。



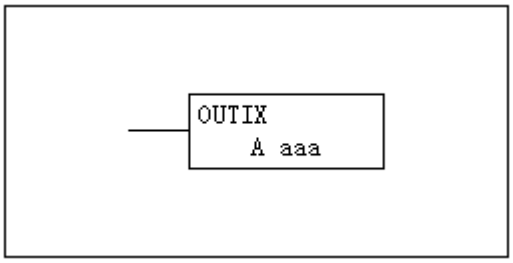
语句视图

```
LD M0
LDF M10 K7
OUTF Q20 K7
```



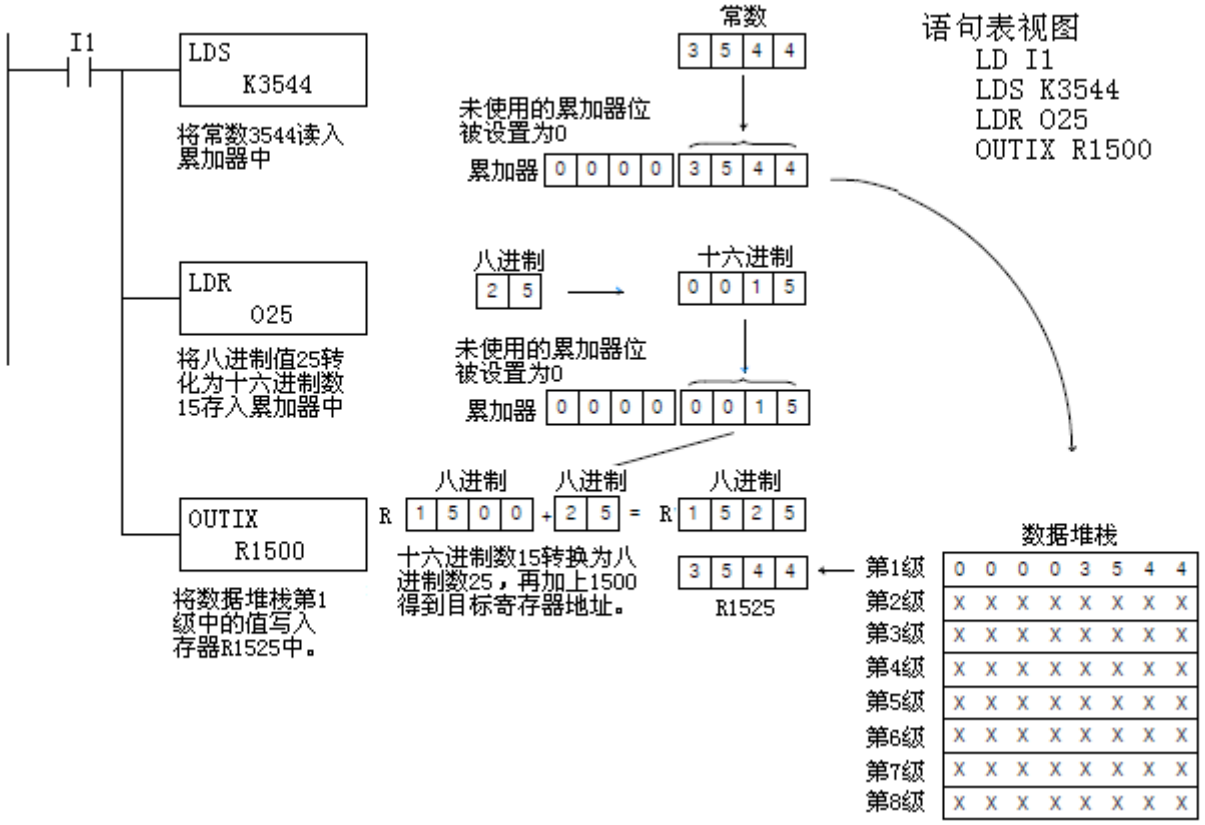
6.15 索引 16 位写入指令（OUTIX）

OUTIX 指令是一个 16 位指令，它将数据堆栈第 1 级中的数据写入到目标地址中，目标地址由指令中指定的地址加上累加器中的偏移量得到。累加器的高 16 位被设置为 0。



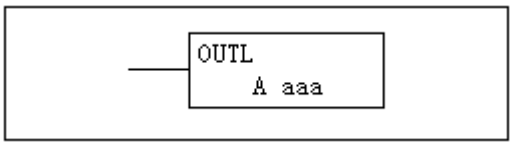
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

下面的例子中，当 I1 为 ON 时，LDS 指令将常数 3544 读入累加器中。LDR 指令将八进制数 25 的十六进制数 15 读入累加器，常数 3544 被压入数据堆栈的第 1 级中。R 寄存器地址 1500 加上累加器中的数 25（八进制）得到目标寄存器地址值 1525，将数据堆栈第 1 级中的数据 3544 读入寄存器 R1525 中。



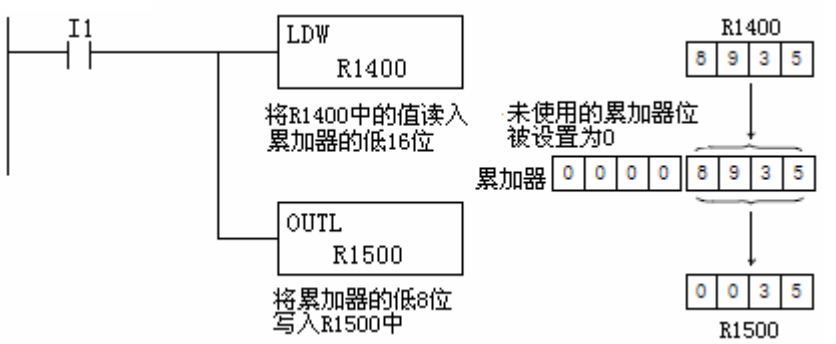
6.16 低 8 位写入指令（OUTL）

OUTL 指令将累加器的低 8 位写入到指定寄存器的低 8 位中。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）

下面的例子中，当 I1 为 ON 时，R1400 中的数据被读入累加器的低 16 位中。累加器的低 8 位被写入 R1500 中。

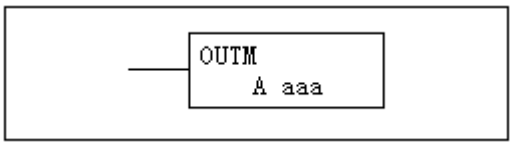


语句视图

```
LD I1
LDW R1400
OUTL R1500
```

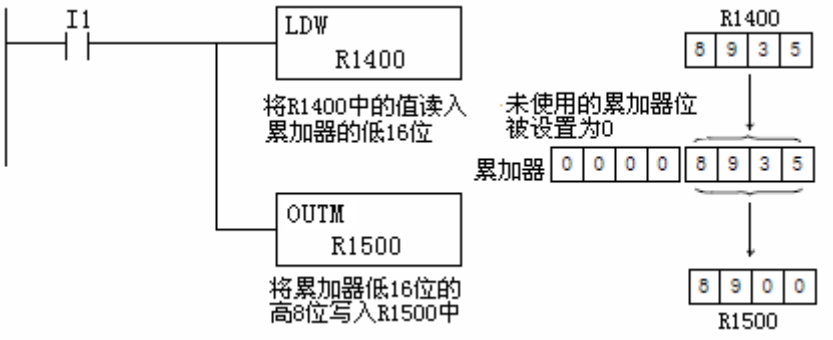
6.17 高 8 位写入指令（OUTM）

OUTM 指令将累加器的低 16 位的高 8 位写入到指定寄存器的高 8 位中。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）

下面的例子中，当 I1 为 ON 时，R1400 中的数据被读入累加器的低 16 位中。累加器低 16 位的高 8 位被写入 R1500 中。

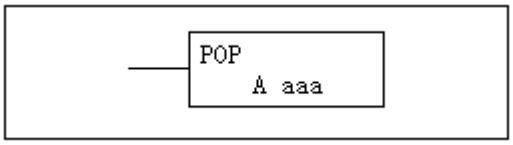


语句视图

```
LD I1
LDW R1400
OUTM R1500
```

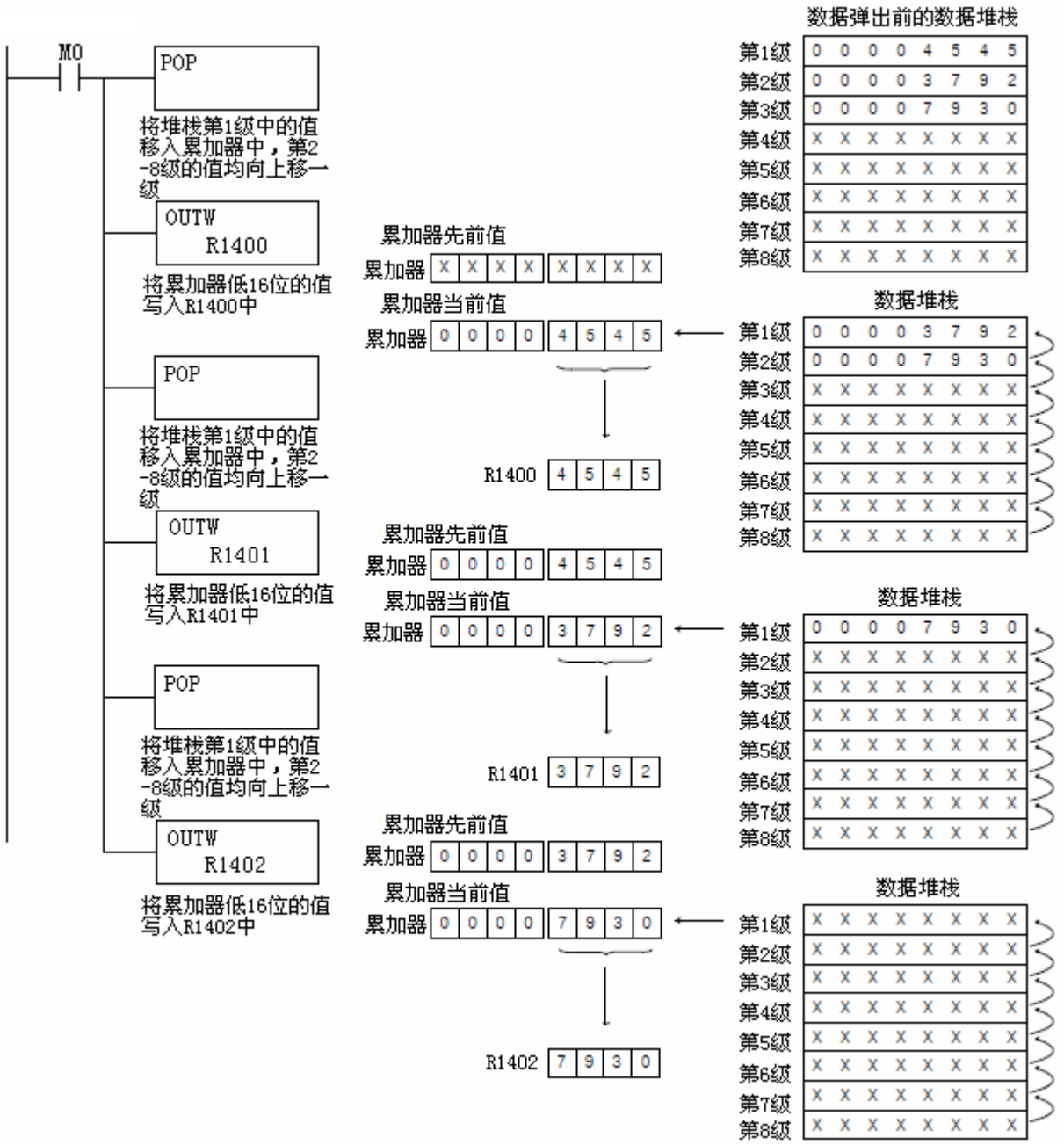
6.18 数据堆栈弹出指令（POP）

POP 指令将数据堆栈第 1 级中的数据移入累加器中，并将堆栈 2-8 级中的数据都向上移一级。



受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON

下面的例子中，当 M0 为 ON 时，第一个 POP 指令将堆栈第 1 级中的数据 4545 移入累加器中，OUTW 指令将累加器的数据写入 R1400 中；第二个 POP 指令将堆栈第 1 级中的数据 3792 移入累加器，OUTW 指令将其写入 R1401 中；第三个 POP 指令将堆栈第 1 级中的数据 7930 移入累加器中，OUTW 指令将其写入 R1402 中。如果堆栈中的数据多于 16 位（4 字），请使用 OUTD 指令，每个数据占用两个连续的寄存器。



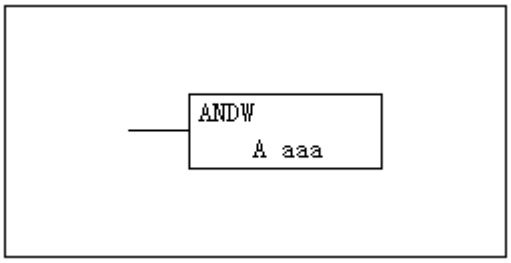
语句视图

```
LD M0
POP
OUTW R1400
POP
OUTW R1401
POP
OUTW R1402
```

第 7 章 累加器逻辑指令

7.1 16 位逻辑与指令（ANDW）

ANDW 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据逻辑与，结果存放在累加器中。如果累加器中的结果是 0，则相应标志线圈 ON。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

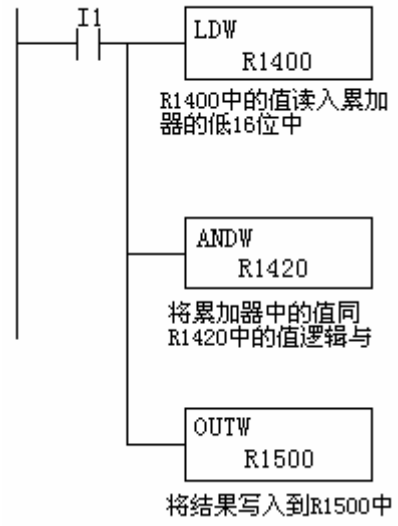
受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON

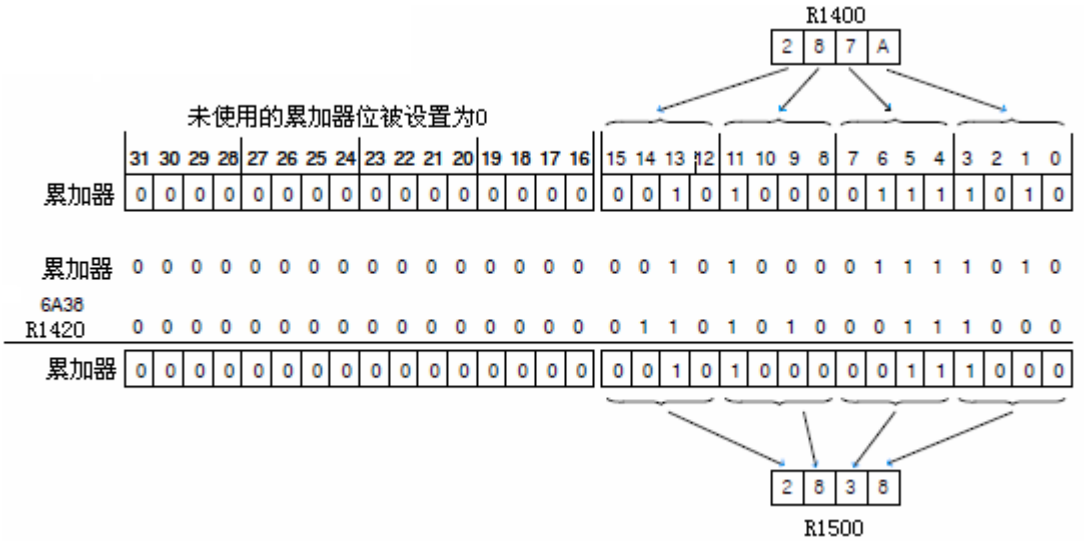
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，ANDW 指令将累加器中的数据同 R1420 中的数据逻辑与，OUTW 指令将结果写入到 R1500 中。

语句视图

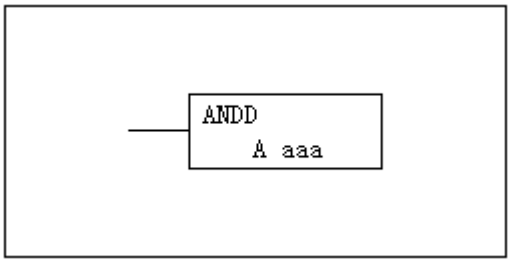
```
LD  I1
LDW  R1400
ANDW  R1420
OUTW  R1500
```





7.2 32 位逻辑与指令（ANDD）

ANDD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据逻辑与，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。

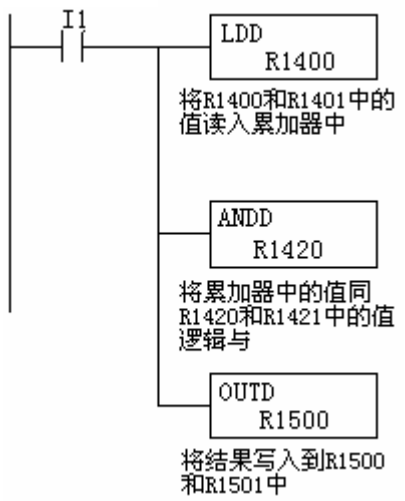


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

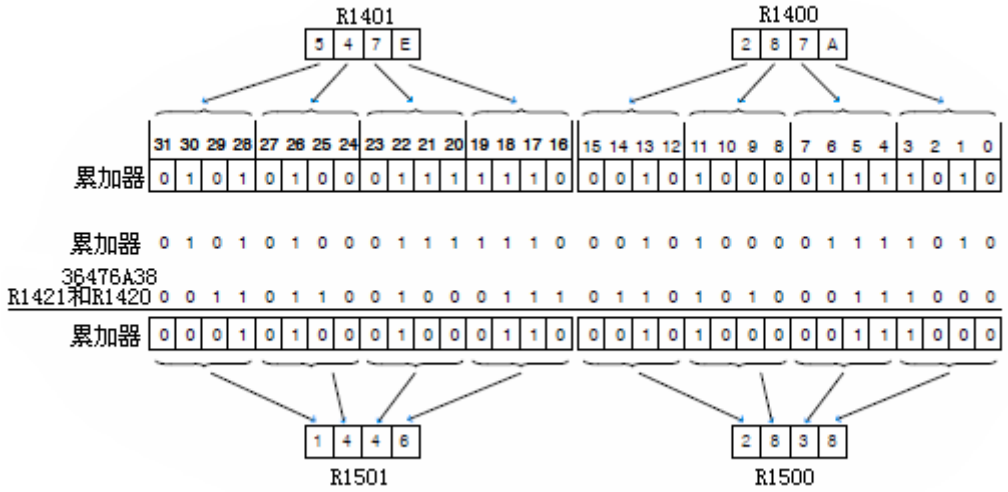
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，ANDD 指令将累加器中的数据同 R1420 和 R1421 中的数据逻辑与，OUTD 指令将结果写入到 R1500 和 R1501 中。



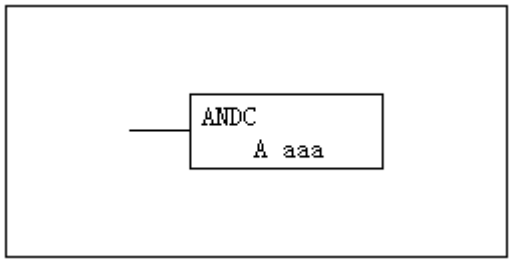
语句视图

```
LD  I1
LDD  R1400
ANDD  R1420
OUTD  R1500
```



7.3 32 位常数逻辑与指令（ANDC）

ANDC 指令是一个 32 位常数指令，将累加器的数据同一个 8 位常数（最大）逻辑与，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。



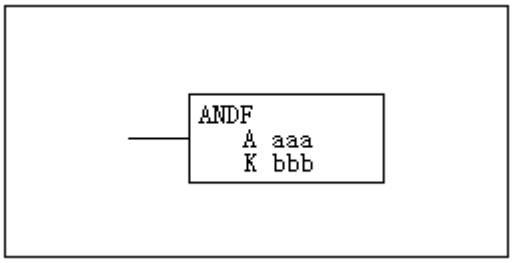
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFFFFFF

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

7.4 任意位长逻辑与指令（ANDF）

ANDF 指令将累加器中的二进制数据同指定位数的二进制数据进行逻辑与，可指定范围为 1-32 位。指令需要指定起始地址（A aaa）和位数（K bbb）。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。

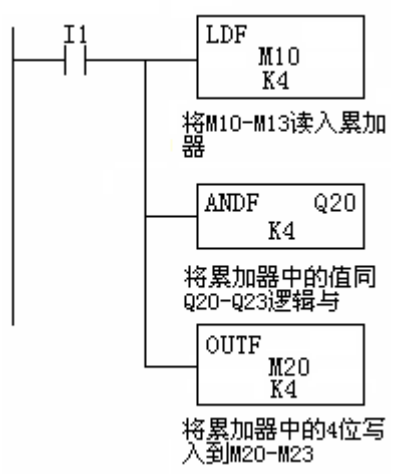


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

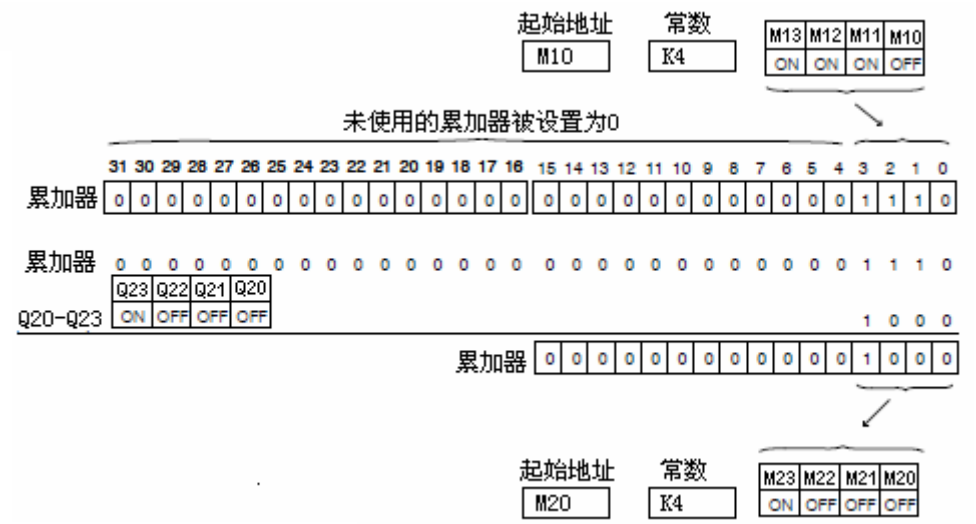
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，ANDF 指令将累加器中的数据同 Q20-Q23 逻辑与，OUTF 指令将结果写入到 M20-M23。



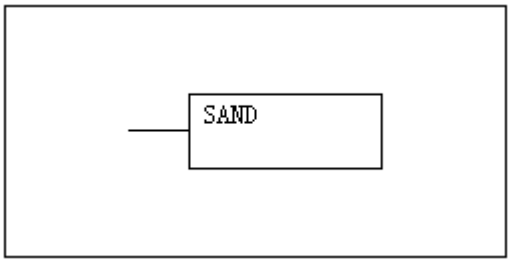
语句视图

```
LD  I1
LDF  M10  K4
ANDF  Q20  K4
OUTF  M20  K4
```



7.5 堆栈逻辑与指令（SAND）

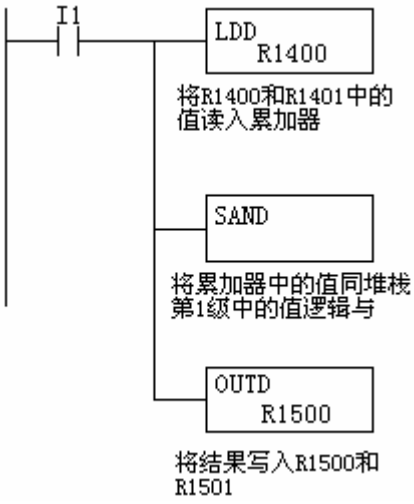
SAND 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据逻辑与，结果存放在累加器中。数据堆栈第 1 级中的数据移出，其它级均向上移一级。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。



受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

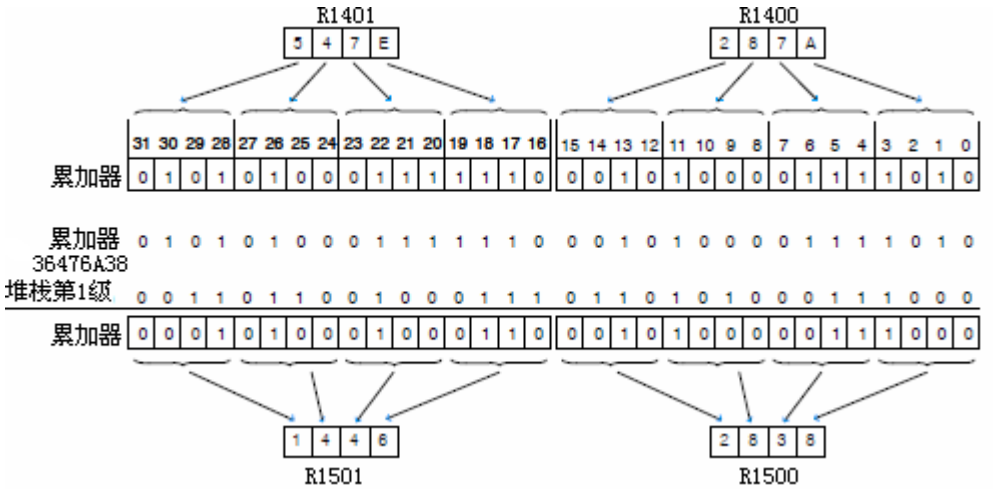
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，累加器中二进制数据同数据堆栈第 1 级中的二进制数据逻辑与，结果存放在累加器中，OUTD 指令将结果写入到 R1500 和 R1501 中。



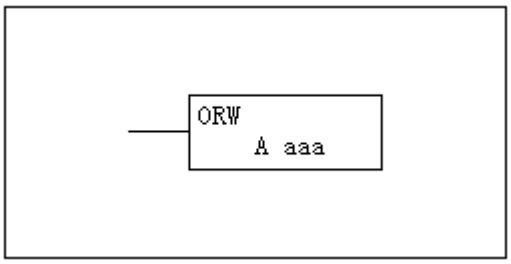
语句视图

```
LD  I1
LDD  R1400
SAND
OUTD  R1500
```



7.6 16 位逻辑或指令（ORW）

ORW 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据逻辑或，结果存放在累加器中。如果累加器中的结果是 0，则相应标志线圈 ON。

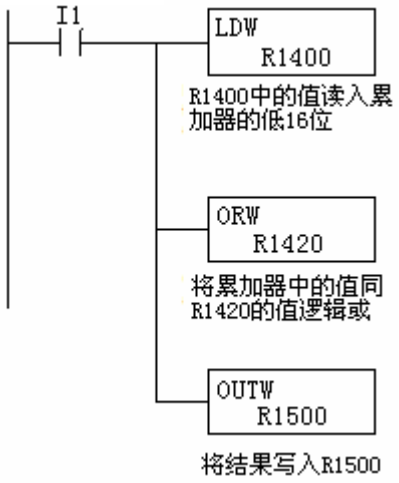


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON

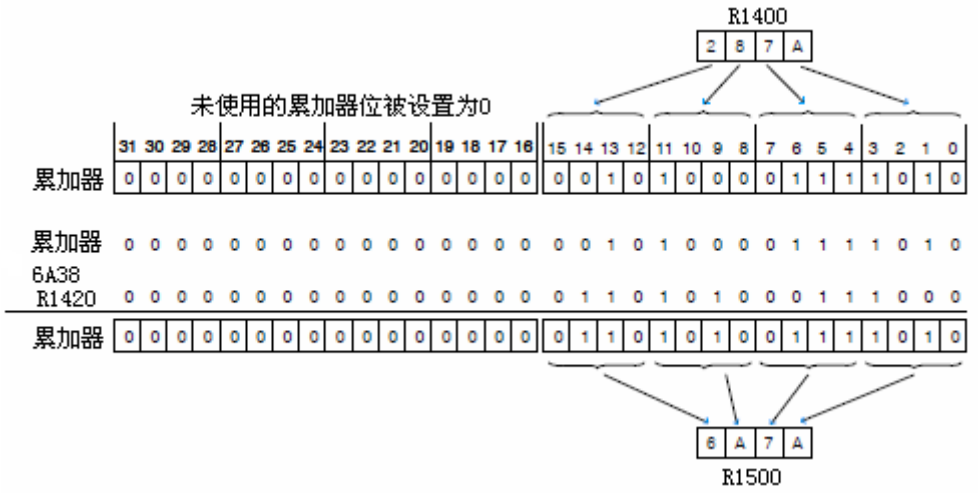
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，ORW 指令将累加器中的数据同 R1420 中的数据逻辑或，OUTW 指令将结果写入到 R1500 中。



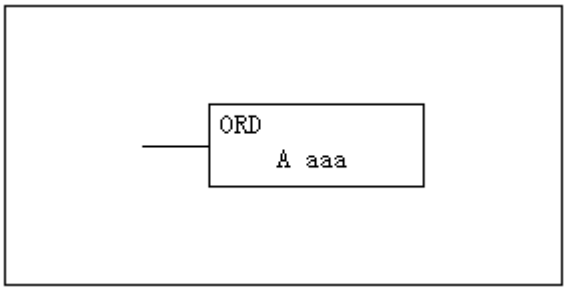
语句视图

```
LD  I1
LDW R1400
ORW R1420
OUTW R1500
```



7.7 32 位逻辑或指令（ORD）

ORD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据逻辑或，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。

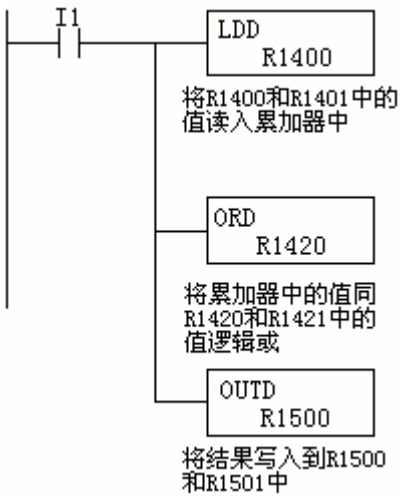


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

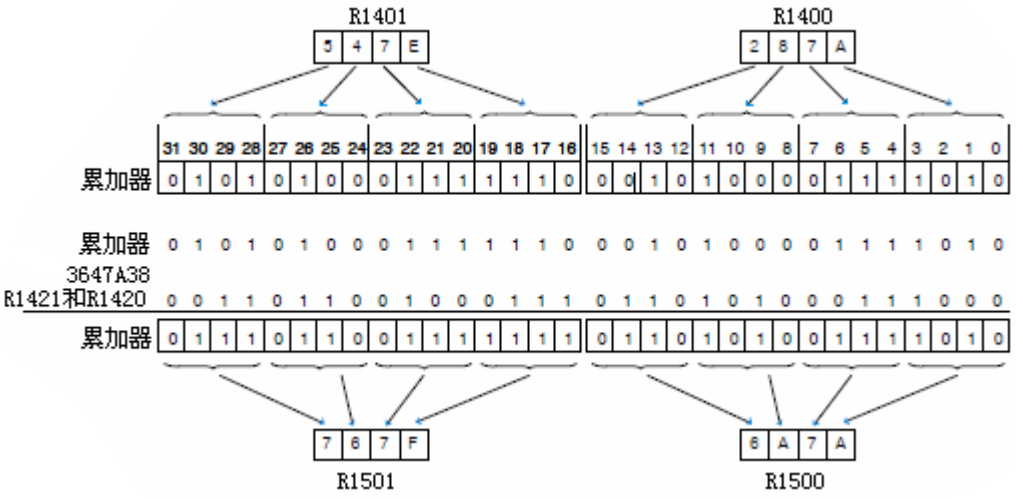
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，ORD 指令将累加器中的数据同 R1420 和 R1421 中的数据逻辑或，OUTD 指令将结果写入到 R1500 和 R1501 中。



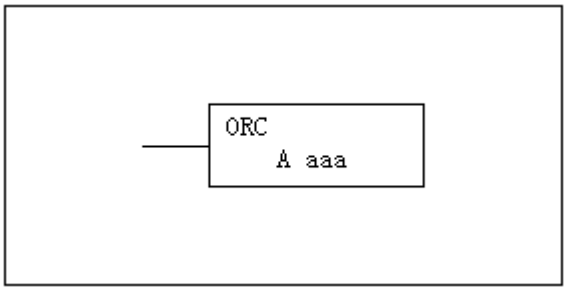
语句视图

```
LD I1
LDD R1400
ORD R1420
OUTD R1500
```



7.8 32 位常数逻辑或指令（ORC）

ORC 指令是一个 32 位常数指令，将累加器的数据同一个 8 位常数（最大）逻辑或，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。



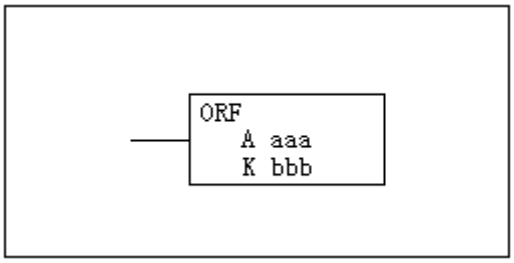
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFFFFFF

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

7.9 任意位长逻辑与指令（ORF）

ORF 指令将累加器中的二进制数据同指定位数的二进制数据进行逻辑或，可指定范围为 1-32 位。指令需要指定起始地址(A aaa)和位数(K bbb)。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。

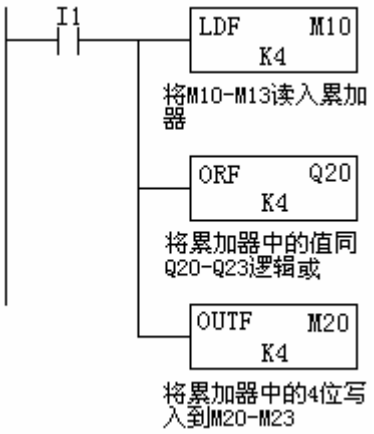


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

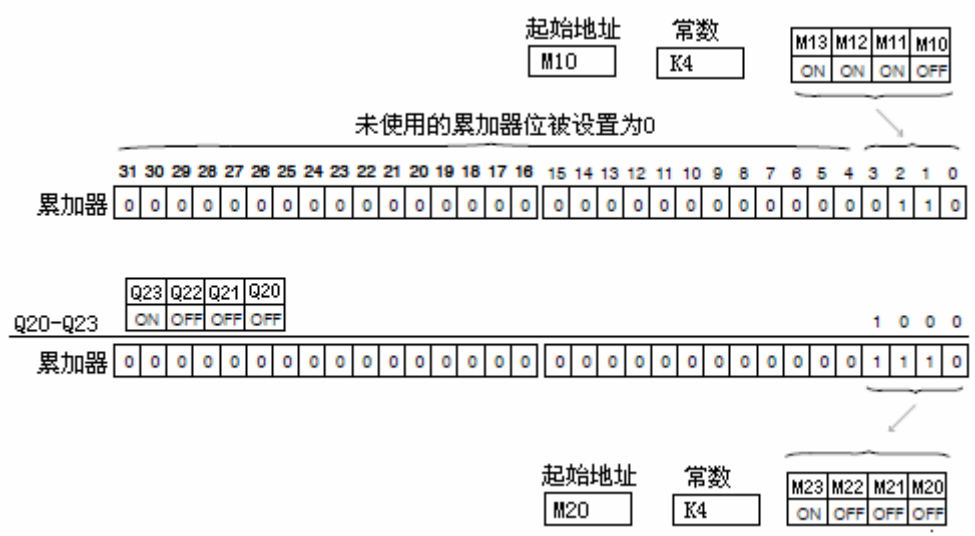
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，ORF 指令将累加器中的数据同 Q20-Q23 逻辑或，OUTF 指令将结果写入 M20-M23。



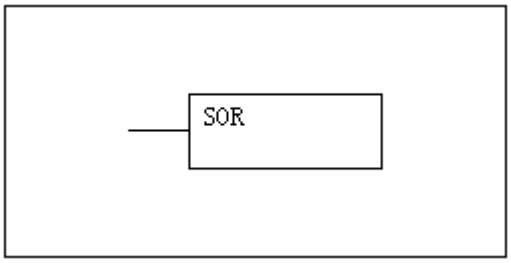
语句视图

```
LD I1
LDF M10 K4
ORF Q20 K4
OUTF M20 K4
```



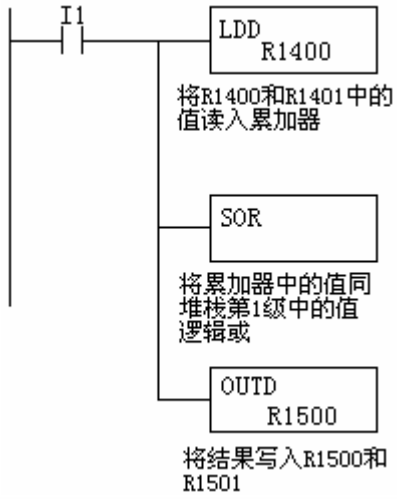
7.10 堆栈逻辑或指令（SOR）

SOR 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据逻辑或，结果存放在累加器中。数据堆栈第 1 级中的数据移出，其它级均向上移一级。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。



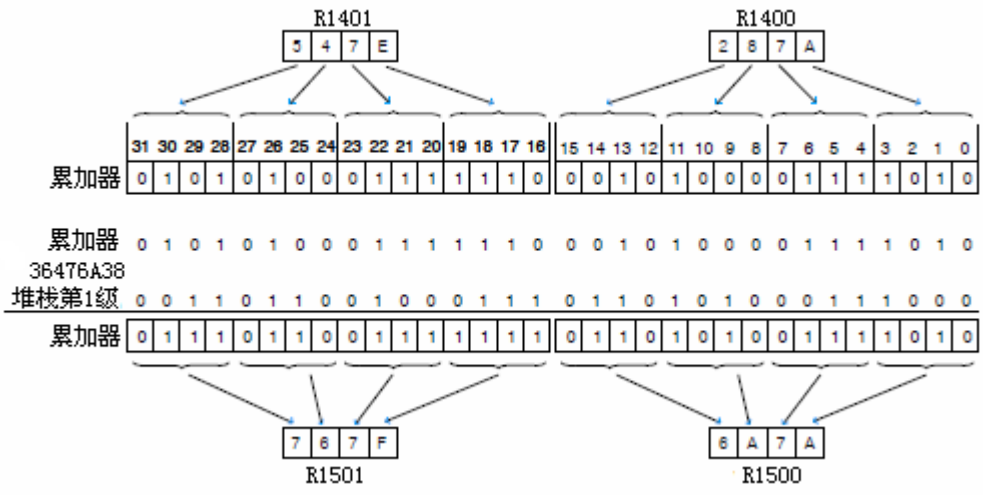
受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

下面的例子中，当 I1 为 ON 时，累加器中的数据同数据堆栈第 1 级中的数据逻辑或，结果存放在累加器中，OUTD 指令将其写入到 R1500 和 R1501 中。



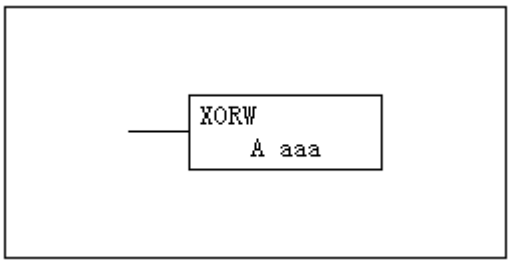
语句视图

```
LD  I1
LDD R1400
SOR
OUTD R1500
```



7.11 16 位逻辑异或指令（XORW）

XORW 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据逻辑异或，结果存放在累加器中。如果累加器中的结果是 0，则相应标志线圈 ON。

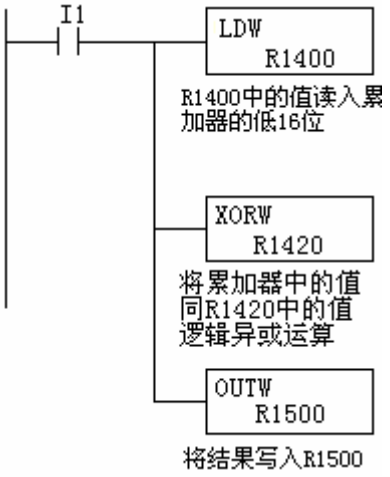


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON

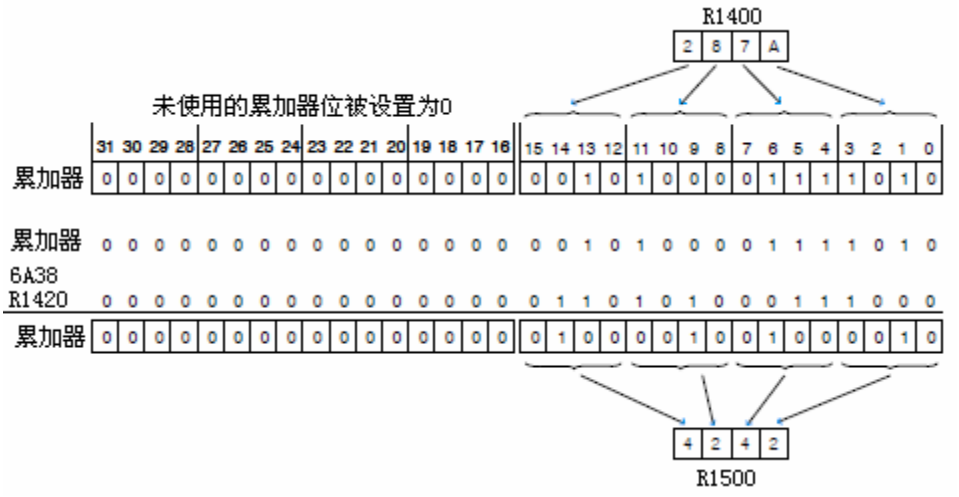
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，XORW 指令将累加器中的数据同 R1420 中的数据进行逻辑异或运算，OUTW 指令将结果写入到 R1500 中。



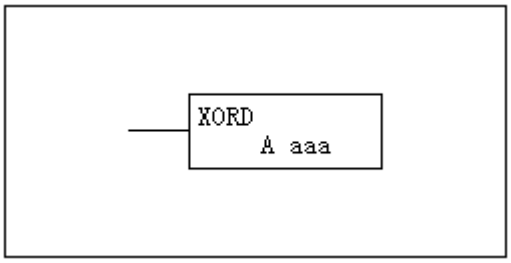
语句视图

```
LD I1
LDW R1400
XORW R1420
OUTW R1500
```



7.12 32 位逻辑异或指令（XORD）

XORD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据进行逻辑异或运算，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。

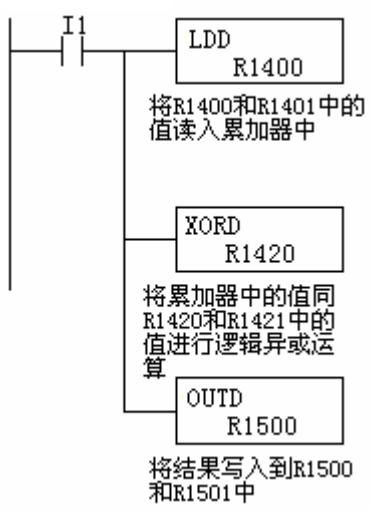


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

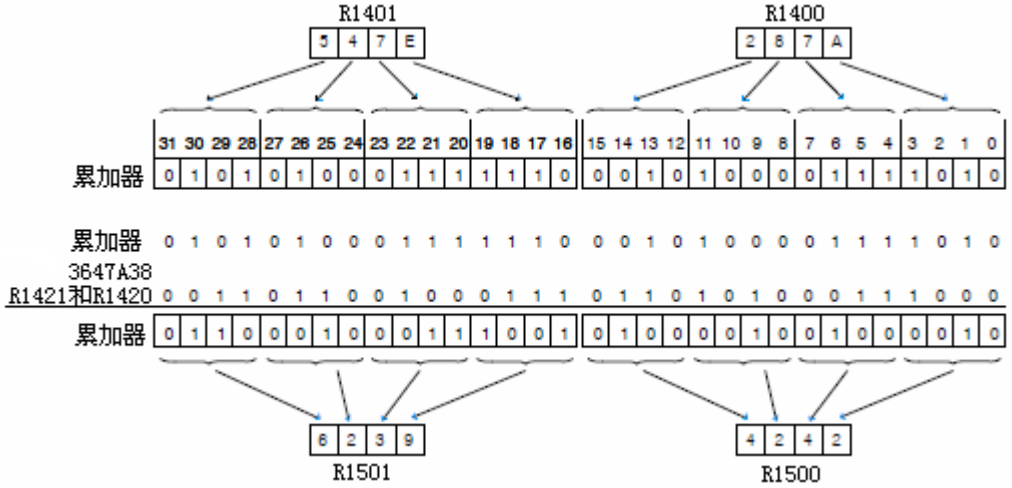
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，XORD 指令将累加器中的数据同 R1420 和 R1421 中的数据进行逻辑异或运算，OUTD 指令将结果写入到 R1500 和 R1501 中。



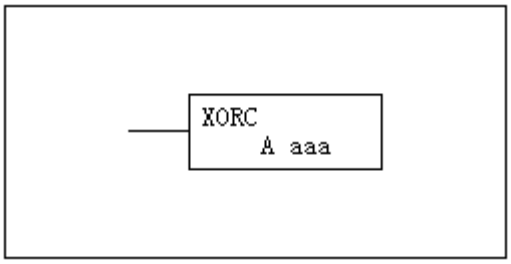
语句视图

```
LD  I1
LDD R1400
XORD R1420
OUTD R1500
```



7.13 32 位常数逻辑异或指令（XORC）

XORC 指令是一个 32 位常数指令，将累加器的数据同一个 8 位常数（最大）进行逻辑异或运算，结果存放在累加器中。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。



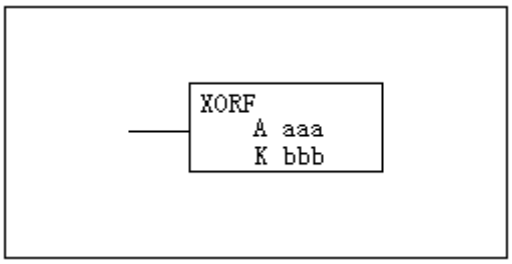
操作数类型		D4-454 范围
	A	aaa
常数	K	0-FFFFFFFF

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

7.14 任意位长逻辑异或指令（XORF）

XORF 指令将累加器中的二进制数同指定位数（1-32 位）的二进制数进行逻辑异或运算。指令需要指定起始地址(A aaa)和位数(K bbb)。如果累加器中的结果是 0 或负数(最高位为 1)，则相应标志线圈 ON。

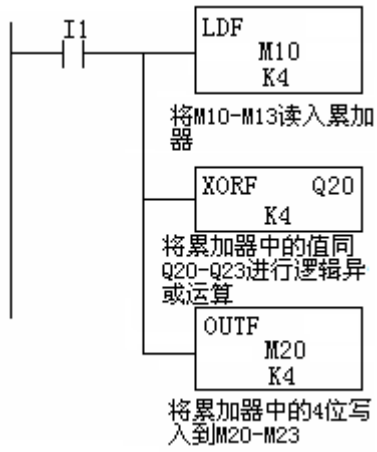


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

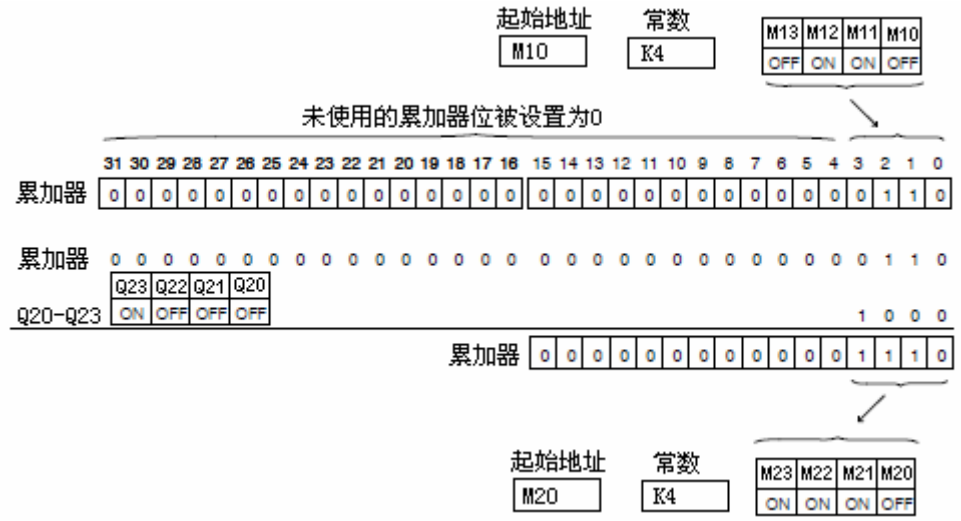
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，XORF 指令将累加器中的数据同 Q20-Q23 进行逻辑异或运算，OUTF 指令将结果写入 M20-M23。



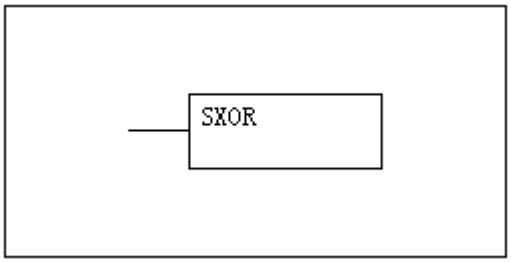
语句视图

```
LD I1
LDF M10 K4
XORF Q20 K4
OUTF M20 K4
```



7.15 堆栈逻辑异或指令（SXOR）

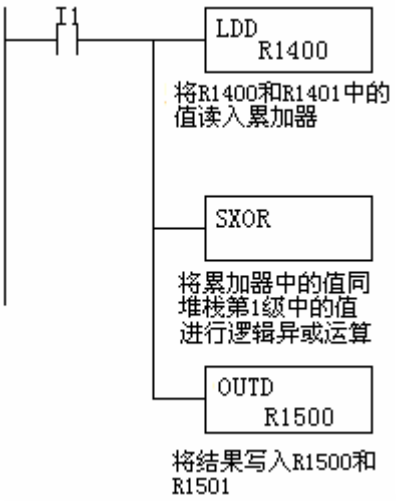
SXOR 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据进行逻辑异或运算，结果存放在累加器中。数据堆栈第 1 级中的数据移出，其它级均向上移一级。如果累加器中的结果是 0 或负数（最高位为 1），则相应标志线圈 ON。



受影响的标志线圈	描述
SP63	指令导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

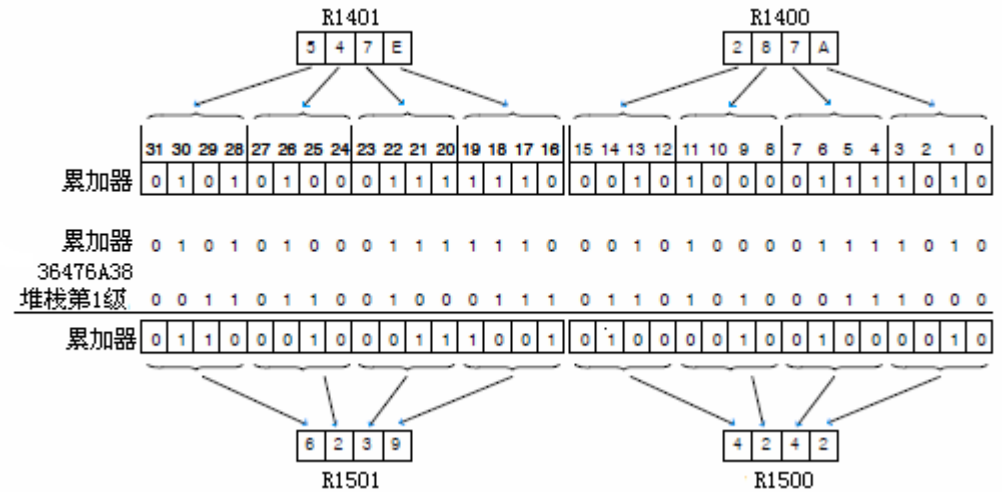
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，累加器中的二进制数同数据堆栈第 1 级中的二进制数进行逻辑异或运算，结果存放在累加器中，OUTD 指令将其写入到 R1500 和 R1501 中。



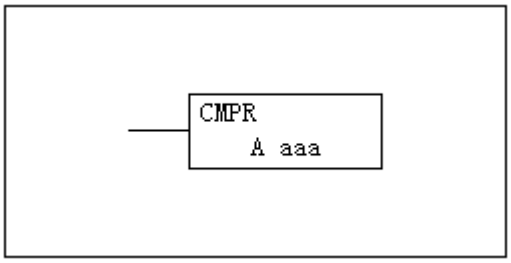
语句视图

```
LD  I1
LDD R1400
SXOR
OUTD R1500
```



7.16 16 位比较指令（CMPR）

CMPR 指令是一个 16 位指令，将累加器低 16 位的数据同指定寄存器（A aaa）中的数据进行比较运算，相应的特殊线圈变为 ON 来显示比较的结果。进行比较的数据格式可以是二进制或是 BCD，两个数据的格式要相同。

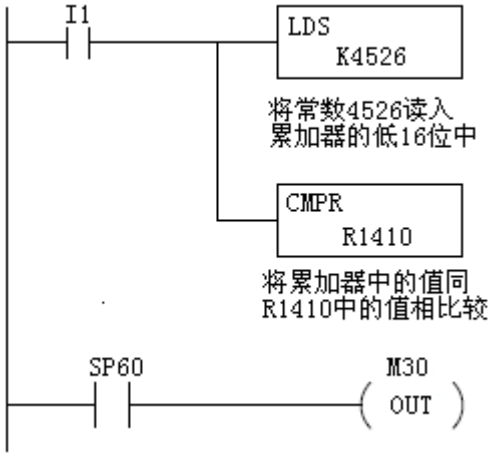


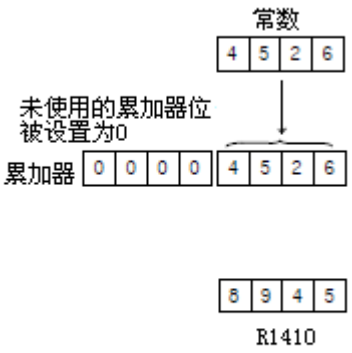
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDS 指令将常数 4526 读入累加器，CMPR 指令将累加器中的数据同 R1410 中的数据进行比较，相应的标志线圈将变为 ON 来显示比较结果。本例中，当累加器中的数据小于 R1410 中的数据时，SP60 就被置为 ON。



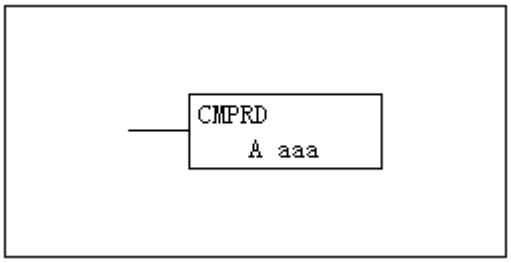


语句视图

```
LD I1
LDS K4526
CMPR R1410
LD SP60
OUT M30
```

7.17 32 位比较指令（CMPRD）

CMPRD 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的数据进行比较运算，相应的特殊线圈变为 ON 来显示比较的结果。进行比较的数据格式可以是二进制或是 BCD，两个数据的格式要相同。

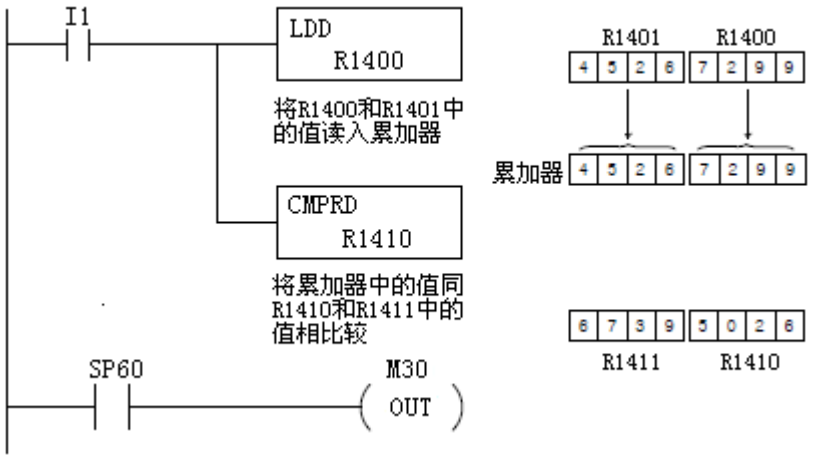


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器，CMPRD 指令将累加器中的数据同 R1410 和 R1411 中的数据进行比较运算，相应的标志线圈将变为 ON 来显示比较结果。本例中，当累加器中的数据小于 R1410 和 R1411 中的数据时，SP60 就被置为 ON。

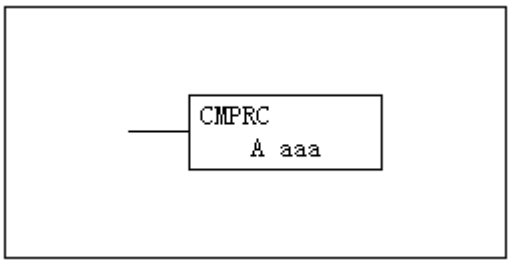


语句视图

```
LD I1
LDD R1400
CMPRD R1410
LD SP60
OUT M30
```

7.18 8 位常数比较指令（CMPRC）

CMPRC 指令是一个 8 位常数指令，将累加器的数据同一个 8 位常数（最大）进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。



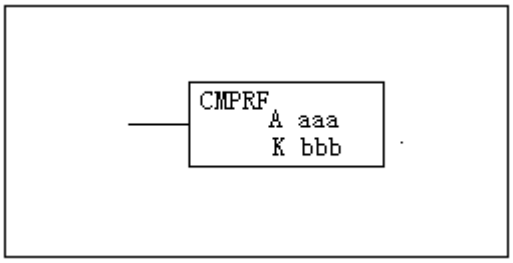
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFFFFFF

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

7.19 任意位长比较指令（CMPRF）

CMPRF 指令将累加器中的数据同指定位数的数据进行比较运算。指令需要指定起始地址（A aaa）和位数（K bbb）。相应的特殊线圈被置为 ON 来显示比较的结果。

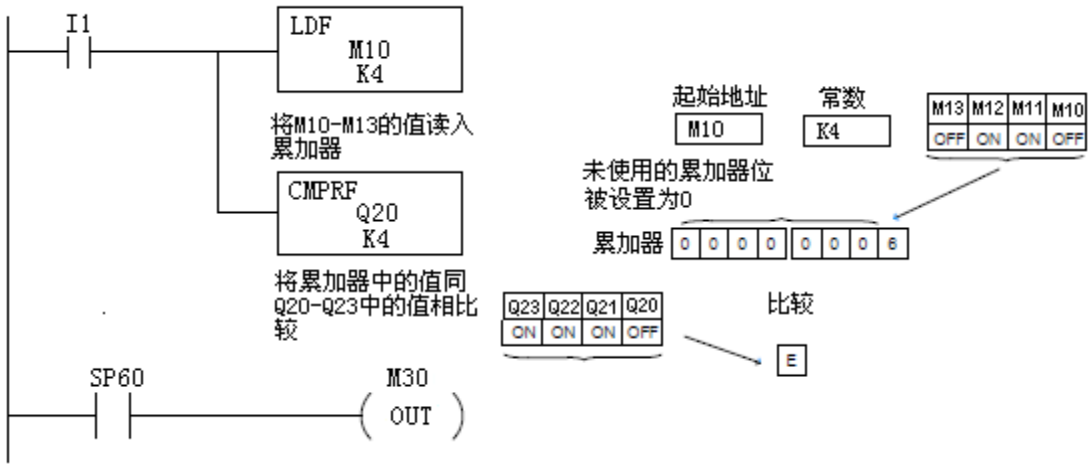


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDF 指令将 M10-M13（4 位）读入累加器，CMPRF 指令将累加器中的数据同 Q20-Q23（十六进制数 E）进行比较运算，相应的标志线圈将被置为 ON 来显示比较的结果。本例中，当累加器中的数据小于 Q20-Q23 中的数据时，SP60 就变为 ON。

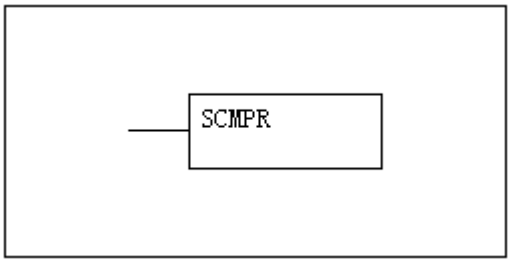


语句视图

```
LD I1
LDF M10 K4
CMPRF Q20 K4
LD SP60
OUT M30
```

7.20 堆栈比较指令（SCMPR）

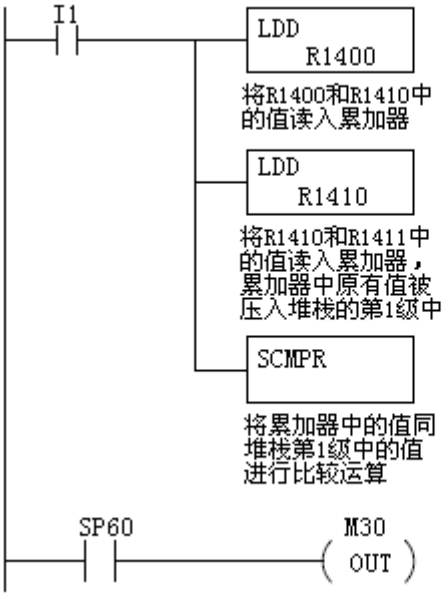
SCMPR 指令是一个 32 位指令，将累加器中的数据同数据堆栈第 1 级中的数据进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。

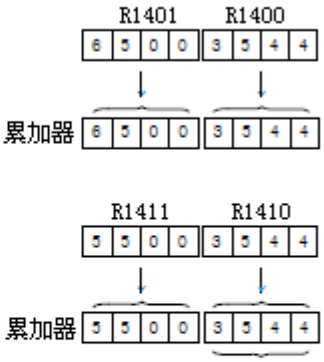


受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器，下一个 LDD 指令将 R1410 和 R1411 中的数据装入累加器，累加器中原有的数据被压入数据堆栈的第 1 级中。SCMPR 指令将累加器中数据同数据堆栈第 1 级中的数据进行比较运算，当累加器中的数据小于堆栈第 1 级中的数据时，SP60 被置为 ON，M30 被接通。



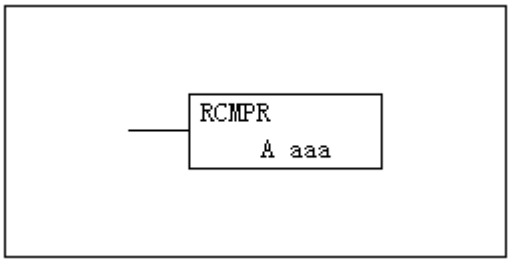


语句视图

```
LD I1
LDD R1400
LDD R1410
SCMPR
LD SP60
OUT M30
```

7.21 实数比较指令（RCMPR）

RCMPR 指令是一个 32 位指令，将累加器的数据同指定的两个连续寄存器中的实数进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。进行比较的数据长度都是 32 位。



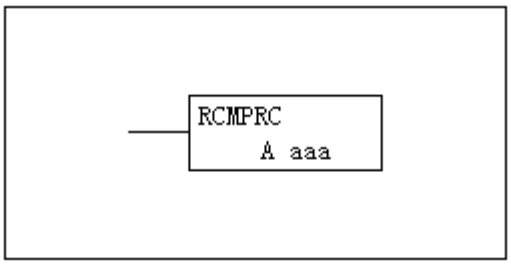
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON
SP71	指定了不存在间接寄存器的区域时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

7.22 实数常数比较指令（RCMPRC）

RCMPRC 指令是一个实数常数指令，将累加器的数据同一个实数常数进行比较运算，相应的特殊线圈被置为 ON 来显示比较的结果。

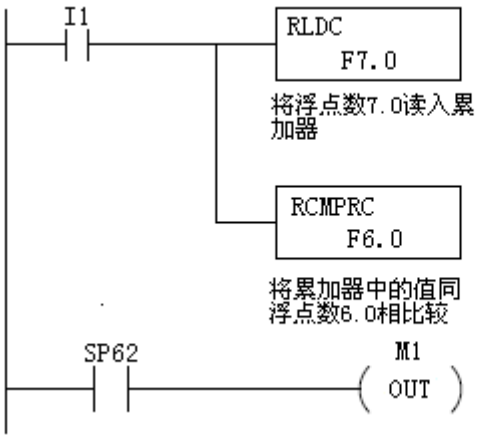


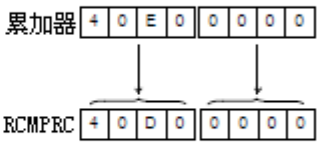
操作数类型	D4-454 范围
A	aaa
常数 K	-3.402823E+038～+3.402823E+038

受影响的标志线圈	描述
SP60	当累加器中数据小于指令中数据时 ON
SP61	当累加器中数据等于指令中数据时 ON
SP62	当累加器中数据大于指令中数据时 ON
SP71	指定了不存在间接寄存器的区域时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，RLDC 指令将实数 7.0 读入累加器，RCMPRC 指令将累加器中的数据同实数 6.0 进行比较运算，因为 7＞6，因此特殊线圈 SP62 被置为 ON。





语句视图

```
LD  I1
RLDC  F7.0
RCMPRC  F6.0
LD  SP62
OUT  M1
```

第 8 章 数据处理指令的解说

8.1 数据的形态

（1）PLC 使用的数据

PLC 可以处理各种不同用途的数据。作为数据处理的对象，是多位数集合起来构成的数据，应该是什么形式都可以的。但当进行算术运算时，因其用途上的特点，必须是数值数据。

数据种类

（2）10 进制数的形态

10 进制数是用得最多的数值，输入 PLC、PLC 输出的数据、常数数据等，大多用 10 进制形式。但是 PLC 的 I/O 模块及内部处理所表示的数据，必须是二进制的 ON/OFF（“1”,“0”）状态的集合。因此，在 PLC 内部表示 10 进制数的是 BCD 码（即二进制码表示的十进制数），即 1 位 10 进制数用 4 位二进制数来表示。

10 进制数

第 8 位	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位
9～0	9～0	9～0	9～0	9～0	9～0	9～0	9～0

BCD 码（2 进制码表示的 10 进制数）

8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1	8, 4, 2, 1
$\times 10^7$	$\times 10^6$	$\times 10^5$	$\times 10^4$	$\times 10^3$	$\times 10^2$	$\times 10$	$\times 1$

BCD 每位与 10 进制数的关系

BCD 运算的场合，不能使用右表以外的代码
例：BCD 码不能将十进制数 10 表示为 1010。
十进制数的 10 以 BCD 码表示为下述数码。

0 0 0 1　0 0 0 0

1　　0

10 进数		0	1	2	3	4	5	6	7	8	9
BCD 码	8	0	0	0	0	0	0	0	0	1	1
	4	0	0	0	0	1	1	1	1	0	0
	2	0	0	1	1	0	0	1	1	0	0
	1	0	1	0	1	0	1	0	1	0	1

（3）16 进制数的形态

16 进制数是 4 位二进制位为单位数字化的数值。（1 位数值由 0～F 来表示）。

16 进制数（下位）

第 4 位 F～0	第 3 位 F～0	第 2 位 F～0	第 1 位 F～0

二进制码（下位）

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
（8 ,4 ，2 ， 1） $\times 2^{12}$				（8 ,4 ，2 ， 1） $\times 2^{8}$				（8 ,4 ，2 ， 1） $\times 2^{4}$				（8 ,4 ，2 ， 1） $\times 2^{0}$			

16 进制数（上位）

第 8 位 F～0	第 7 位 F～0	第 6 位 F～0	第 5 位 F～0

二进制码（上位）

2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
（8 ,4 ，2 ， 1） $\times 2^{28}$				（8 ,4 ，2 ， 1） $\times 2^{24}$				（8 ,4 ，2 ， 1） $\times 2^{20}$				（8 ,4 ，2 ， 1） $\times 2^{16}$			

1 位 16 进制数的二进制码 (10) (11) (12) (13) (14) (15)

16 进制数		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
二进制码	8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	4	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

10 进制数与 16 进制数

10 进制数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		32
16 进制数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10		20

（4）8 进制数的形态

8 进制数是将二进制码以 3 位为单位数字化的数值，用于功能存储器编号等。

※I/O 内部继电器、级、定时器、计数器等均用 8 进制数表示。

8 进制数

	第 5 位 7～0	第 4 位 7～0	第 3 位 7～0	第 2 位 7～0	第 1 位 7～0

二进制码（下位）

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
(4 ，2 ， 1） $\times 2^{12}$				（ 4 ，2 ， 1） $\times 2^{9}$				（4 ，2 ， 1） $\times 2^{6}$				（4 ，2 ， 1） $\times 2^{3}$				（4 ，2 ， 1） $\times 2^{0}$			

1 位 8 进制数的二进制码

8 进制数		0	1	2	3	4	5	6	7
二进制码	4	0	0	0	0	1	1	1	1
	2	0	0	1	1	0	0	1	1
	1	0	1	0	1	0	1	0	1

10 进制数与 8 进制数

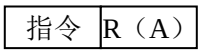
10 进制数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8 进制数	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24

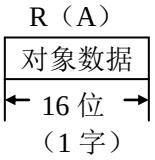
8.2 数据的指定

（1）指定寄存器号 R

指令中的操作数指定存储对象数据的寄存器号，另外，以 R 指定的指令中有以 16 位为对象的指令和以 32 位为对象的指令。

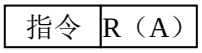
16 位对象指定

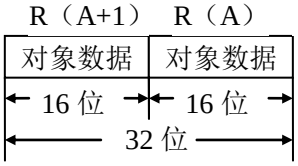




（1 字）

32 位对象指定





（A）：寄存器号

（2 字）

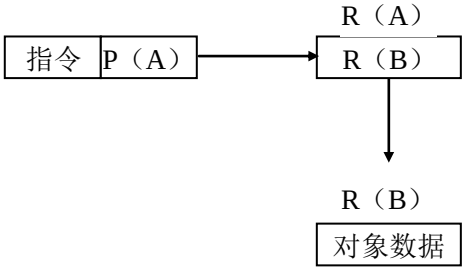
例：将 8 位 10 进制数存储在 2 个寄存器里的场合。

10 进制数		7	3	4	1	5	8	2	4
BCD 码	位数	第 8 位	第 7 位	第 6 位	第 5 位	第 4 位	第 3 位	第 2 位	第 1 位
	加权	8，4，2，1	8，4，2，1	8，4，2，1	8，4，2，1	8，4，2，1	8，4，2，1	8，4，2，1	8，4，2，1
	数据	0 1 1 1	0 0 1 1	0 1 0 0	0 0 0 1	0 1 0 1	1 0 0 0	0 0 10	0 1 0 0
寄存器号		R2001				R2000			

（2）寄存器号的间接指定 P

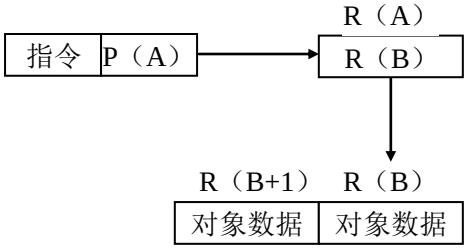
间接指定 P 所指定的 R（A）寄存器中存储的是 R（B）寄存器号，R（B）寄存器中存储所需处理的对象数据。

16 位对象指定



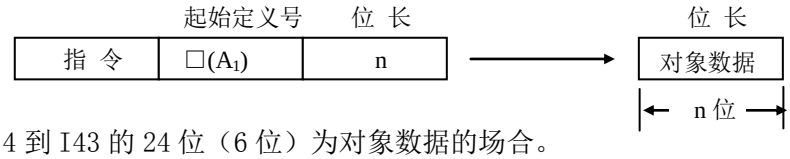
（A）、（B）：寄存器号

32 位对象指定

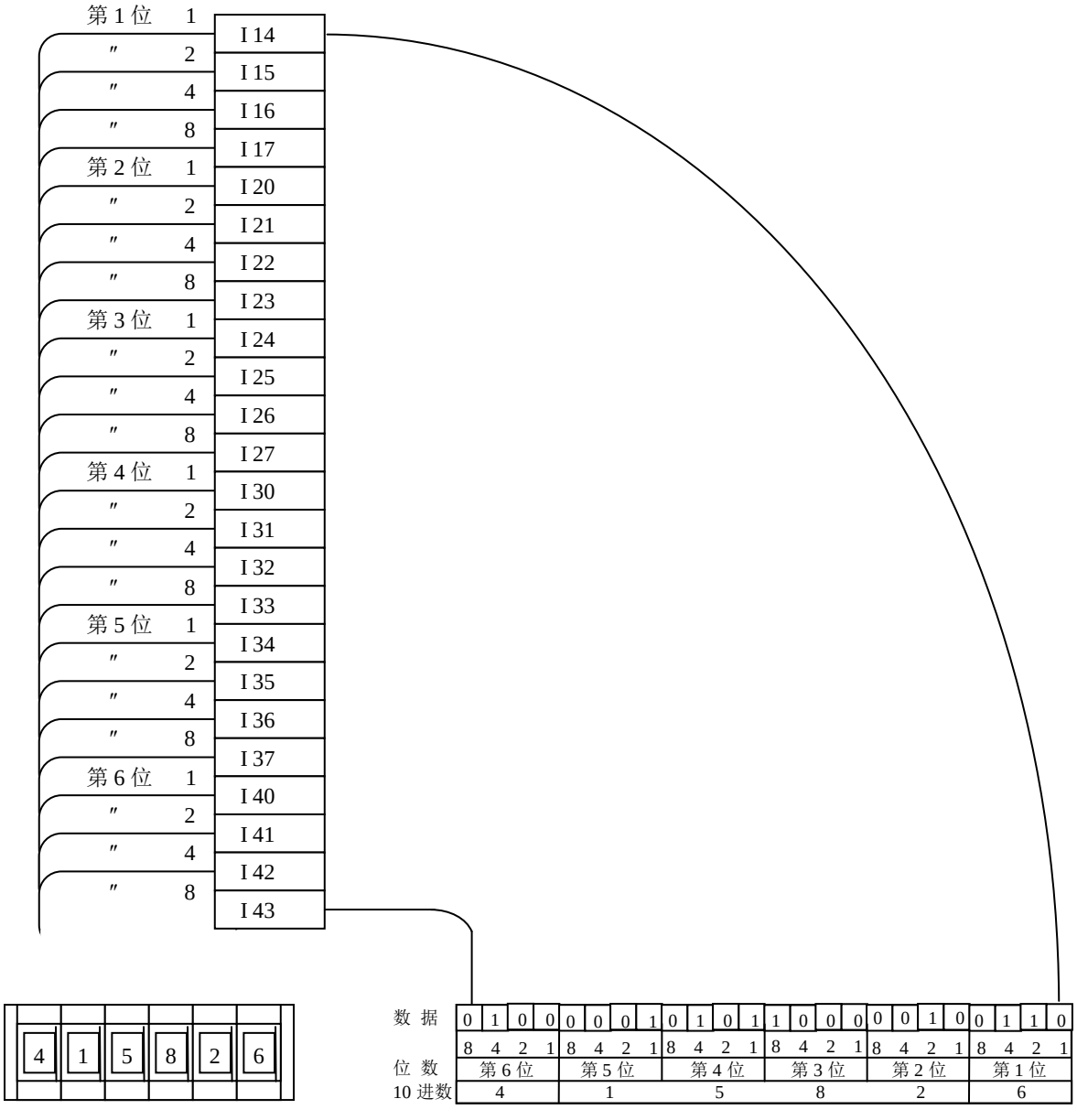


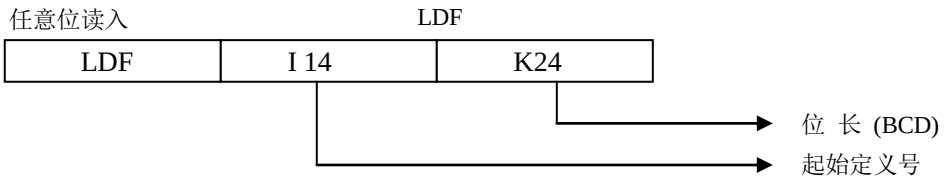
（3）任意位长指定 I，Q，GI，GQ，M，S，T，C，SP

对象数据不为 16 位或 32 位时，或从位号中途开始(即不是从×××0 定义号开始)的数据时，用该方法。



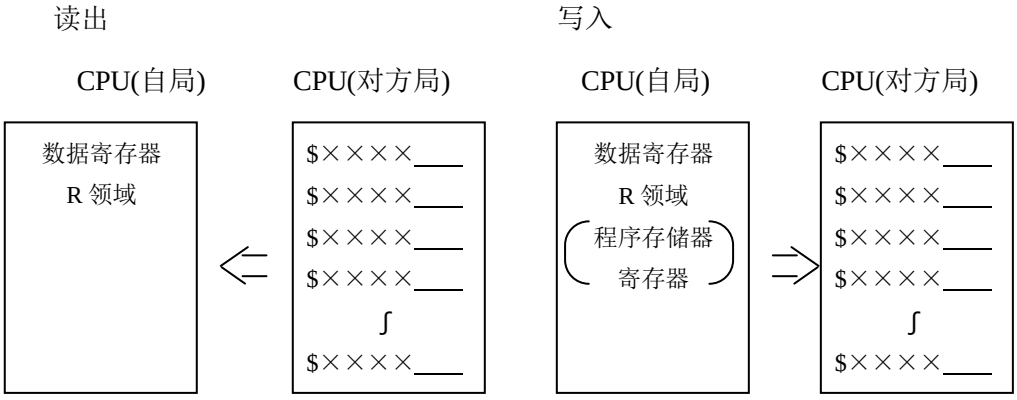
例：以从输入 I14 到 I43 的 24 位（6 位）为对象数据的场合。





（4）程序存储器地址指定$

用于模块对象的特殊指令，通过 PLC 通讯模块从其他 PLC 读出程序或者向其他 PLC 写入程序时使用。

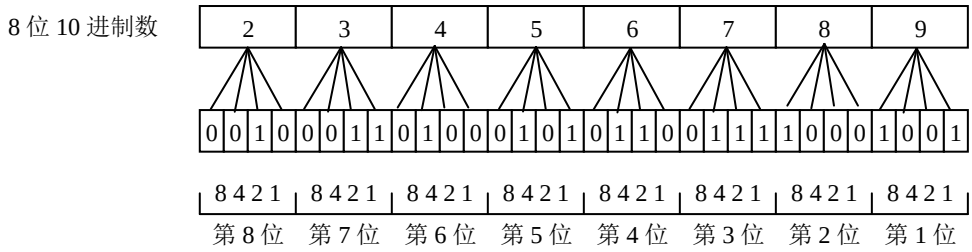


（5）10 进制或 16 进制常数指定 K

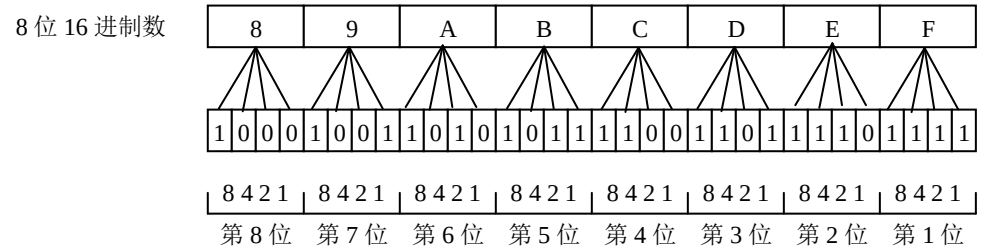
数据处理上所需要的固定数值由程序指令直接给出。

K 用来指定 10 进制常数（BCD 码）或者 16 进制常数（二进制码）。10 进制常数时由 0～9 的键来指定，16 进常数时，由 0～F 键来指令。以 K 指定的数据，将 1 位分解为 4 位二进制来记忆。

例：8 位 10 进制数的表示



例：8 位 16 进制数的表示



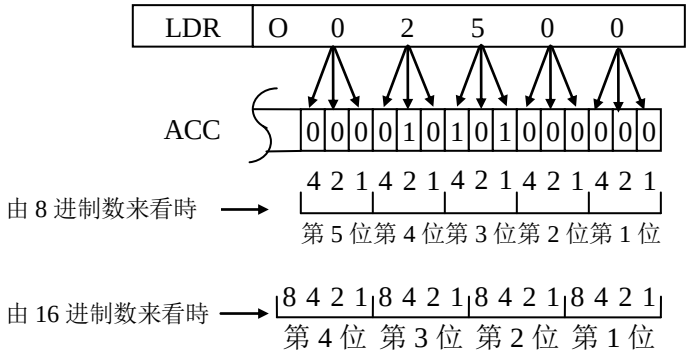
（6）8 进制常数指定（寄存器号）O（octal）

寄存器号以 8 进制数来分配，将该寄存器号作为数据在程序上进行处理时，以 O（octal）来指定。

具体用 LDR 指令

指令	操作数
LDR	O（A）

例：（A）为 2500



由操作常数指定 8 进制数，为二进制码，因此可进行二进制运算。

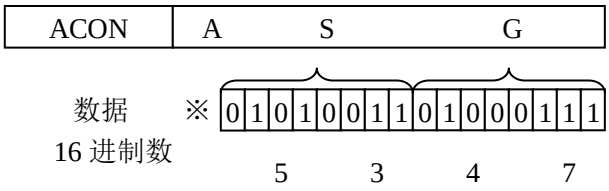
（7）ASCII 码指定 A（ASCII）

登记 ASCII 码数据时用。

指令	操作数
ACON	A（B）

在（B）处，可设定两个 0~9 及 A~Z 的英文字母、数字。

例：登记 ASCII 文字“SG”

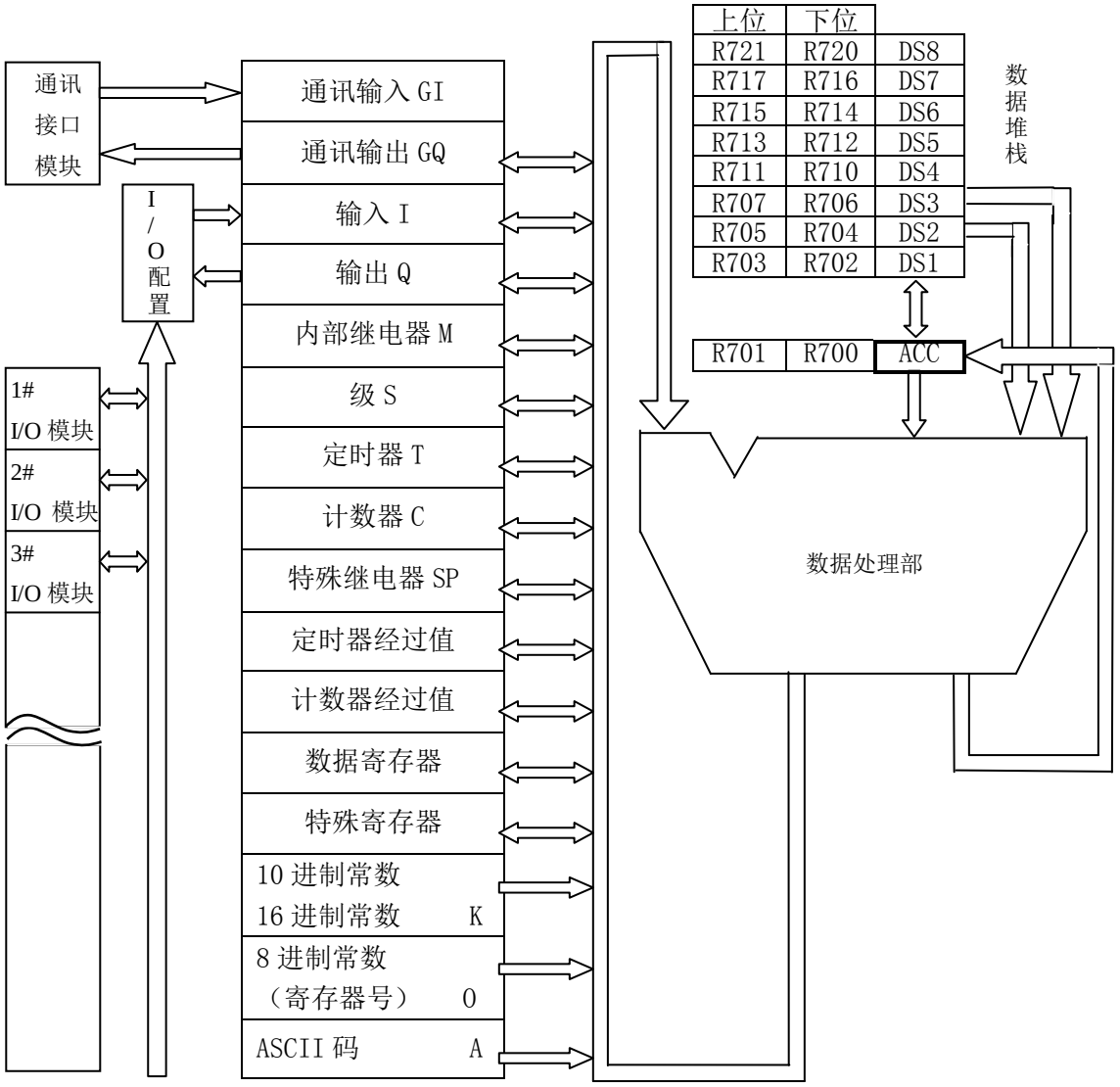


※请参见附录“ASCII 字码表”。

8.3 数据处理的基本形式

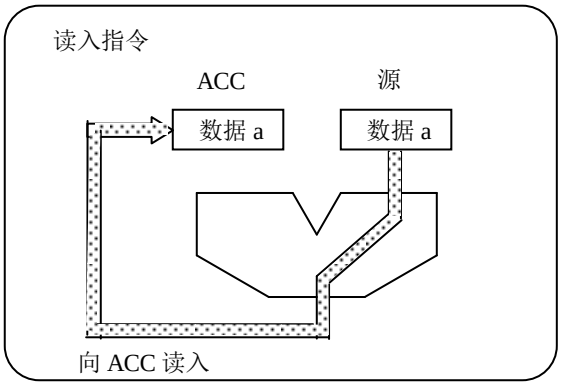
8.3.1 数据的流向

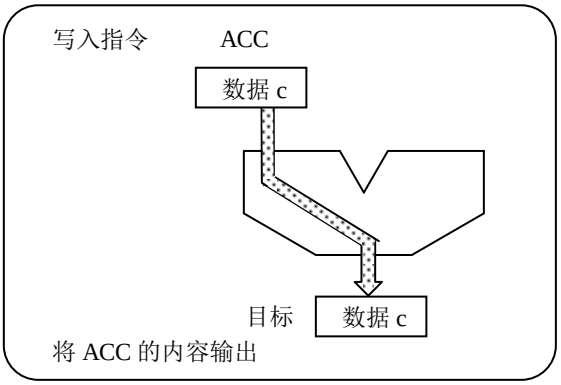
各机种 PLC 可处理各种存储器内的数据和常数，而这些数据处理基本上以 ACC（累加器）为中心进行处理。

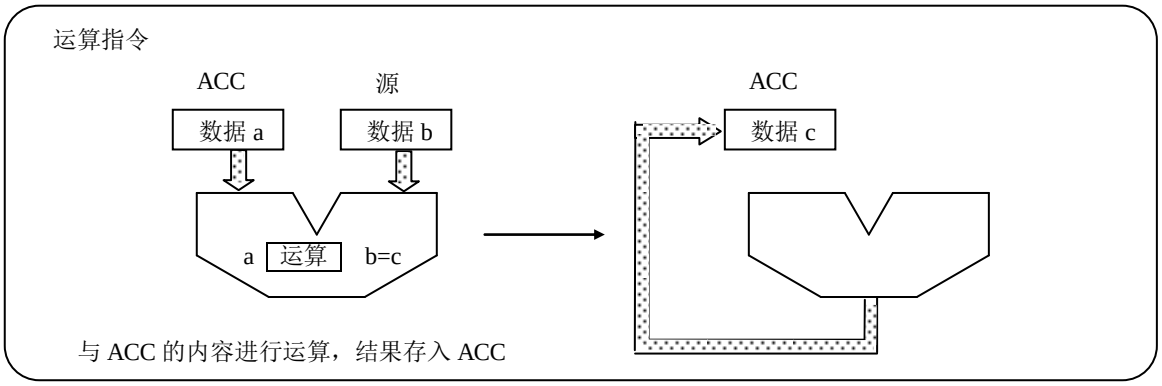


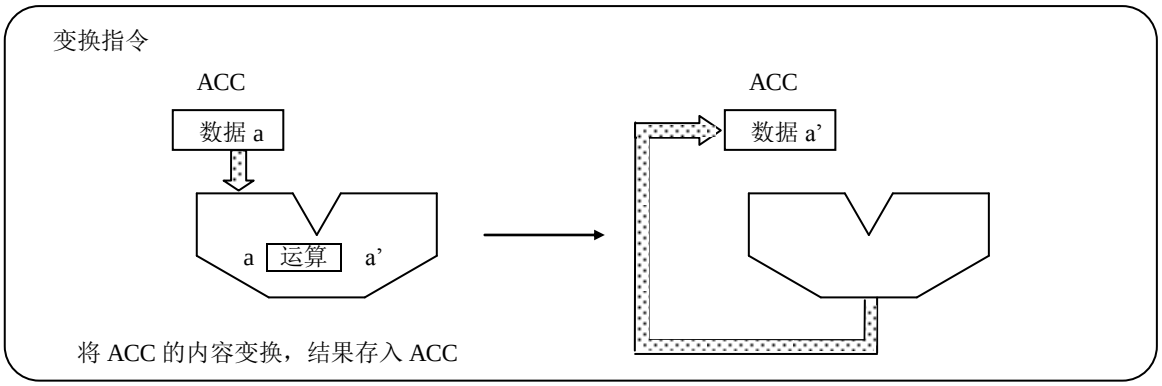
8.3.2 累加器的性质

累加器是一种特殊寄存器，它由 32 位构成，暂时存储数据处理结果。
基本的数据处理指令分为如下 4 种。



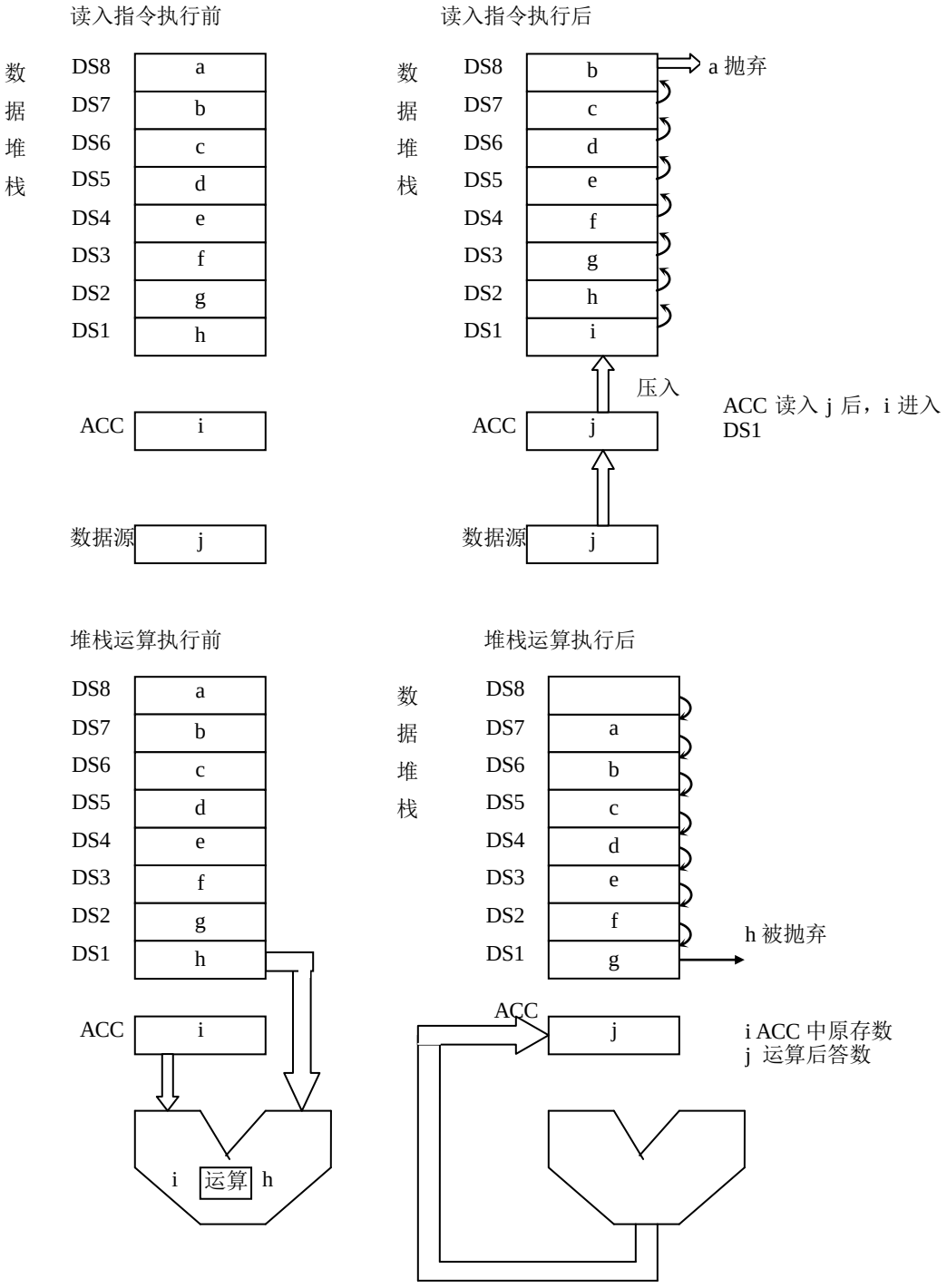






8.3.3 数据堆栈的性质

数据堆栈是累加器内的数据暂存用的特殊寄存器，执行读入指令时，累加器的内容被压入数据堆栈，由堆栈运算指令或 POP 指令取出。



另外，在许多有参照数据的指令中，数据堆栈与 ACC 一起，用于存储参照数据。

8.3.4 标志的性质

表示数据处理的结果数据以外的信息，有 13 种标志。

某些指令执行后，通过有关的标志继电器的 ON 或 OFF 状态来表示指令执行的部分结果。与该指令无关的标志继电器状态不变。

运算出错	小于	等于	大于	零	半借位	借位	半进位	进位	符号	溢出	数据出错	读零
SP053	SP060	SP061	SP062	SP063	SP064	SP065	SP066	SP067	SP070	SP073	SP075	SP076

1.运算出错标志（SP053）（ER1）

某些指令中，如操作数超出规定范围，而使运算结果出错，则为 ON

2.小于（SP060）、等于（SP061）、大于（SP062）标志。

执行比较指令时，ACC 的值与比较值相比，根据比较结果使对应的一个继电器为 ON 状态，另两个为 OFF 状态。

3.零标志（SP063）（Z）

处理结果为“0”时 ON，不为“0”时 OFF。

4.半借位标志（SP064）（HB）

相减的结果，由第 4 位向第 5 位借位时 ON，没有借位时 OFF。

5.借位标志（SP065）（B）

相减的结果，差为负值（数据为 100000000 补数）而向第 9 位借位时 ON，没有借位时 OFF。

6.半进位标志（SP066）（HC）

加法运算的结果从第 4 位向第 5 位进时 ON，没有进位 OFF。

7.进位标志（SP067）（C）

加法运算的结果，超过 8 位向第 9 位进位时 ON，没有进位时 OFF。

8.符号（SP070）（S）

实行装入指令或运算指令时，指令实行后，把 ACC 的最上位的状态作为该标志继电器的状态。

9.溢出（SP073）（OV）

处理的结果数据超过 32 位时 ON，正常时 OFF。

10.数据出错（SP075）（DE）

数据运算时，运算数据不是 BCD 数据时 ON，是 BCD 数据时 OFF。

11.读零标志（SP076）（RZ）

指令指定的源数据为“0”时 ON，不为“0”时 OFF。

注：在本手册中，提到 4 位，8 位数据，这时，位指的是由四位二进制数（bit）组成的一位 BCD 数或十六进制数。

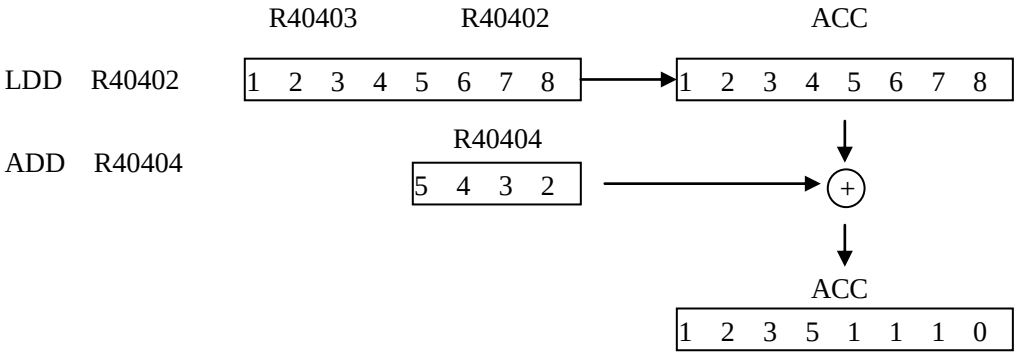
提到 16 位，32 位，此处的位，指二进制数的一位（bit）。

8.3.5 算术运算的思考方法

（1）运算位数

在算术运算中，除特例外，运算数据是指定的位数，被运算数据是 8 位数，运算结果除特例外也为 8 位数。

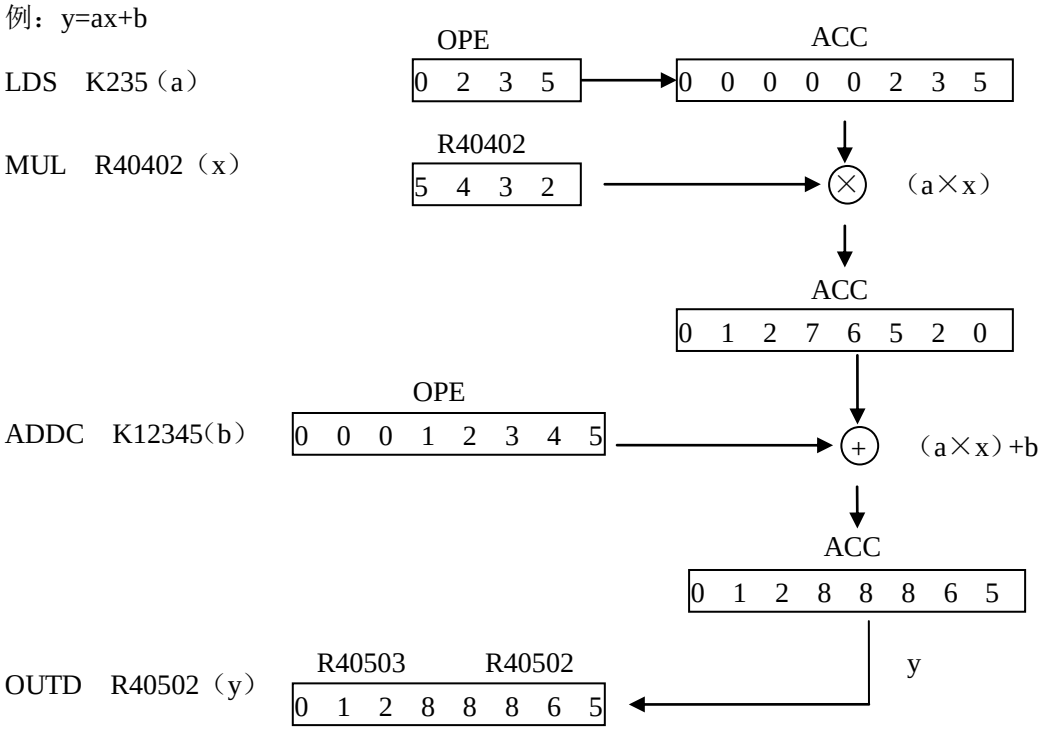
例：8 位数+4 位数＝8 位数



（2）算式的指令化

运算结果总是存在 ACC 里，对结果再进行别的运算时，可继续进行。

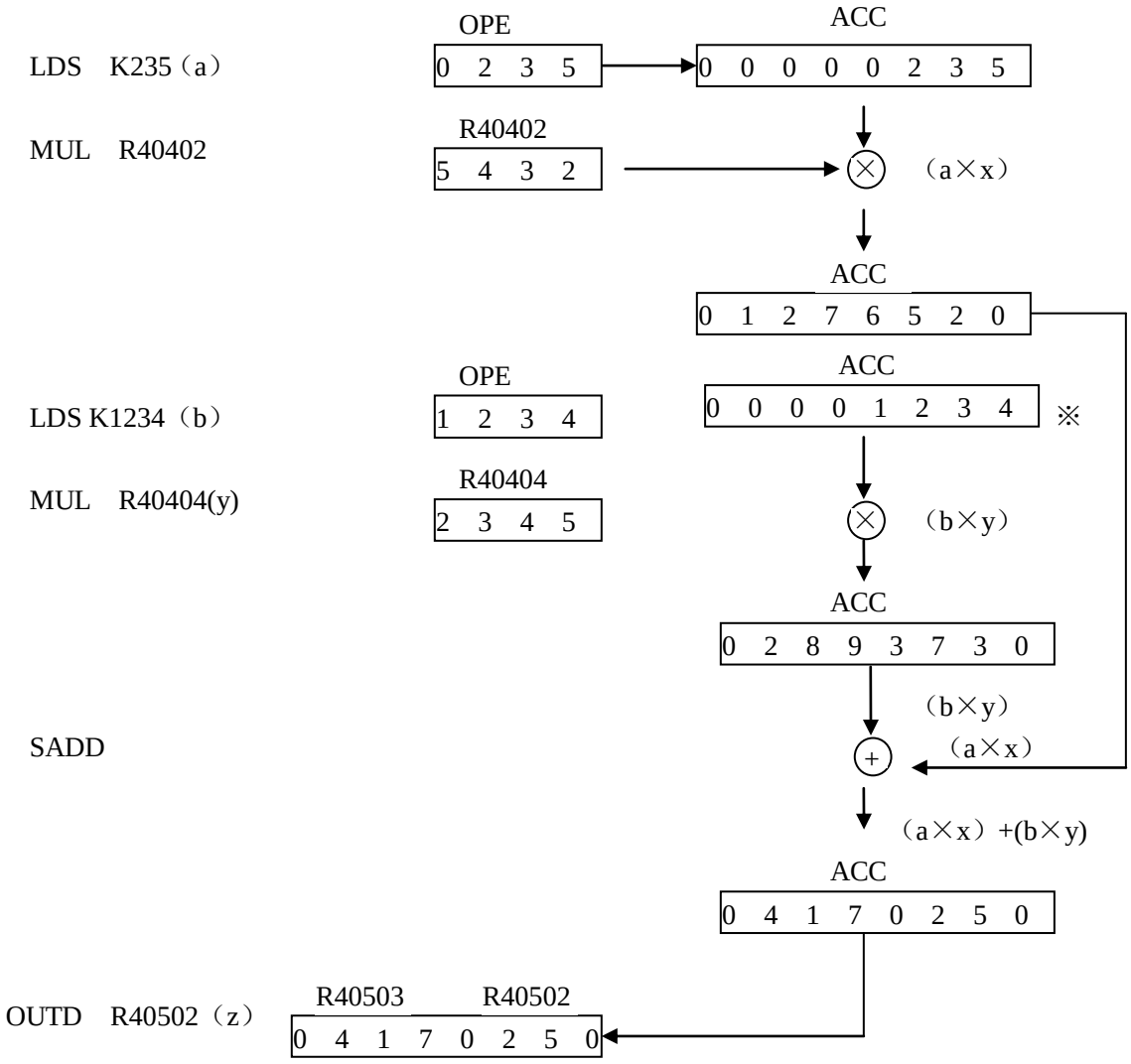
例：y=ax+b



注：“OPE”表示“操作数”。

对各分别算出的答案再进行运算时，用堆栈运算。

例：z=ax+by



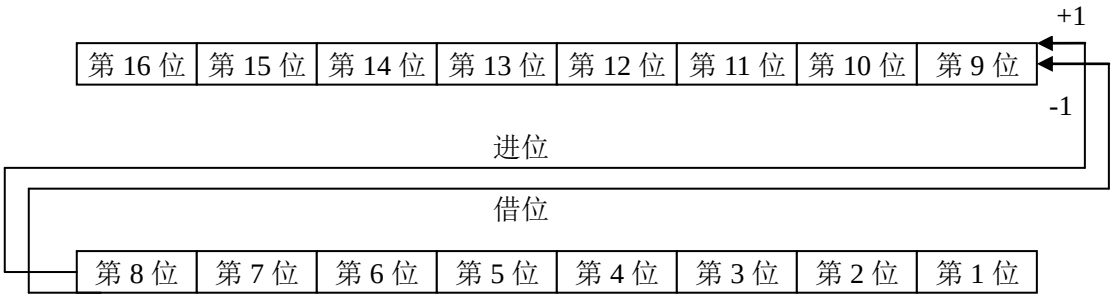
※执行 LDS K1234(b)指令时，把 ACC 中的原数值(01276520)压入数据堆栈。

(3)负值范围的数值。

本来 CPU 上基本只处理 8 位正整数。但如在使用方法上想想办法，负值范围的数据也可处理。

●只在正值范围使用时的思考方法

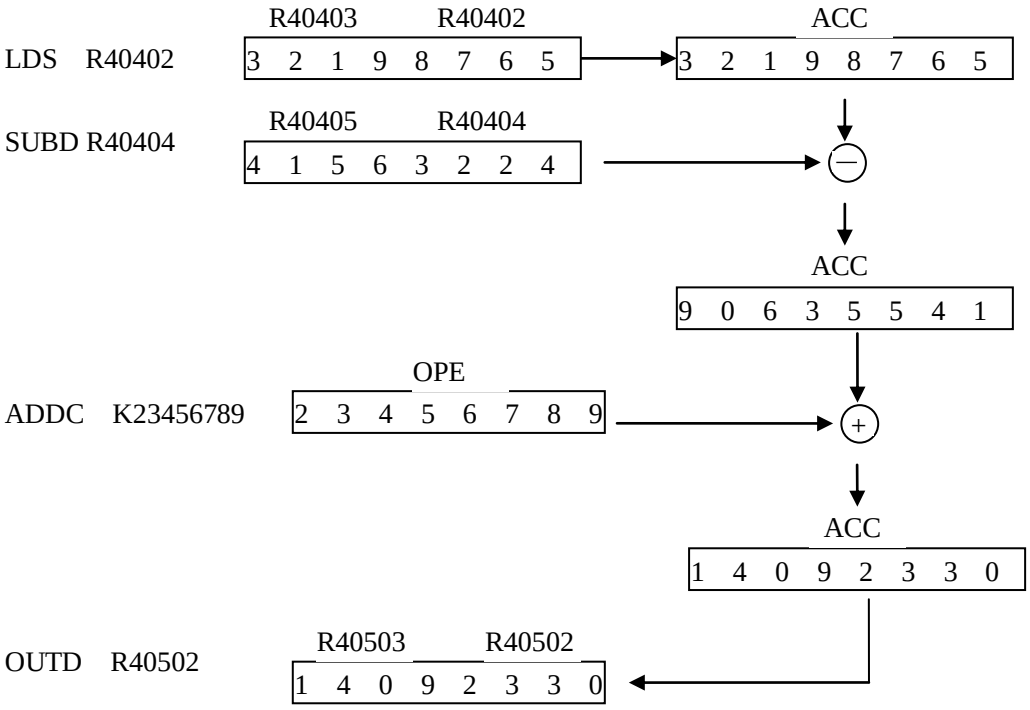
算术运算进行 8 位正整数运算，因此可将 8 位全部作为数据进行处理。此时 BCD 数据的范围为 0～99999999，二进制数据的范围为 0～FFFFFFFF，另外执行算术运算指令时的进位，成为向第 9 位进位的信号，而借位则为向第 9 位借位的信号，因此超过 8 位的数据也可管理。



●计算途中取负值而最终结果为正值时

进行算术运算时，中途数据一时为负值，而最终取正值时，以通常的处理即可达到目的。

例：



将该计算与带符号的数值相比如下：

带符号计算		数据处理
32198765	借位	32198765
－41563224		－41563224
－09364459		90635541
＋23456789	进位	＋23456789
＋14092330		14092330

该计算过程中，途中经过情况不同，但最终答案一致，是因为数据处理中借位的数与进位的数一致。

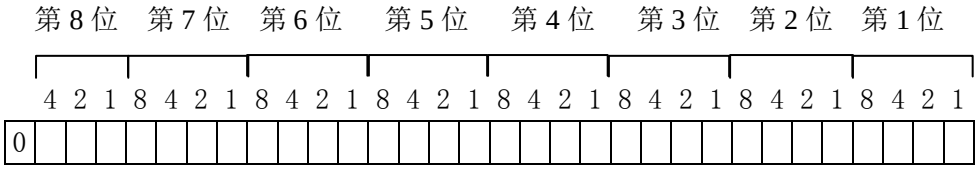
（因－100000000+100000000＝0）

最终的计算结果肯定是正值时，因借位与进位一致，用通常运算处理没问题。

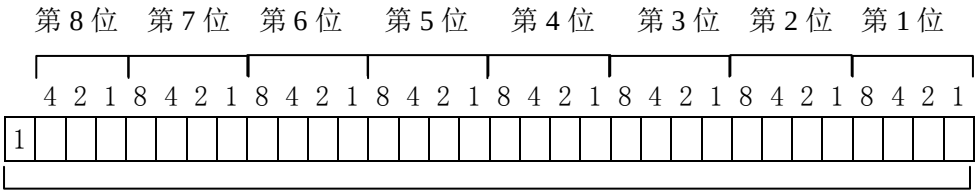
●数据在－19999999～79999999 范围内时

设定 32 位数据的最上位为符号位，“0”时为正值，“1”时为负值，数据看作 100000000 的补数。

正范围



负范围



最上位为“1”时，该数据为负值，存入的是该数据的 100000000 的补数。

以这样的方法管理的数据，外部输出数据等需要绝对值时，用 BCDCPL 指令求出绝对值再输出。

例：

带符号计算	
00000001	
－00000002	
99999999	←最上位为 1，因此为负值
补数变换	
00000001	←该值为绝对值

(4) 二进制运算时的符号处理

- 有符号、无符号的演算例

  下面是有符号运算及无符号运算时的例子程序。

```
例：    LDS     K 8080
        BMULS   K 20
        OUTD    R 2000
```

◆ 有符号

```
    ＦＦＦＦ８０８０（符号扩展）
×             ２０
--------------------
    ＦＦＦ０１０００
```

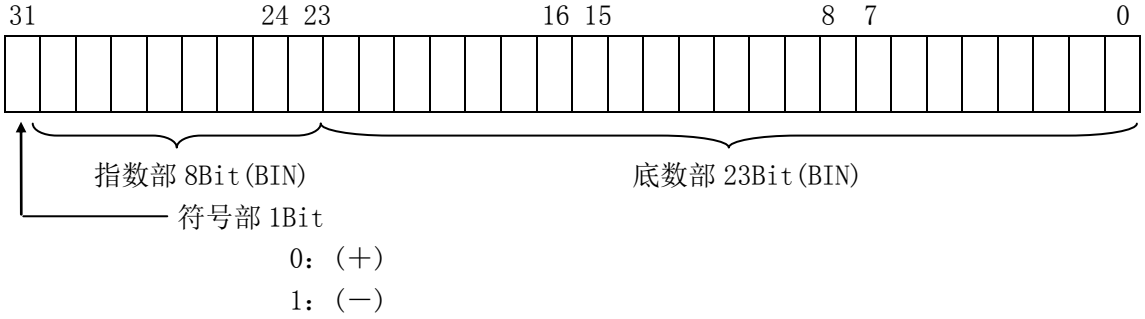
◆ 无符号

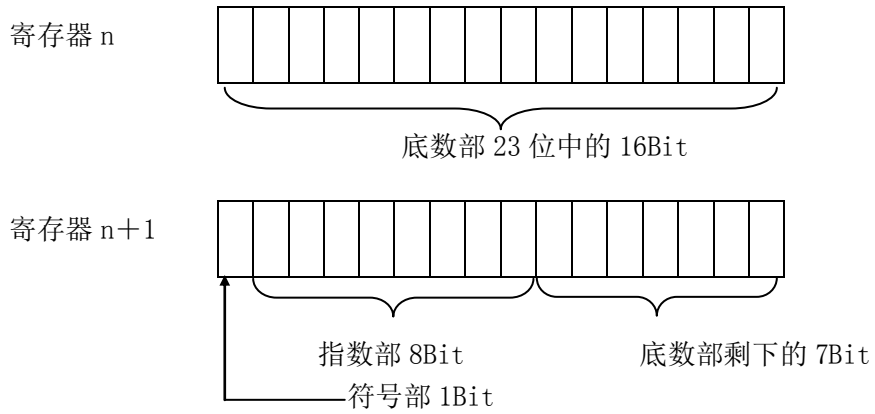
```
    ０００００８０８０（无符号扩展）
×             ２０
--------------------
    ００１０１０００
```

注意：有符号和无符号的运算结果可能会不相同。

8.4 浮点数的形式

D4-454 CPU 可存取的浮点数的形式为 IEEE 标准(IEEE 标准 745－1985)的 32 位单精度数。





※ 浮点数的表现形式如下表

指数部 / 底数部	0	非全 0 或全 1	全 Bit 位为 1
0	0	正规化数	无穷大
0 以外	非正规化数	正规化数	非数

1） 正规化数
正规化数值表示为： （符号部±）$2^{(指数部)-127} \times (1+底数部\times 2^{-23})$

2） 非正规化数
非正规化数值表示为：（符号部±）$2^{(指数部)-127} \times (底数部\times 2^{-23})$

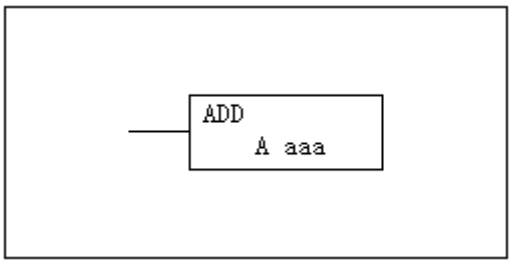
3） 无穷大
其值表示为：－∞、＋∞

4） 非数
没有对应的数值或对应的无穷大数。

第 9 章 算术运算指令

9.1 4 位 BCD 加法指令（ADD）

ADD 指令是一个 16 位指令，将累加器中的 BCD 数同指定寄存器（A aaa）中的 BCD 数相加。结果存放在累加器中。

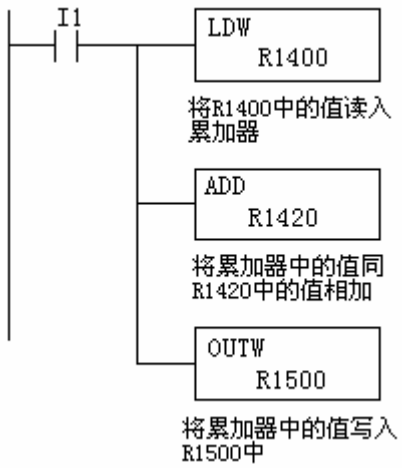


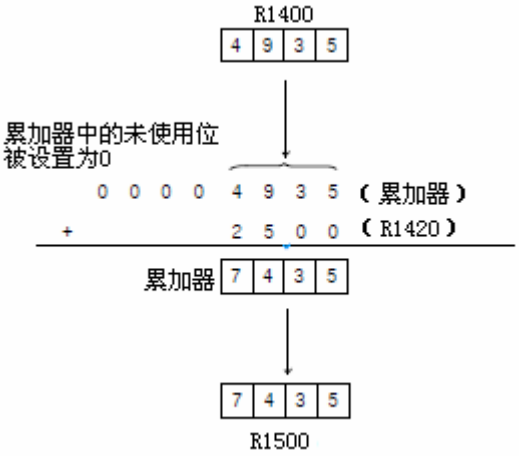
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，ADD 指令将累加器中低 16 位的数据同 R1420 中的数据相加，OUTW 指令将累加器中的数据写入 R1500 中。



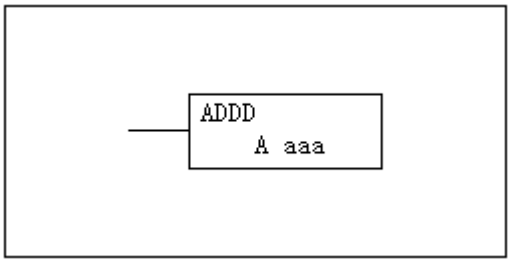


语句视图

```
LD I1
LDW R1400
ADD R1420
OUTW R1500
```

9.2 8 位 BCD 加法指令（ADDD）

ADDD 指令是一个 32 位指令，将累加器中的 BCD 数同指定寄存器（A aaa）中的 BCD 数相加，结果存放在累加器中。

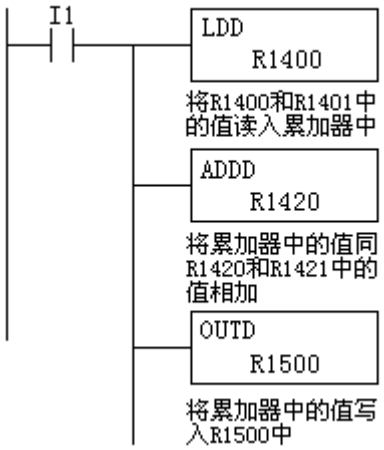


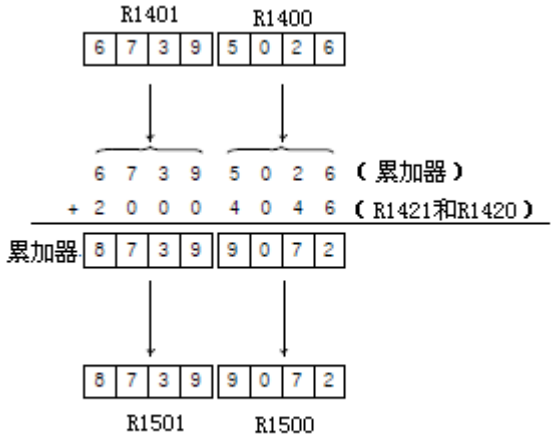
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，ADDD 指令将累加器中的数据同 R1420 和 R1421 中的数据相加，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。



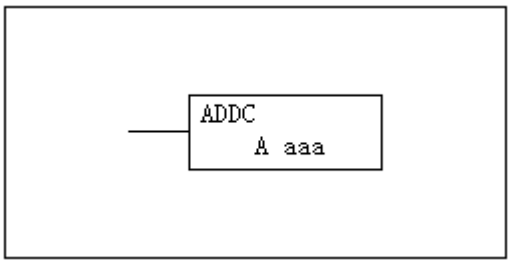


语句视图

```
LD I1
LDD R1400
ADDD R1420
OUTD R1500
```

9.3 8 位 BCD 常数加法指令（ADDC）

ADDC 指令是一个 32 位指令，将累加器中的 BCD 数同一个 8 位 BCD 常数相加，结果存放在累加器中。



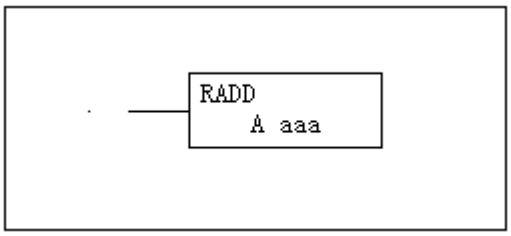
操作数类型		D4-454 范围
	A	aaa
常数	K	0-99999999

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.4 实数加法指令（RADD）

RADD 指令是一个 32 位指令，将累加器中的实数同两个连续寄存器中的实数相加。相加的两个实数必须是 IEEE 实数格式，结果存放在累加器中。



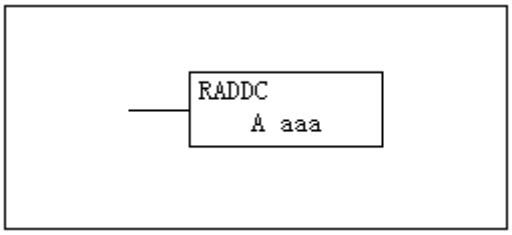
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.5 实数常数加法指令（RADDC）

RADDC 指令是一个 32 位指令，将累加器中的实数同一个实数常数相加。相加的两个实数必须是 IEEE 实数格式，结果存放在累加器中。



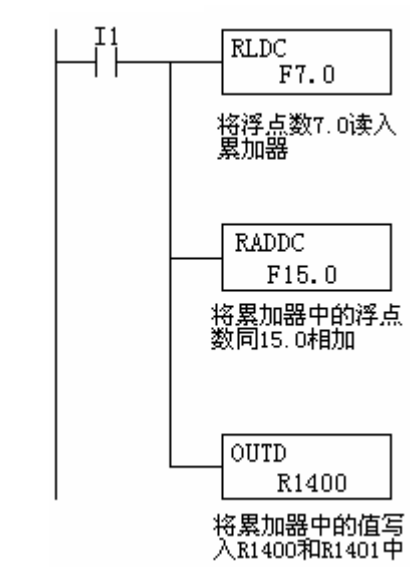
操作数类型		D4-454 范围
	A	aaa
常数	F	-3.402823E+038～+3.402823E+038

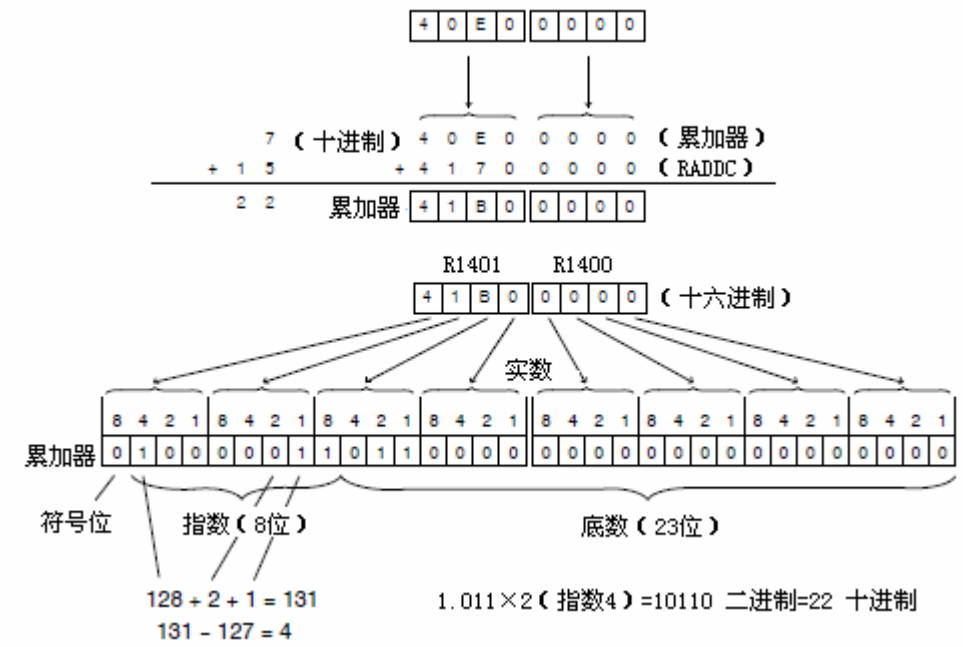
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

语句视图

```
LD  I1
RLDC  F7.0
RADDC  F15.0
OUTD  R1400
```

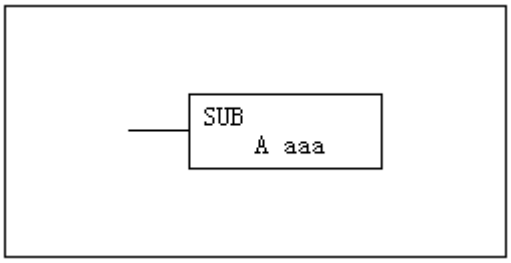




注意：如果指令中的实数比累加器中的实数小 16,777,216 倍，指令将不执行。

9.6 4 位 BCD 减法指令（SUB）

SUB 指令是一个 16 位指令，将累加器中的 BCD 数减去指定寄存器（A aaa）中的 BCD 数。结果存放在累加器中。

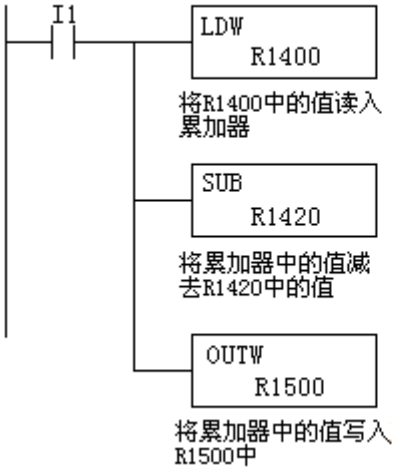


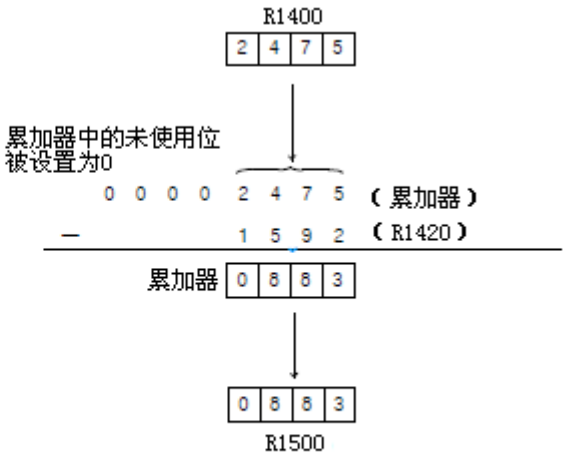
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，SUB 指令将累加器中低 16 位的数据减去 R1420 中的数据，OUTW 指令将累加器中的数据写入 R1500 中。



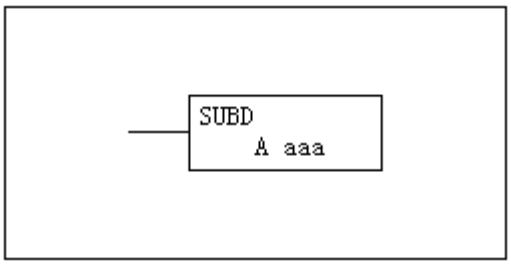


语句视图

```
LD I1
LDW R1400
SUB R1420
OUTW R1500
```

9.7 8 位 BCD 减法指令（SUBD）

SUBD 指令是一个 32 位指令，将累加器中的 BCD 数减去指定寄存器(A aaa)中的 BCD 数，结果存放在累加器中。

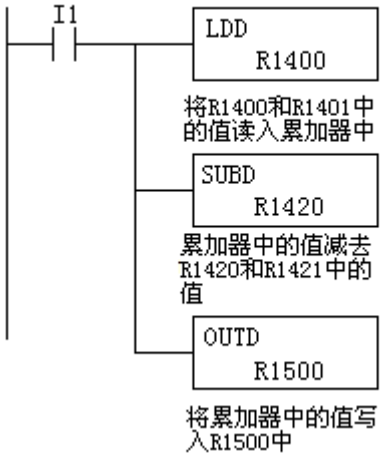


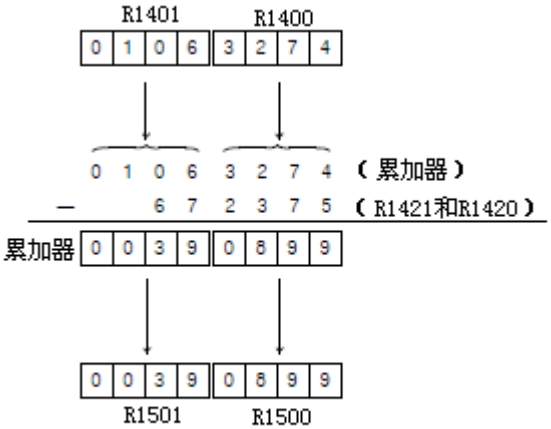
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，SUBD 指令将累加器中的数据减去 R1420 和 R1421 中的数据，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。



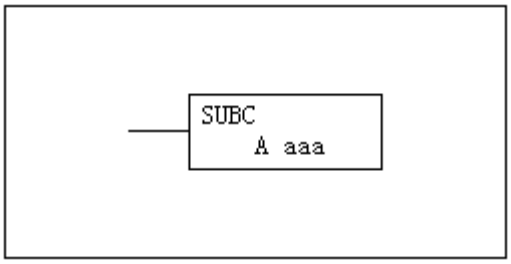


语句视图

```
LD I1
LDD R1400
SUBD R1420
OUTD R1500
```

9.8 8 位 BCD 常数减法指令（SUBC）

SUBC 指令是一个 32 位指令，将累加器中的 BCD 数减去一个 8 位 BCD 常数，结果存放在累加器中。



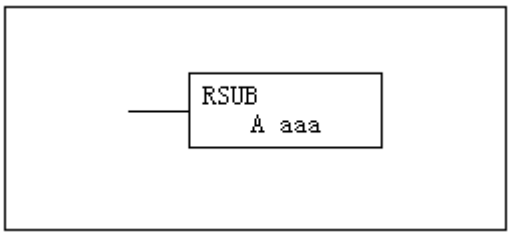
操作数类型	D4-454 范围
A	aaa
常数 K	0-99999999

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.9 实数减法指令（RSUB）

RSUB 指令是一个 32 位指令，将累加器中的实数减去两个连续寄存器中的实数。两个实数必须都是 IEEE 实数格式。运算结果是 32 位实数，存放在累加器中。



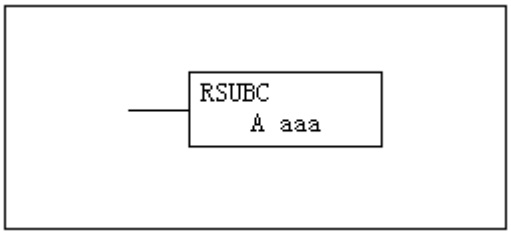
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.10 实数常数减法指令（RSUBC）

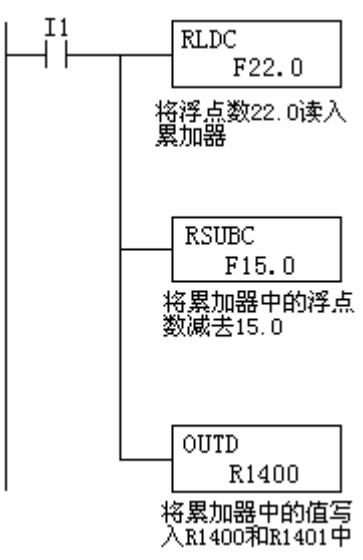
RSUBC 指令是一个 32 位指令，将累加器中的实数减去一个实数常数。两个实数必须是 IEEE 实数格式，结果存放在累加器中。



操作数类型		D4-454 范围
	A	aaa
常数	F	-3.402823E+038～+3.402823E+038

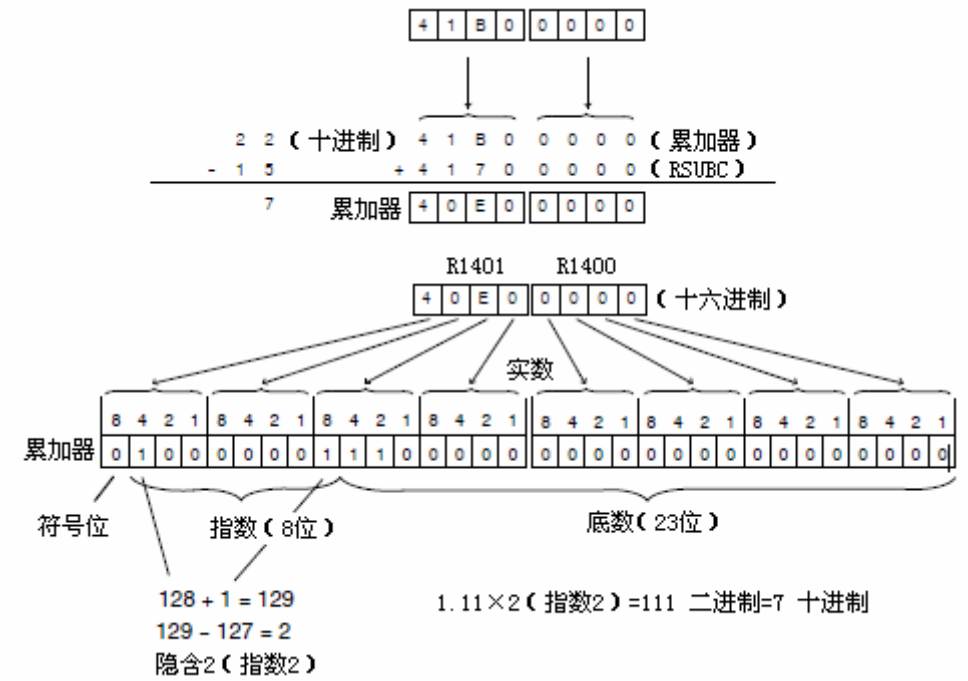
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。



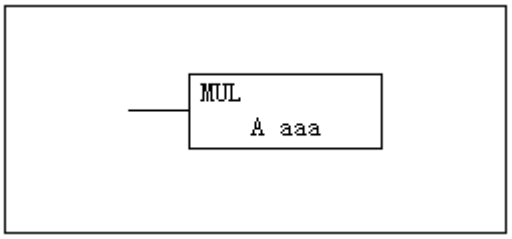
语句视图

```
LD I1
RLDC F22.0
RSUBC F15.0
OUTD R1400
```



9.11 4 位 BCD 乘法指令（MUL）

MUL 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数乘以指定寄存器（A aaa）中的 BCD 数，结果有可能是 8 位 BCD 数，存放在累加器中。

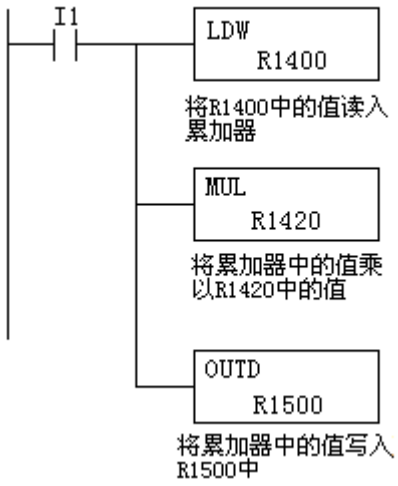


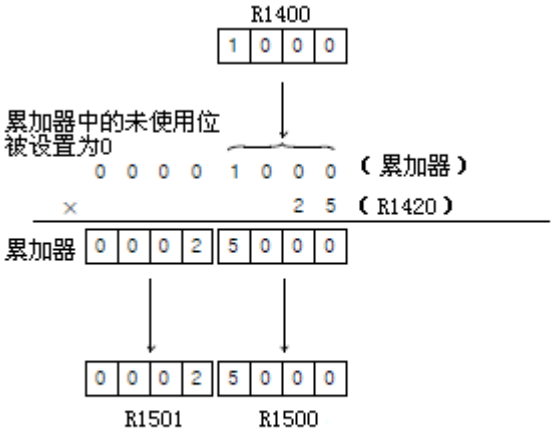
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，MUL 指令将累加器中低 16 位的数据乘以 R1420 中的数据，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。



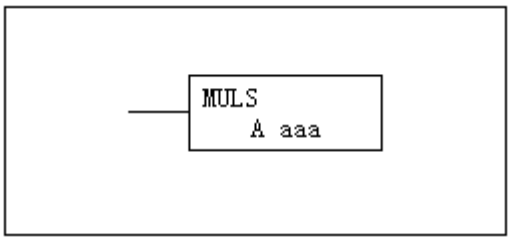


语句视图

```
LD I1
LDW R1400
MUL R1420
OUTD R1500
```

9.12 4 位 BCD 常数乘法指令（MULS）

MULS 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数乘以一个 4 位(最大)BCD 常数，结果有可能是 8 位 BCD 数，存放在累加器中。



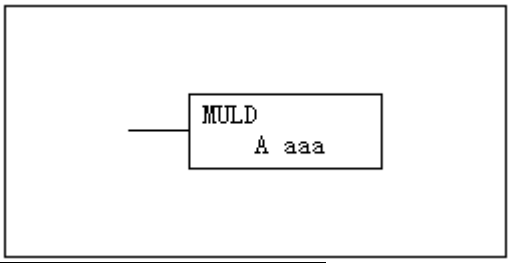
操作数类型		D4-454 范围
	A	aaa
常数	K	0-9999

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.13 8 位 BCD 乘法指令（MULD）

MULD 指令是一个 32 位指令，将累加器中的 8 位 BCD 数乘以两个连续寄存器（A aaa）中的 BCD 数，结果的低 8 位存放在累加器中，高位存放在数据堆栈中。

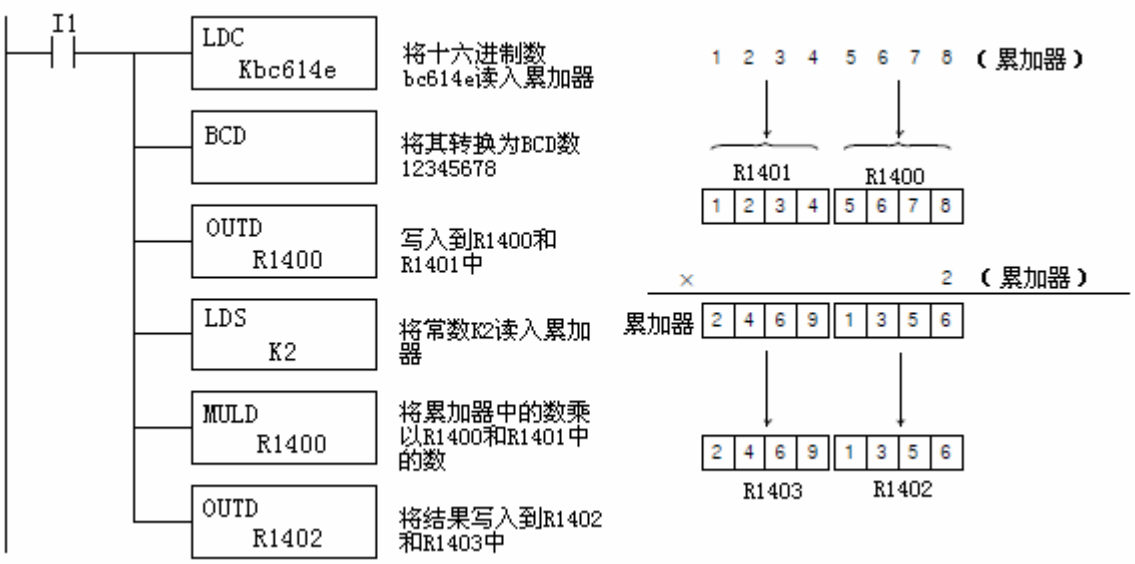


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDC 指令将十六进制常数 Kbc614e 读入累加器中，BCD 指令将其转换为 BCD 数 12345678，写入到 R1400 和 R1401 中。LDS 指令将常数 K2 读入累加器中，MULD 指令将累加器中的数据乘以 R1400 和 R1401 中的数据 12345678，OUTD 指令将累加器中的数据写入 R1402 和 R1403 中。

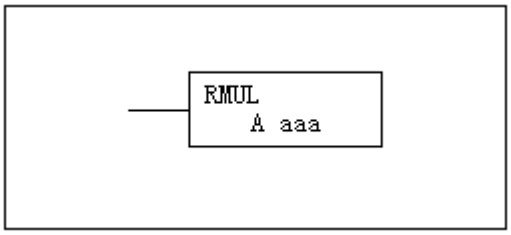


语句视图

```
LD I1
LDC Kbc614e
BCD
OUTD R1400
LDS K2
MULD R1400
OUTD R1402
```

9.14 实数乘法指令（RMUL）

RMUL 指令将累加器中的实数乘以两个连续寄存器中的实数。两个实数必须都是 IEEE 实数格式，结果存放在累加器中。



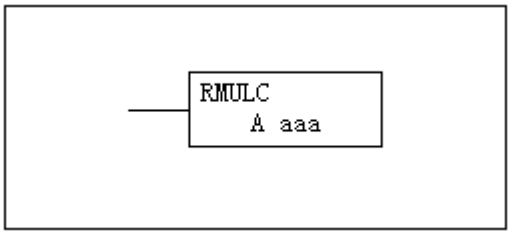
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

9.15 实数常数乘法指令（RMULC）

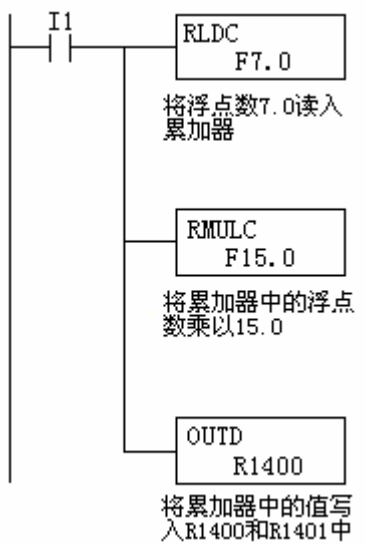
RMULC 指令将累加器中的实数乘以一个实数常数。两个实数必须是 IEEE 实数格式。结果存放在累加器中。



操作数类型		D4-454 范围
	A	aaa
常数	F	-3.402823E+038～+3.402823E+038

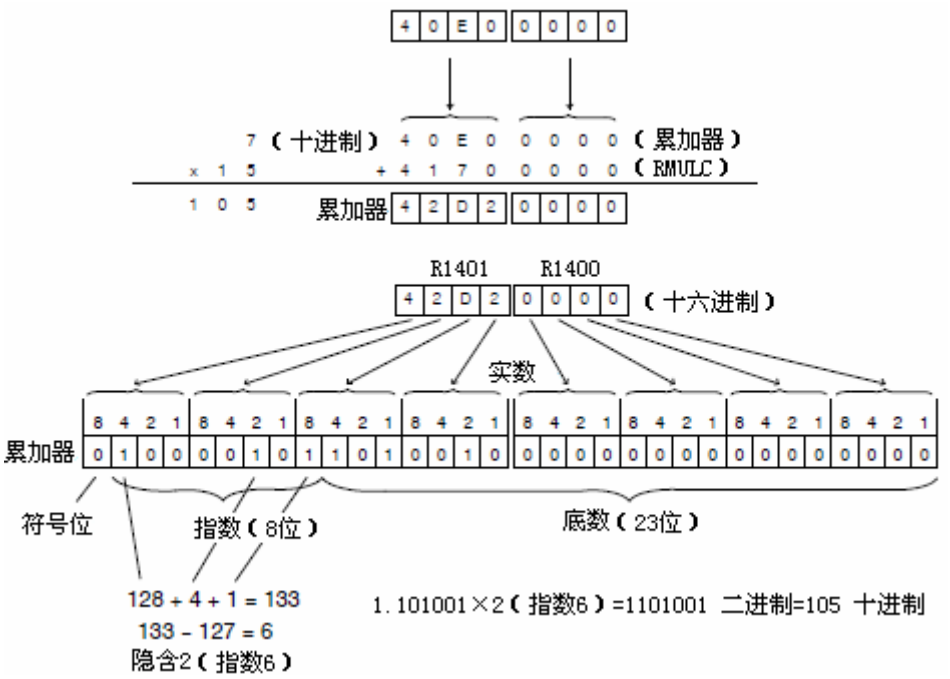
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。



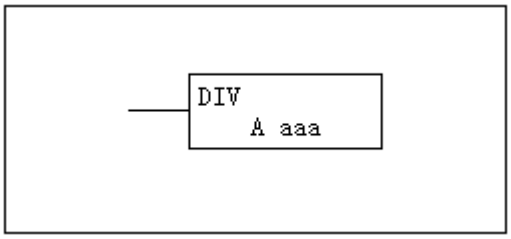
语句视图

```
LD I1
RLDC F7.0
RMULC F15.0
OUTD R1400
```



9.16 4 位 BCD 除法指令（DIV）

DIV 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数除以指定寄存器（A aaa）中的 BCD 数，商存放在累加器中，余数存放在数据堆栈的第 1 级中。

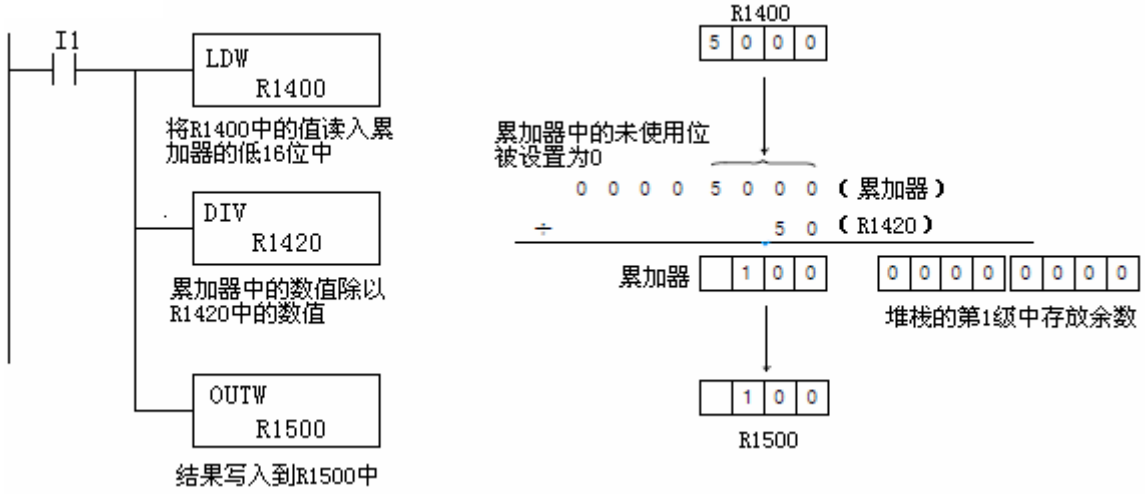


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP53	操作数超出累加器处理范围时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，DIV 指令将累加器中低 16 位的数据除以 R1420 中的数据，OUTW 指令将累加器中的数据写入 R1500 中。

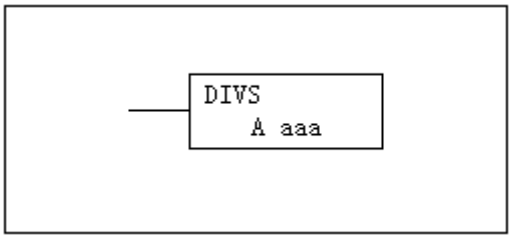


语句视图

```
LD I1
LDW R1400
DIV R1420
OUTW R1500
```

9.17 4 位 BCD 常数除法指令（DIVS）

DIVS 指令是一个 16 位指令，将累加器中低 16 位中的 BCD 数除以一个 4 位（最大）BCD 常数，商存放在累加器中，余数存放在数据堆栈第 1 级中。



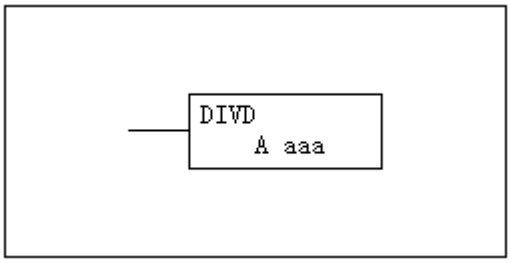
操作数类型	D4-454 范围
A	aaa
常数 K	1-9999

受影响的标志线圈	描述
SP53	操作数超出累加器处理范围时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.18 8 位 BCD 除法指令（DIVD）

DIVD 指令是一个 32 位指令，将累加器中的 BCD 数除以指定的两个连续寄存器（A aaa）中的 BCD 数，商存放在累加器中，余数存放在数据堆栈的第 1 级中。

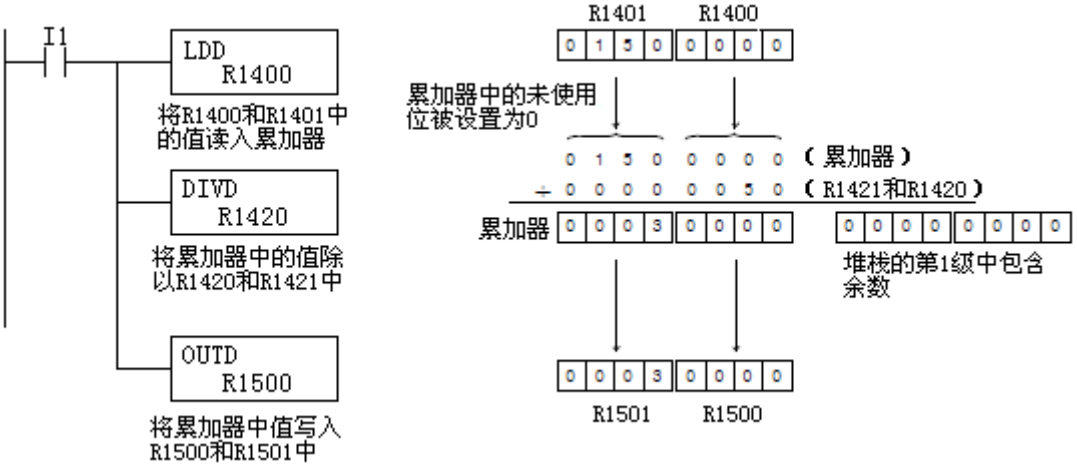


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP53	操作数超出累加器处理范围时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器中，DIVD 指令将累加器中的数据除以 R1420 和 R1421 中的数据，商存放在累加器中，余数存放在数据堆栈的第 1 级中，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

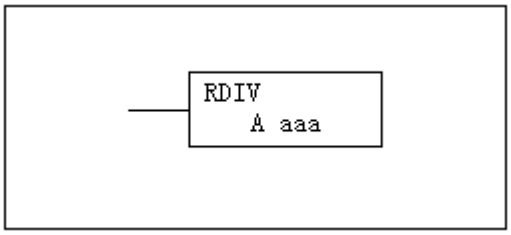


语句视图

```
LD I1
LDD R1400
DIVD R1420
OUTD R1500
```

9.19 实数除法指令（RDIV）

RDIV 指令将累加器中的实数除以两个连续寄存器中的实数。两个实数必须都是 IEEE 实数格式，结果存放在累加器中。



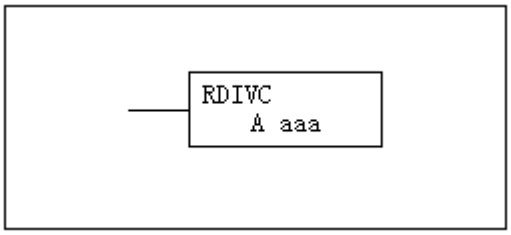
操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.20 实数常数除法指令（RDIVC）

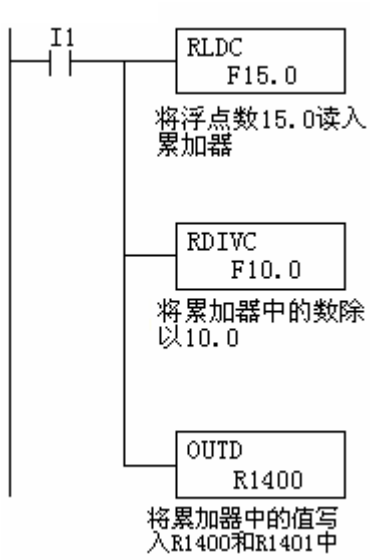
RDIVC 指令将累加器中的实数除以一个实数常数。两个实数必须是 IEEE 实数格式。结果存放在累加器中。



操作数类型		D4-454 范围
	A	aaa
常数	F	-3.402823E+038～+3.402823E+038

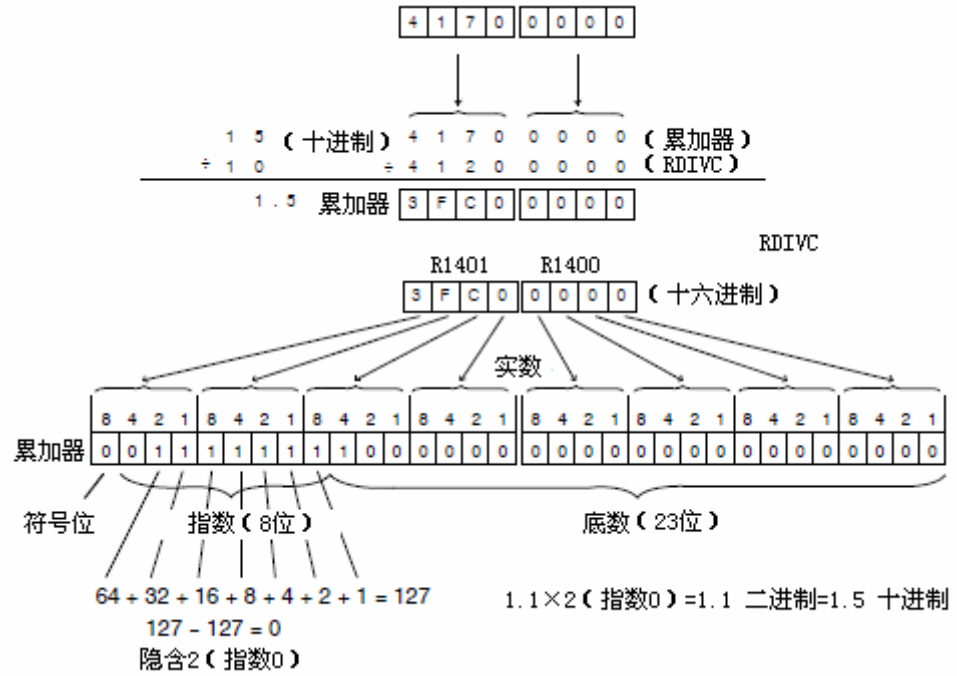
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP71	指定了不存在间接寄存器的区域时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP74	实数运算结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。



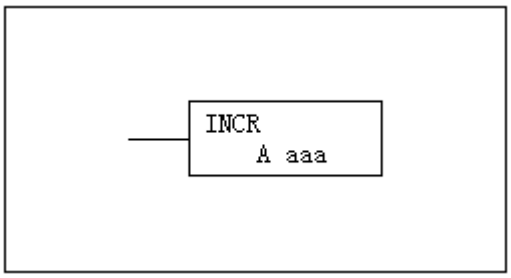
语句视图

```
LD I1
RLDC F15.0
RDIVC F10.0
OUTD R1400
```



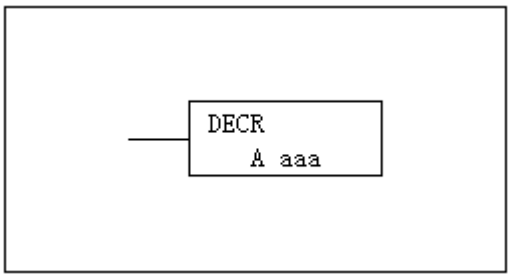
9.21 BCD 加 1 指令（INCR）

INCR 指令对由操作数指定的寄存器中存储的数据进行加“1”运算，指令执行一次，就进行一次加“1”运算。



9.22 BCD 减 1 指令（DECR）

DECR 指令对由操作数指定的寄存器中存储的数据进行减“1”运算，指令执行一次，就进行一次减“1”运算。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，每当 M5 为 ON 时，R1400 中的数据就增加 1。

语句视图

```
LD M5
INCR R1400
```

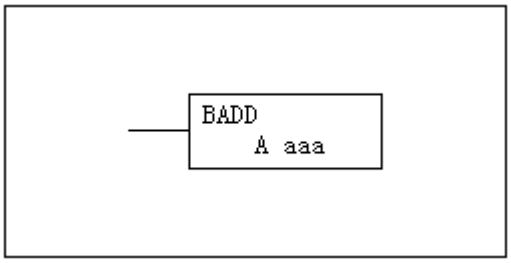
下面的例子中，每当 M5 为 ON 时，R1400 中的数据就减 1。

语句视图

```
LD M5
DECR R1400
```

9.23 16 位（BIN）加法指令（BADD）

BADD 指令是一个 16 位指令，将累加器低 16 位中的数据同指定寄存器（A aaa）中的二进制数相加，结果有可能为 32 位，存放在累加器中。

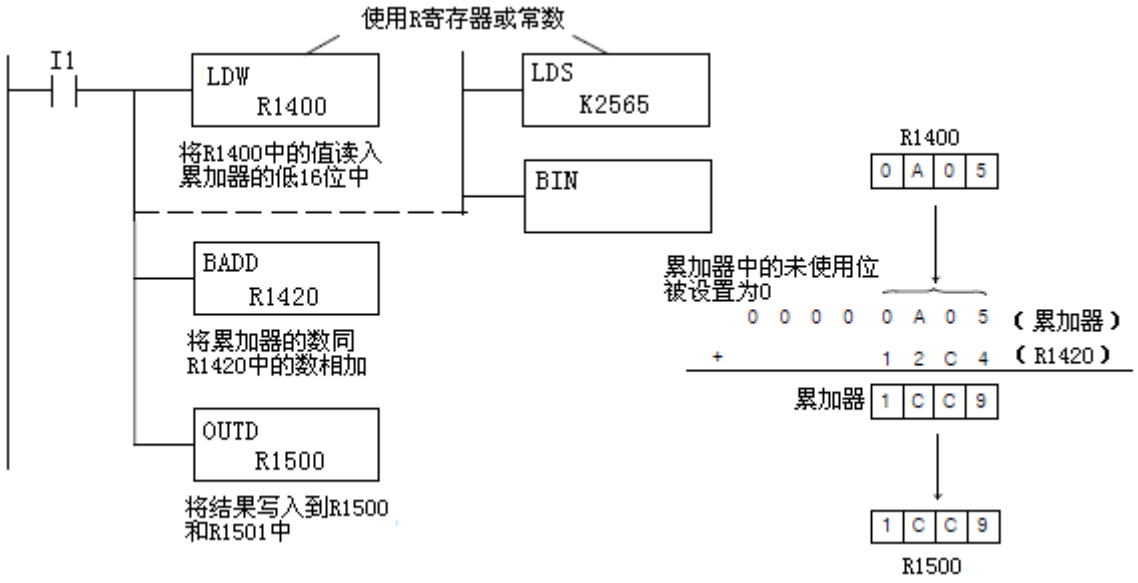


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BADD 指令将累加器中低 16 位的数据同 R1420 中的数据相加，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

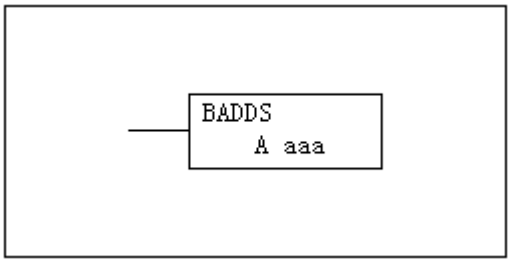


语句视图

```
LD I1
LDW R1400
BADD R1420
OUTD R1500
```

9.24 16 位常数（BIN）加法指令（BADDS）

BADDS 指令是一个 16 位指令，将累加器低 16 位中的数据同一个 4 位常数相加，结果有可能为 32 位，存放在累加器中。



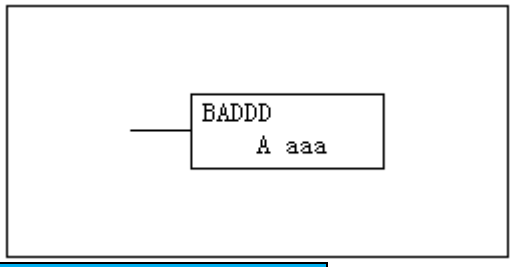
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFF

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.25 32 位（BIN）加法指令（BADDD）

BADDD 指令是一个 32 位指令，将累加器中的二进制数同指定的两个连续寄存器（A aaa）中的二进制数相加，结果存放在累加器中。

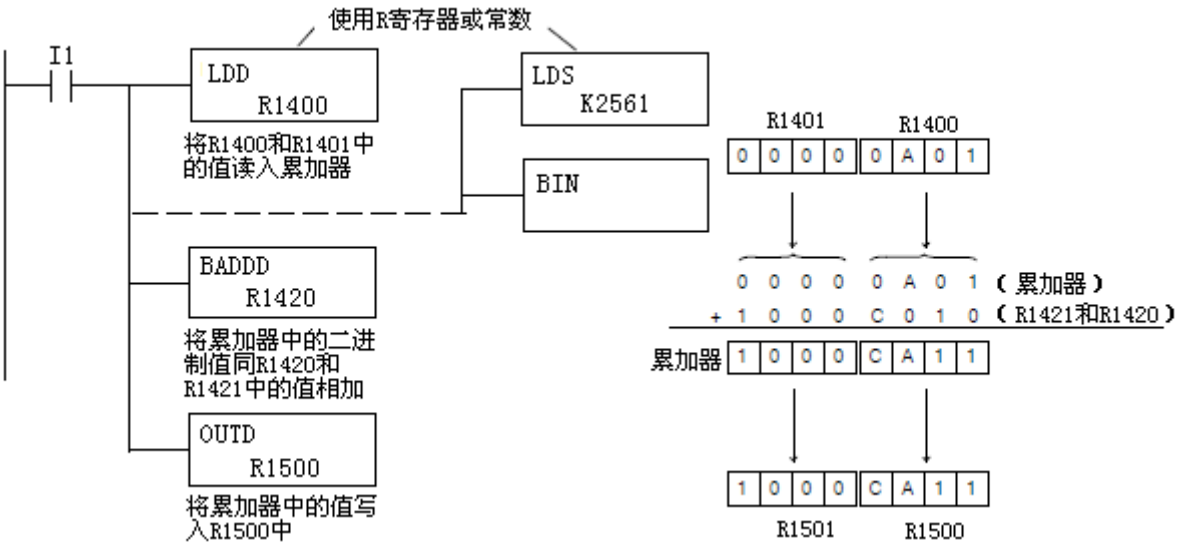


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，BADDD 指令将累加器中的二进制数据同 R1420 和 R1421 中的二进制数据相加，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

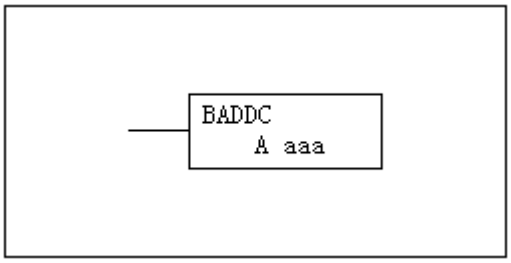


语句视图

```
LD I1
LDD R1400
BADDD R1420
OUTD R1500
```

9.26 32 位常数（BIN）加法指令（BADDC）

BADDC 指令是一个 32 位指令，将累加器中的二进制数同一个 8 位（最大）常数相加，结果存放在累加器中。



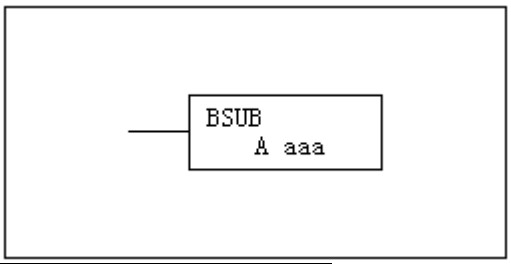
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFFFFFF

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.27 16 位（BIN）减法指令（BSUB）

BSUB 指令是一个 16 位指令，将累加器低 16 位中的数据减去指定寄存器（A aaa）中的二进制数，结果存放在累加器中。

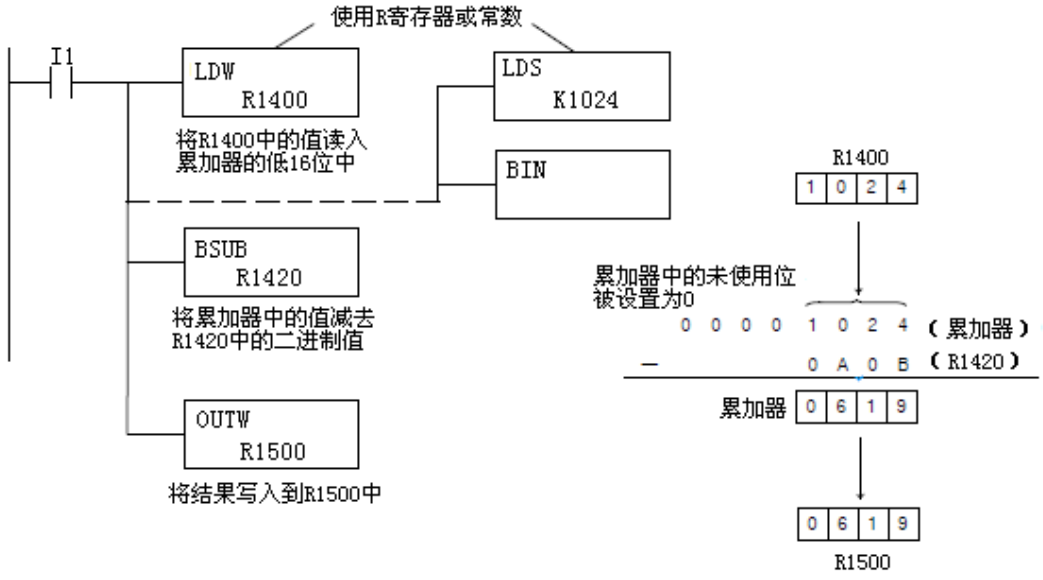


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BSUB 指令将累加器中低 16 位的数据减去 R1420 中的二进制数据，OUTW 指令将累加器中的数据写入 R1500 中。

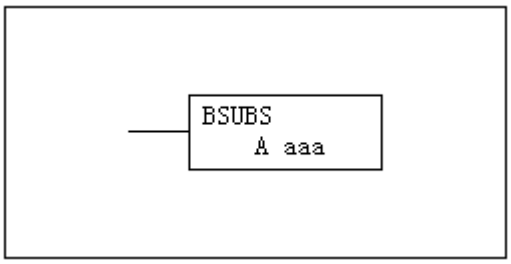


语句视图

```
LD I1
LDW R1400
BSUB R1420
OUTW R1500
```

9.28 16 位常数（BIN）减法指令（BSUBS）

BSUBS 指令是一个 16 位指令，将累加器低 16 位中的数据减去一个 4 位（最大）常数，结果存放在累加器中。



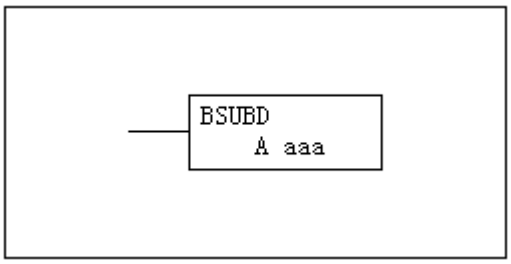
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFF

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.29 32 位（BIN）减法指令（BSUBD）

BSUBD 指令是一个 32 位指令，将累加器中的二进制数减去指定的两个连续寄存器(A aaa)中的二进制数，结果存放在累加器中。

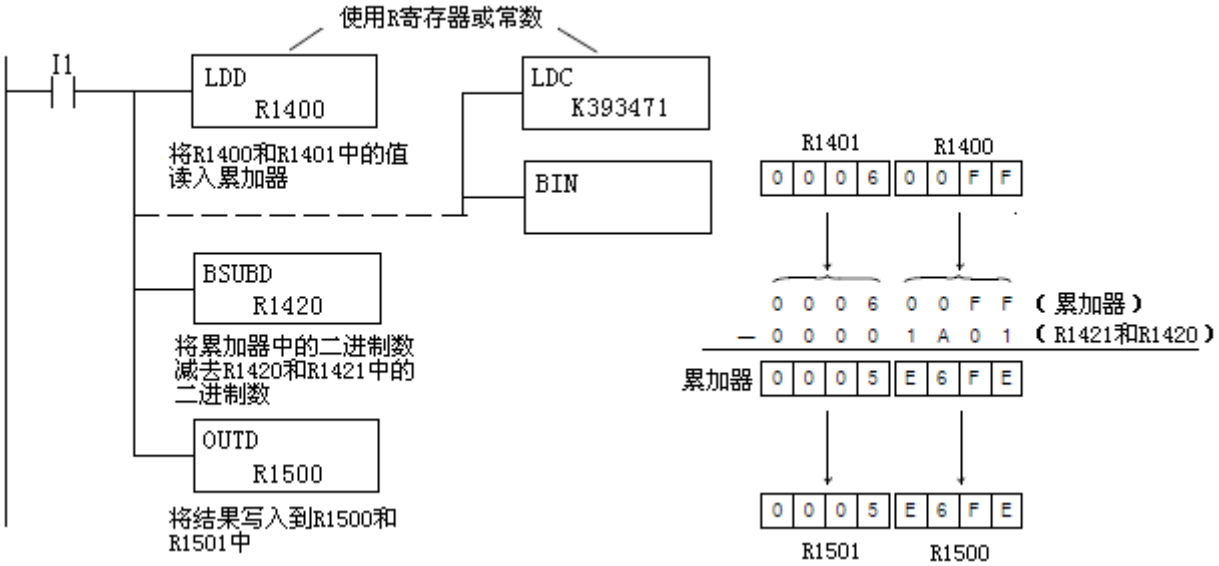


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据装入累加器中，BSUBD 指令将累加器中的二进制数减去 R1420 和 R1421 中的二进制数，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

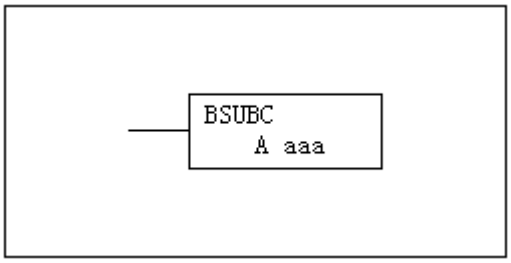


语句视图

```
LD I1
LDD R1400
BSUBD R1420
OUTD R1500
```

9.30 32 位常数（BIN）减法指令（BSUBC）

BSUBC 指令是一个 32 位指令，将累加器中的二进制数减去一个 8 位（最大）常数，结果存放在累加器中。



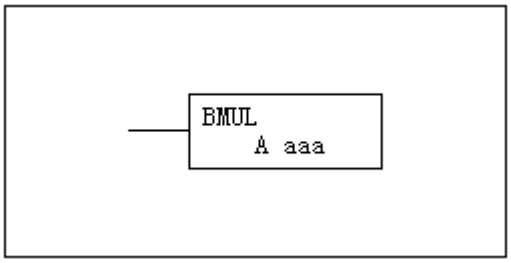
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFFFFFF

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.31 16 位（BIN）乘法指令（BMUL）

BMUL 指令是一个 16 位指令，将累加器低 16 位中的数据乘以指定寄存器（A aaa）中的二进制数，结果可能会达到 32 位，存放在累加器中。

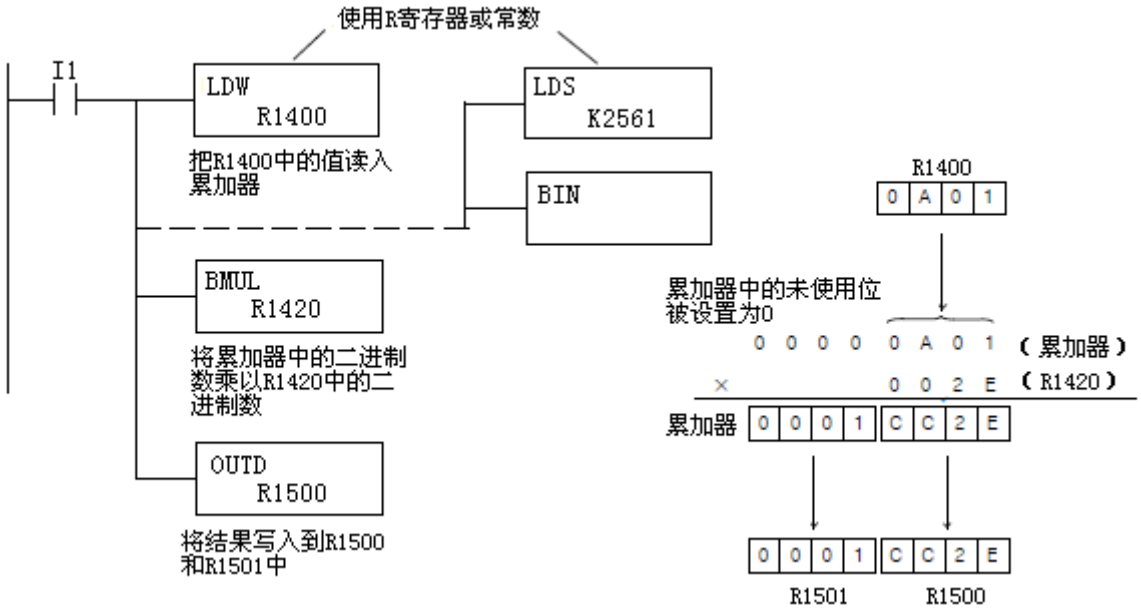


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BMUL 指令将累加器中低 16 位的数据乘以 R1420 中的二进制数，OUTD 指令将累加器中的数据写入 R1500 和 R1501 中。

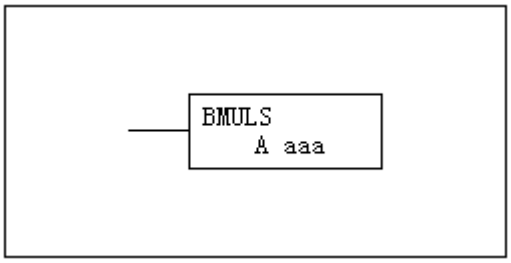


语句视图

```
LD I1
LDW R1400
BMUL R1420
OUTD R1500
```

9.32 16 位（BIN）常数乘法指令（BMULS）

BMULS 指令是一个 16 位指令，将累加器低 16 位中的数据乘以一个 4 位（最大）常数，结果可能达到 32 位，存放在累加器中。



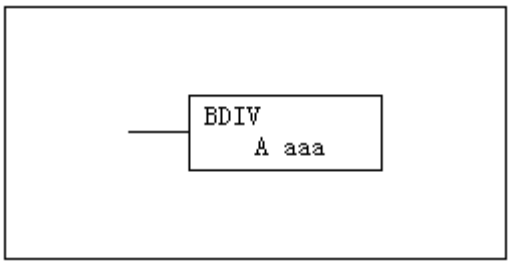
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFF

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

9.33 16 位（BIN）除法指令（BDIV）

BDIV 指令是一个 16 位指令，将累加器低 16 位中的数据除以指定寄存器（A aaa）中的二进制数，商存放在累加器中，余数存放在堆栈的第 1 级中。

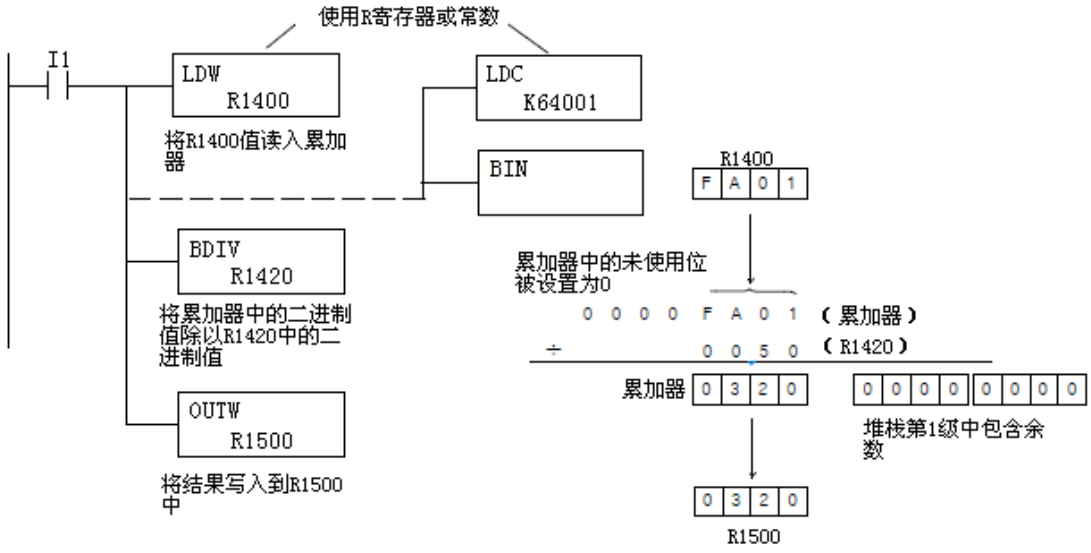


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP53	操作数超出累加器处理范围时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 装入累加器，BDIV 指令将累加器中的二进制数除以 R1420 中的二进制数，OUTD 指令将累加器中的数据写入 R1500 中。

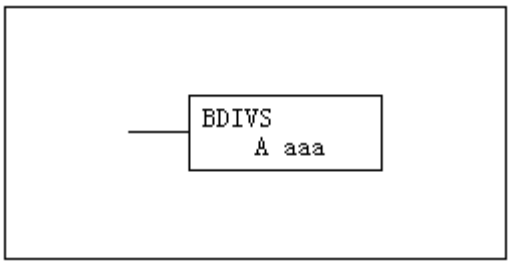


语句视图

```
LD I1
LDW R1400
BDIV R1420
OUTW R1500
```

9.34 16 位（BIN）常数除法指令（BDIVS）

BDIVS 指令是一个 16 位指令，将累加器低 16 位中的数据除以一个 4 位（最大）二进制常数，结果存放在累加器中。



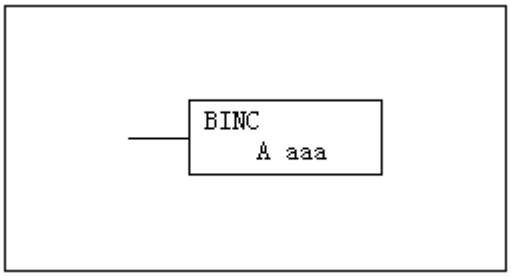
操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFF

受影响的标志线圈	描述
SP53	操作数超出累加器处理范围时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

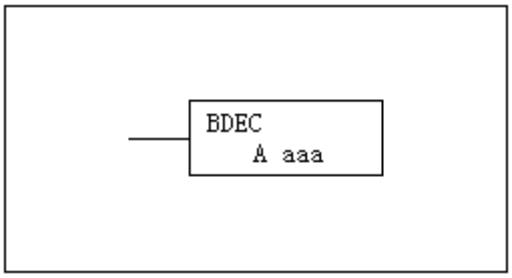
9.35 BIN 加 1 指令（BINC）

BINC 指令对由操作数指定的寄存器中存储的二进制数据进行加“1”运算，指令每执行一次，就进行一次加“1”运算。



9.36 BIN 减 1 指令（BDEC）

BDEC 指令对由操作数指定的寄存器中存储的二进制数据进行减“1”运算，指令每执行一次，就进行一次减“1”运算。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，每当 M5 为 ON 时，R1400 中的数据就增加 1。

语句视图

LD M5

BINC R1400

下面的例子中，每当 M5 为 ON 时，R1400 中的数据就减少 1。

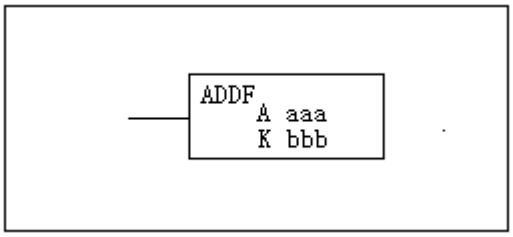
语句视图

LD M5

BDEC R1400

9.37 任意位长（BCD）加法指令（ADDF）

ADDF 指令是一个 32 位指令，将累加器中的 BCD 数同一个指定位数的 BCD 数相加。指定的位数范围是 1-32 个连续的位，结果存放在累加器中。

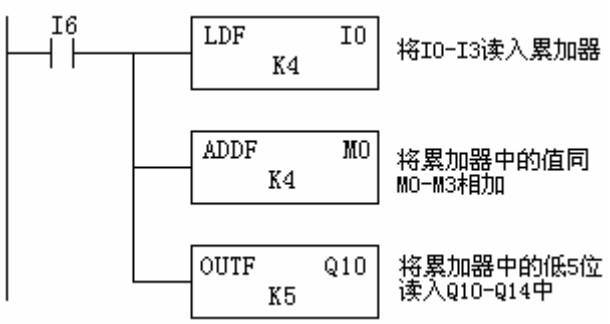


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

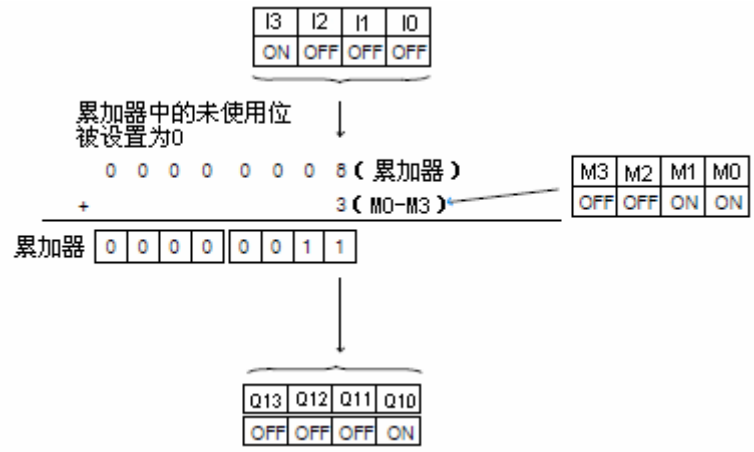
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，ADDF 指令将累加器中的数据同 M0-M3 相加，结果存放在累加器中，OUTF 指令将累加器的低 5 位写入 Q10-Q14 中。



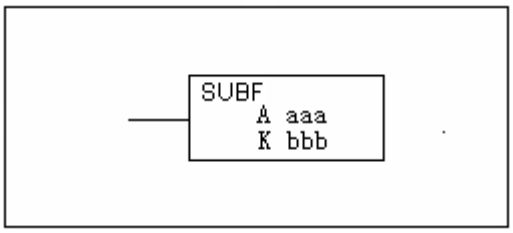
语句视图

```
LD I6
LDF I0 K4
ADDF M0 K4
OUTF Q10 K5
```



9.38 任意位长（BCD）减法指令（SUBF）

SUBF 指令是一个 32 位指令，将累加器中的 BCD 数减去一个指定位数的 BCD 数。指定的位数范围是 1-32 个连续的位,结果存放在累加器中。

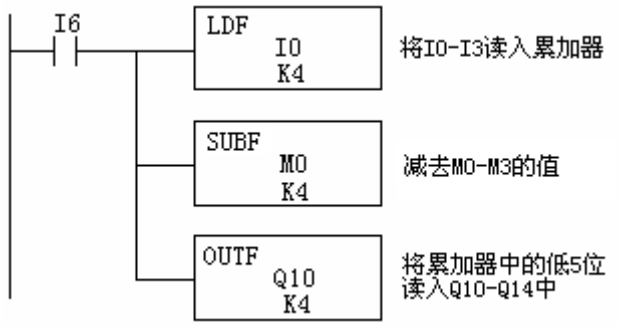


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

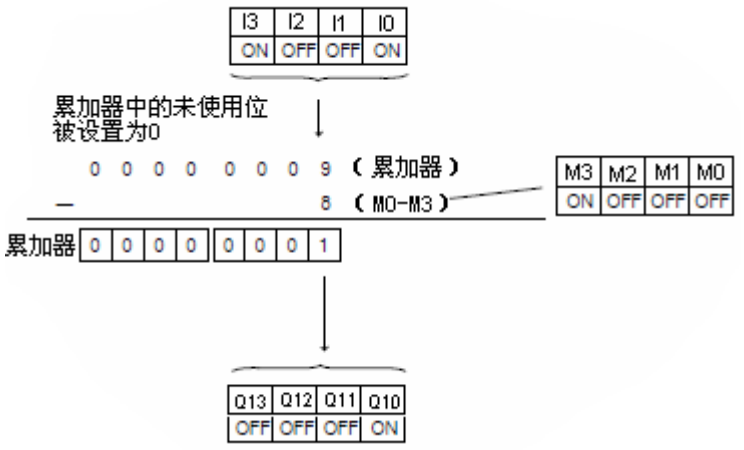
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，SUBF 指令将累加器中的数据减去 M0-M3 的值，结果存放在累加器中，OUTF 指令将累加器的低 4 位写入 Q10-Q13 中。



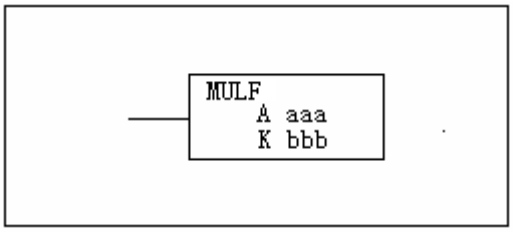
语句视图

```
LD I6
LDF I0 K4
SUBF M0 K4
OUTF Q10 K4
```



9.39 任意位长（BCD）乘法指令（MULF）

MULF 指令是一个 16 位指令，将累加器中的 BCD 数乘以一个指定位数的 BCD 数。指定的位数范围是 1-16 个连续的位，结果存放在累加器中。

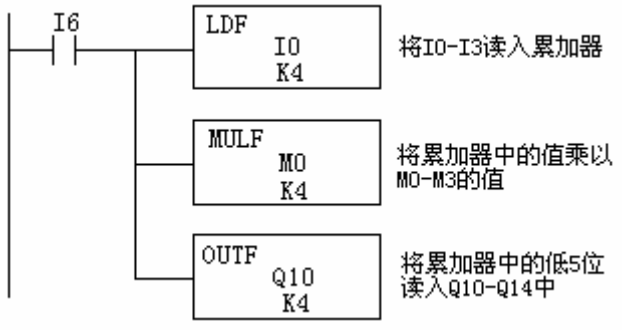


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-16

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

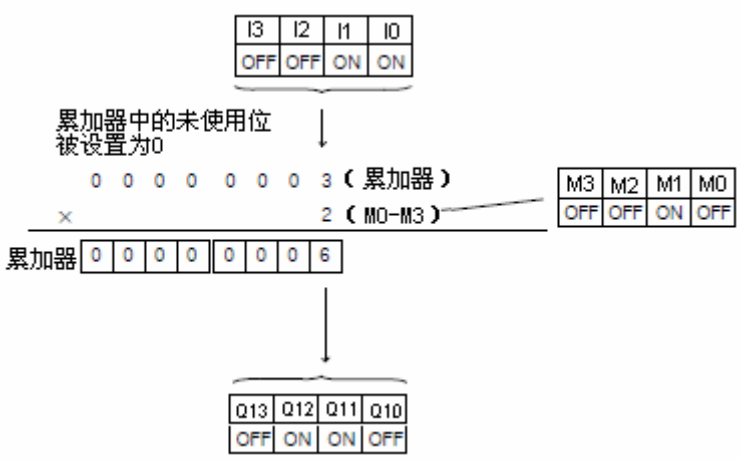
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，MULF 指令将累加器中的数据乘以 M0-M3 的值，结果存放在累加器中，OUTF 指令将累加器的低 4 位写入 Q10-Q13 中。



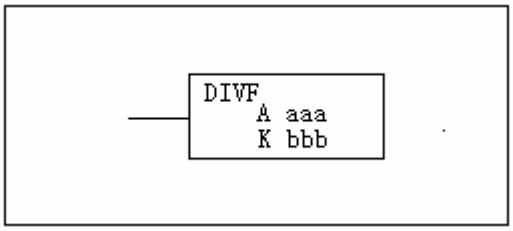
语句视图

```
LD I6
LDF I0 K4
MULF M0 K4
OUTF Q10 K4
```



9.40 任意位长（BCD）除法指令（DIVF）

DIVF 指令是一个 16 位指令，将累加器中的 BCD 数除以一个指定位数的 BCD 数。指定的位数范围是 1-16 个连续的位，商存放在累加器中，余数存放在数据堆栈的第 1 级中。

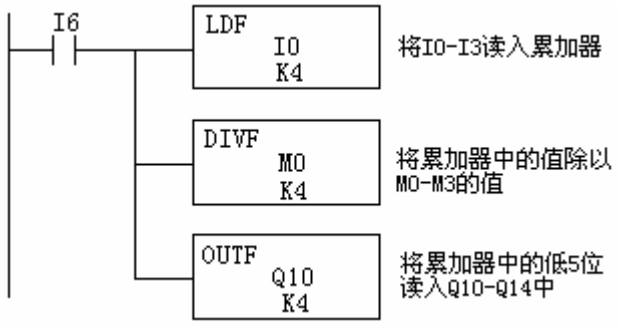


操作数类型		D4-454 范围	
	A	aaa	bbb
输入	I	0-1777	--
输出	Q	0-1777	--
中间继电器	M	0-3777	--
级	S	0-1777	--
定时器	T	0-377	--
计数器	C	0-377	--
特殊线圈	SP	0-777	--
通讯输入	GI	0-3777	--
通讯输出	GQ	0-3777	--
常数	K	--	1-16

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

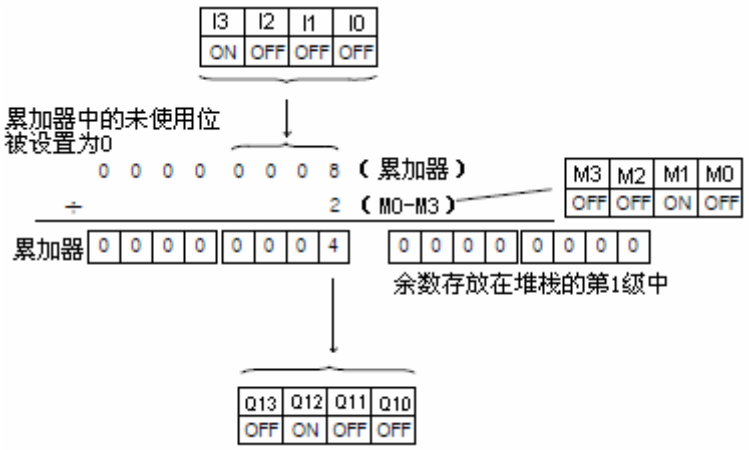
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I6 为 ON 时，LDF 指令将 I0-I3 读入累加器，DIVF 指令将累加器中的数据除以 M0-M3 的值，结果存放在累加器中，OUTF 指令将累加器的低 4 位写入 Q10-Q13 中。



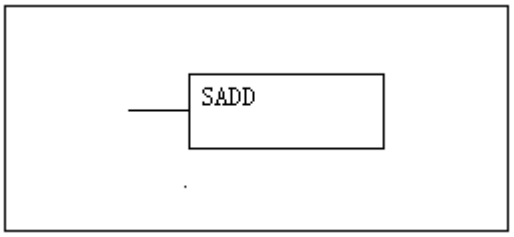
语句视图

```
LD I6
LDF I0 K4
DIVF M0 K4
OUTF Q10 K4
```



9.41 堆栈加法指令（SADD）

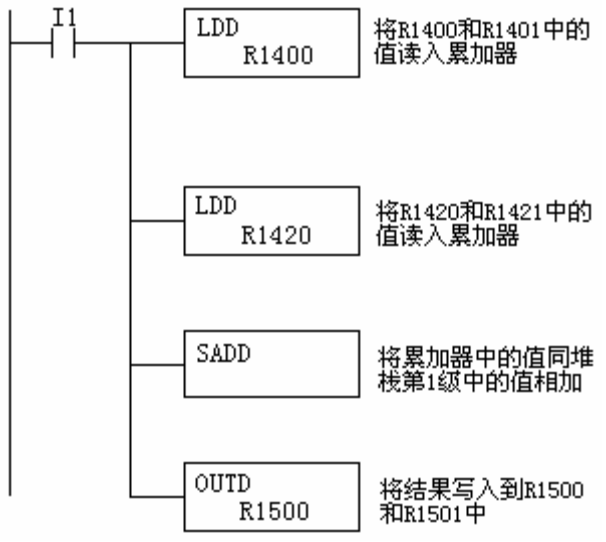
SADD 指令是一个 32 位指令，将累加器中的 BCD 数加上数据堆栈第 1 级中的 BCD 数，结果存放在累加器中。堆栈第 1 级中的数据移出，其他级均向上移动一级。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

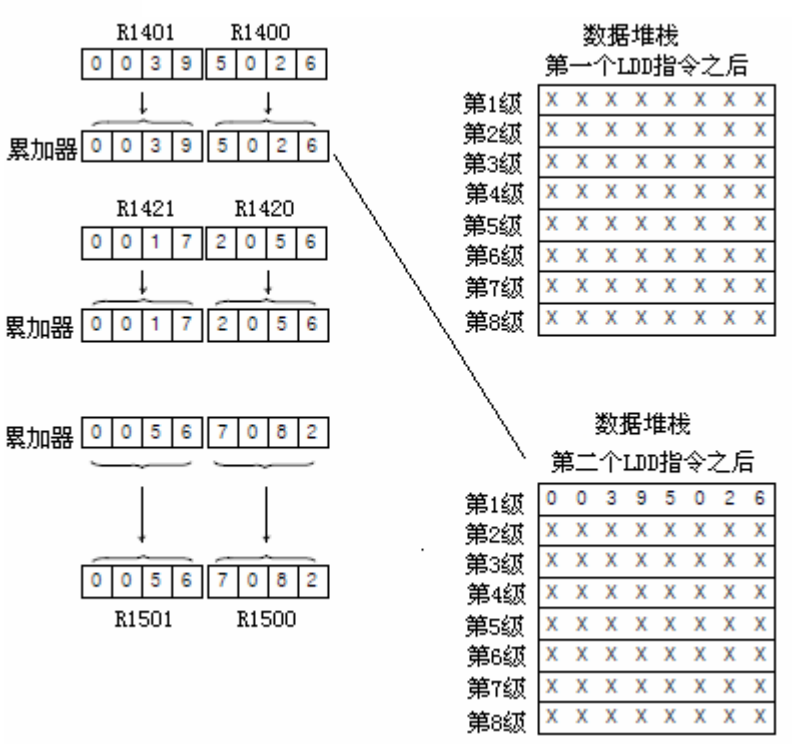
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SADD 指令将累加器中的数据加上堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



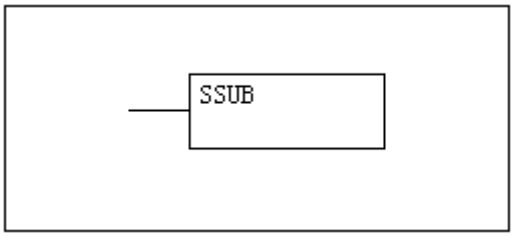
语句视图

```
LD I1
LDD R1400
LDD R1420
SADD
OUTD R1500
```



9.42 堆栈减法指令（SSUB）

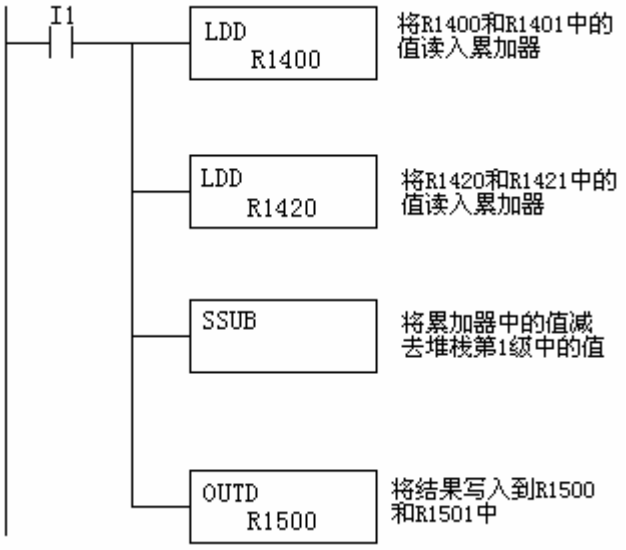
SSUB 指令是一个 32 位指令，将累加器中的 BCD 数减去数据堆栈第 1 级中的 BCD 数，结果存放在累加器中。堆栈第 1 级中的数据移出，其他级均向上移动一级。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

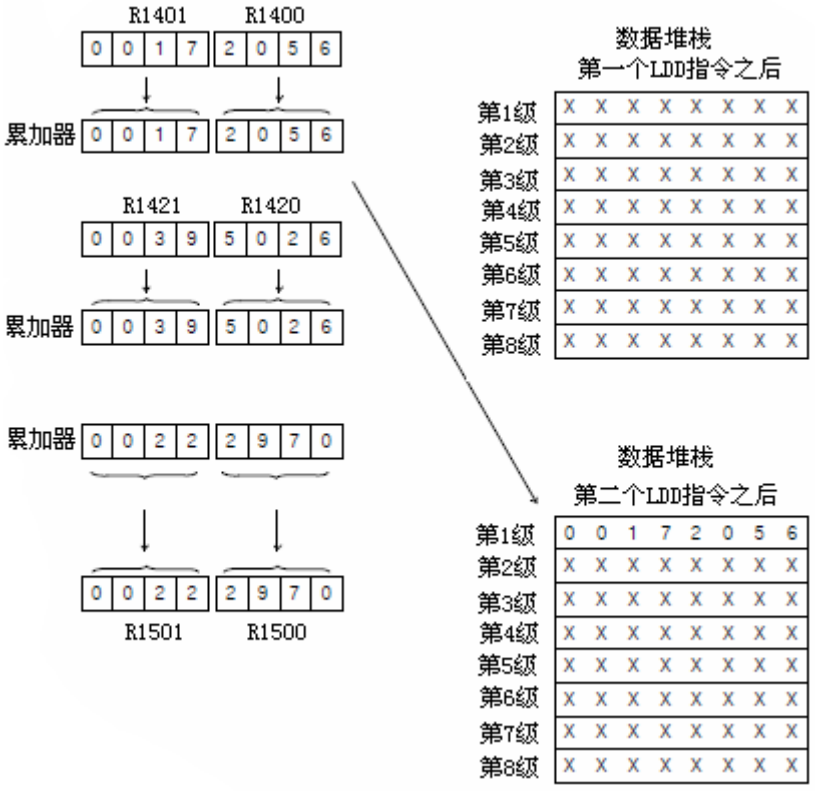
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SSUB 指令将累加器中的数据减去堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



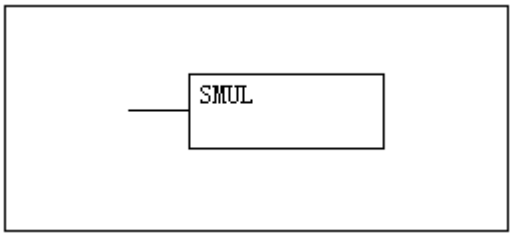
语句视图

```
LD I1
LDD R1400
LDD R1420
SSUB
OUTD R1500
```



9.43 堆栈乘法指令（SMUL）

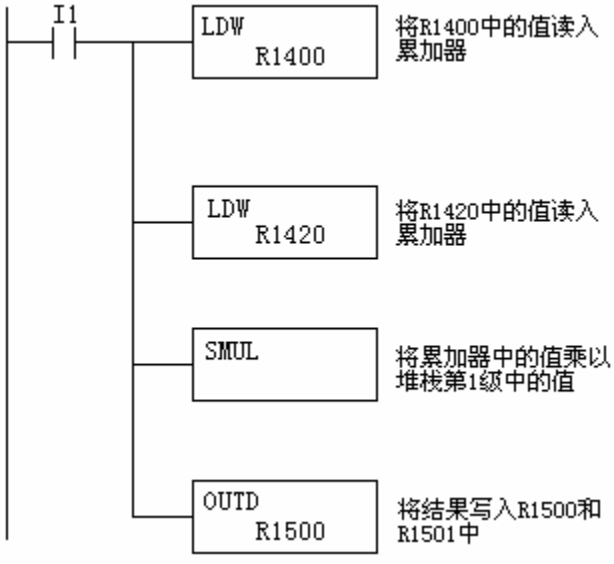
SMUL 指令是一个 16 位指令，将累加器中的 4 位 BCD 数乘以数据堆栈第 1 级中的 4 位 BCD 数，结果存放在累加器中。堆栈第 1 级中的数据移出，其他级均向上移动一级。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

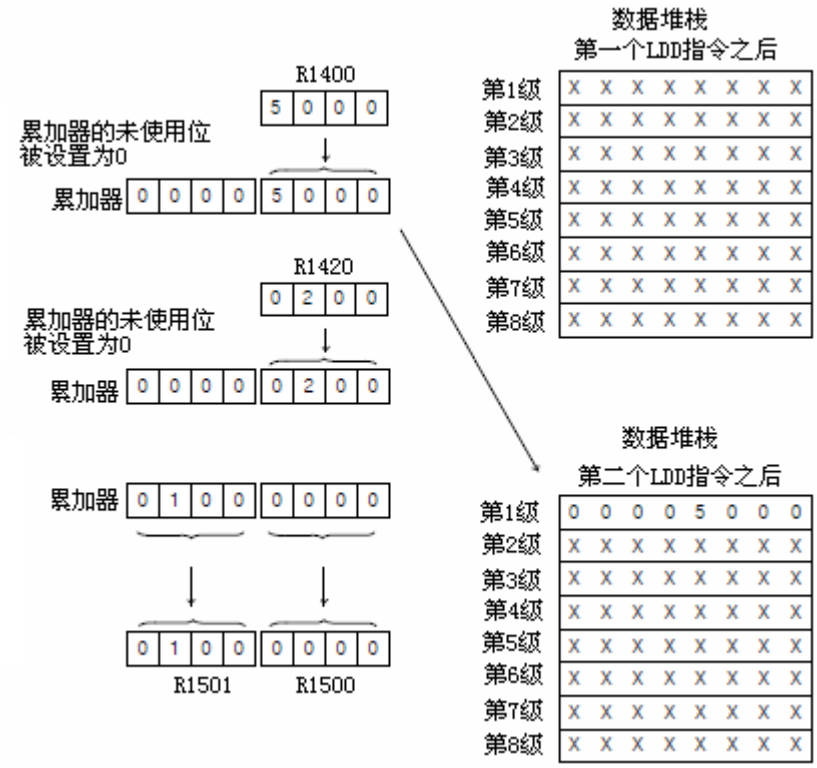
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，第一个 LDW 指令将 R1400 中的数据读入累加器，第二个 LDW 指令将 R1420 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SMUL 指令将累加器中的数据乘以堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



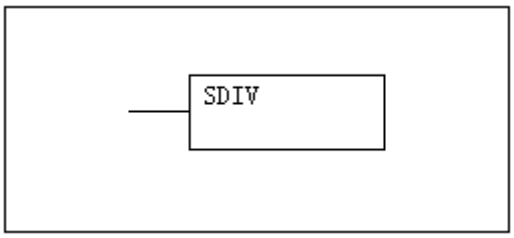
语句视图

```
LD I1
LDW R1400
LDW R1420
SMUL
OUTD R1500
```



9.44 堆栈除法指令（SDIV）

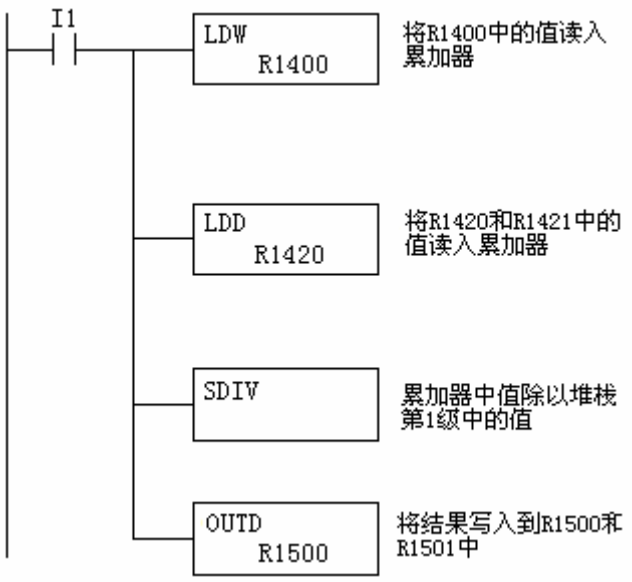
SDIV 指令是一个 32 位指令，将累加器中的 8 位 BCD 数除以数据堆栈第 1 级中的 4 位 BCD 数，商存放在累加器中，余数存放在堆栈第 1 级中。



受影响的标志线圈	描述
SP53	操作数里的数据大于累加器中的数据以致不能运算时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

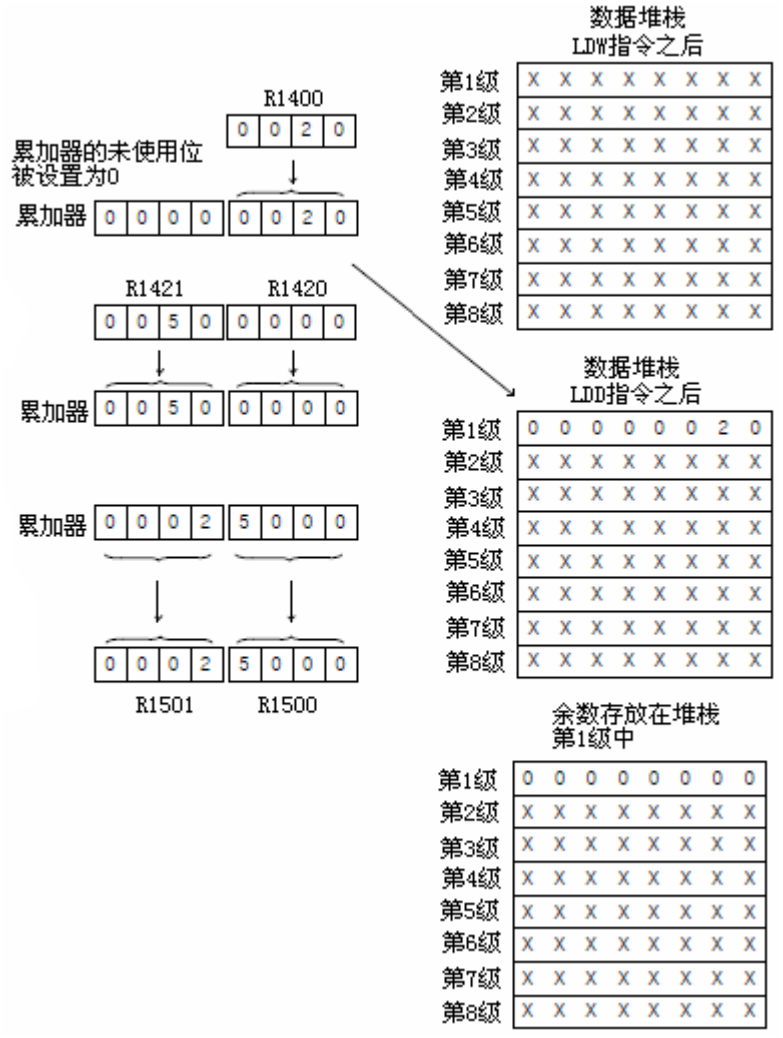
注意：只有在其它使用相同标志线圈的指令执行前，状态标志才有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器中的原有数据被压入数据堆栈的第 1 级中。SDIV 指令将累加器中的数据除以堆栈第 1 级中的数据，结果写入到 R1500 和 R1501 中。



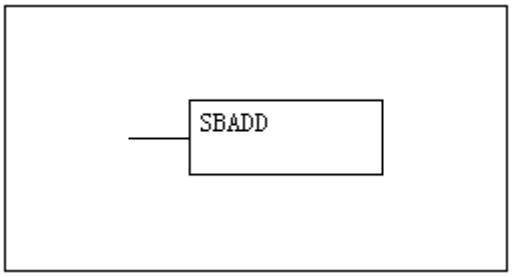
语句视图

```
LD I1
LDW R1400
LDD R1420
SDIV
OUTD R1500
```



9.45 堆栈二进制加法指令（SBADD）

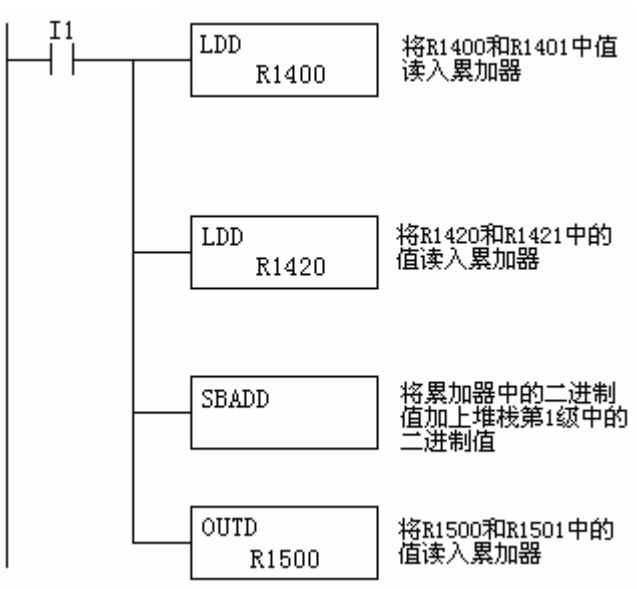
SBADD 指令是一个 32 位指令，将累加器中的二进制数加上堆栈第 1 级中的二进制数，结果存放在累加器中。堆栈第 1 级的数据移出，其他级均向上移动一级。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP66	半进位标记，当运算结果的第 15 位向第 16 位进位时 ON
SP67	进位标记，当运算结果的第 31 位进位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

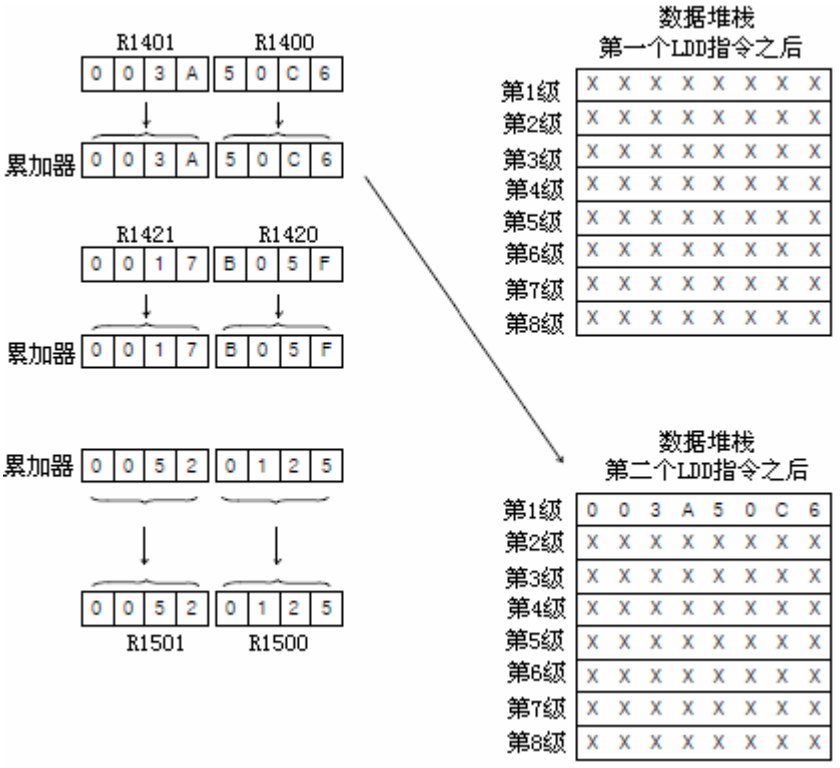
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBADD 指令将累加器中的二进制数据加上堆栈第 1 级中的二进制数据，结果写入到 R1500 和 R1501 中。



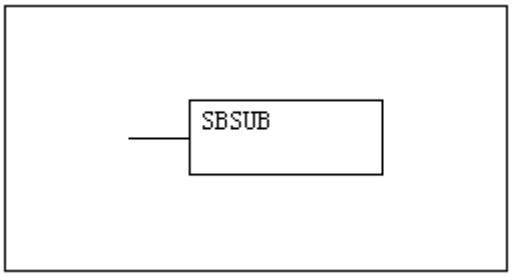
语句视图

```
LD I1
LDD R1400
LDD R1420
SBADD
OUTD R1500
```



9.46 堆栈二进制减法指令（SBSUB）

SBSUB 指令是一个 32 位指令，将累加器中的二进制数据减去堆栈第 1 级中的二进制数据，结果存放在累加器中。堆栈第 1 级的数据移出，其他级均向上移动一级。



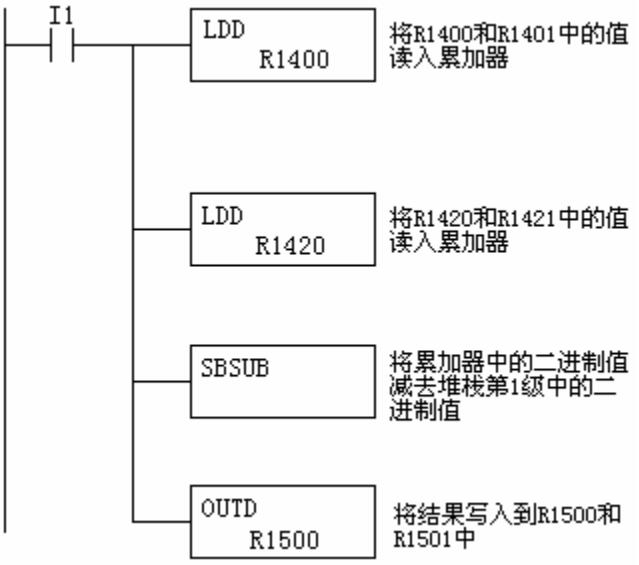
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP64	半借位标记，当运算结果的第 15 位向第 16 位借位时 ON
SP65	借位标记，当运算结果的第 31 位向第 32 位借位时 ON
SP70	累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

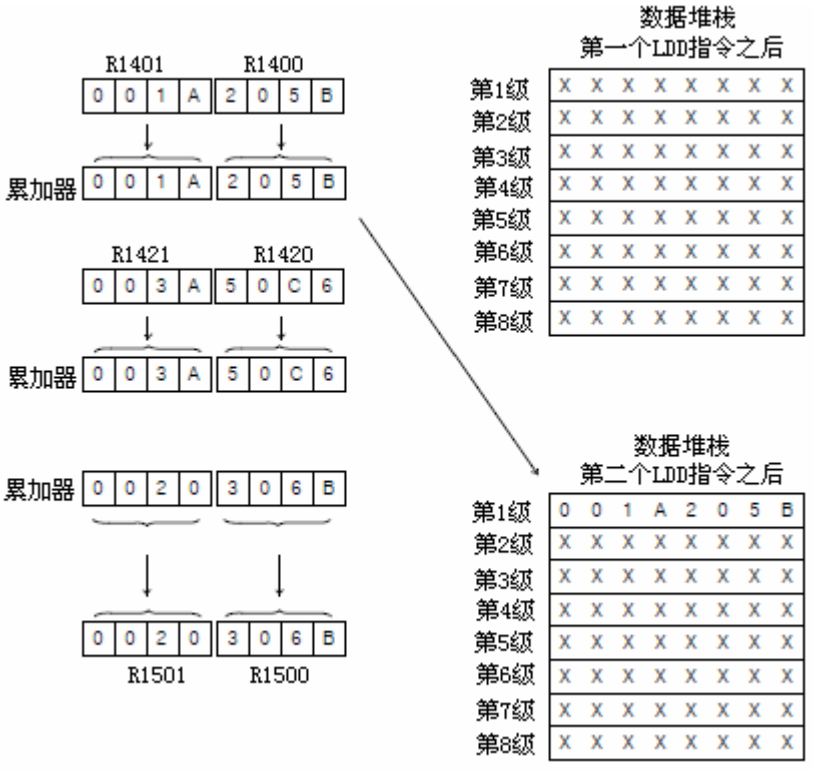
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，第一个 LDD 指令将 R1400 和 R1401 中的数据读入累加器，第二个 LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBSUB 指令将累加器中的二进制数据减去堆栈第 1 级中的二进制数据，结果写入到 R1500 和 R1501 中。

语句视图

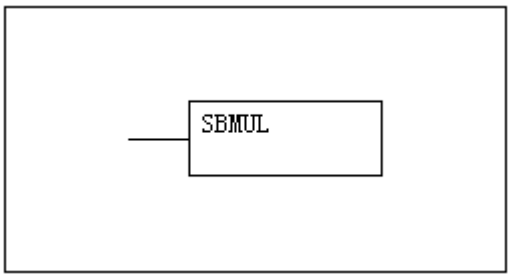
```
LD I1
LDD R1400
LDD R1420
SBSUB
OUTD R1500
```





9.47 堆栈二进制乘法指令（SBMUL）

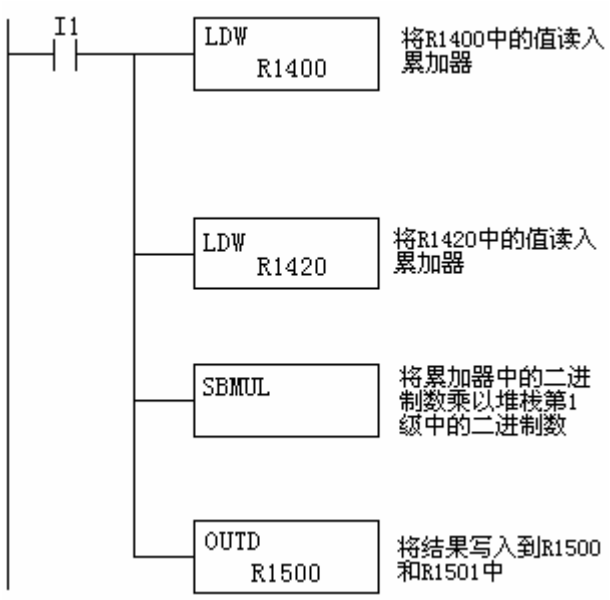
SBMUL 指令是一个 16 位指令，将累加器中的二进制数据乘以堆栈第 1 级中的二进制数据，结果可能是 32 位（最大），存放在累加器中。堆栈第 1 级的数据移出，其他级均向上移动一级。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

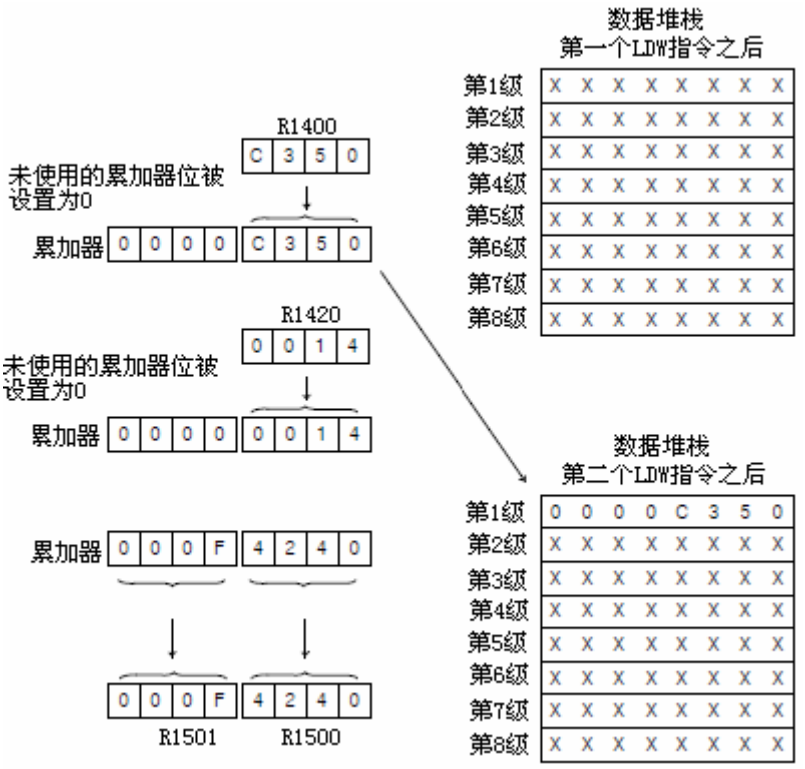
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，第一个 LDW 指令将 R1400 中的数据读入累加器，第二个 LDW 指令将 R1420 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBMUL 指令将累加器中的二进制数据乘以堆栈第 1 级中的二进制数据，结果写入到 R1500 和 R1501 中。



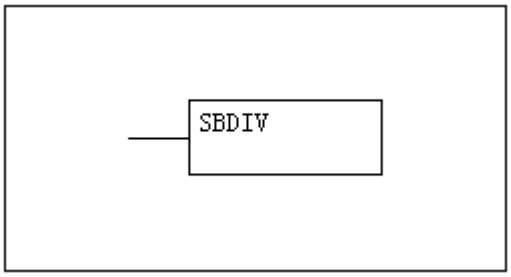
语句视图

```
LD I1
LDW R1400
LDW R1420
SBMUL
OUTD R1500
```



9.48 堆栈二进制除法指令（SBDIV）

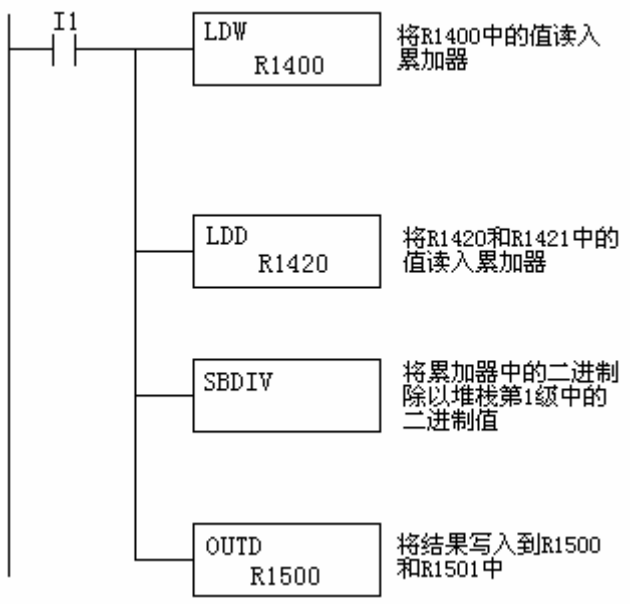
SBDIV 指令是一个 32 位指令，将累加器中的 32 位二进制数据除以堆栈第 1 级中的 16 位二进制数据，商存放在累加器中，余数存放在堆栈第 1 级中。



受影响的标志线圈	描述
SP53	操作数里的数据大于累加器中的数据以致不能运算时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

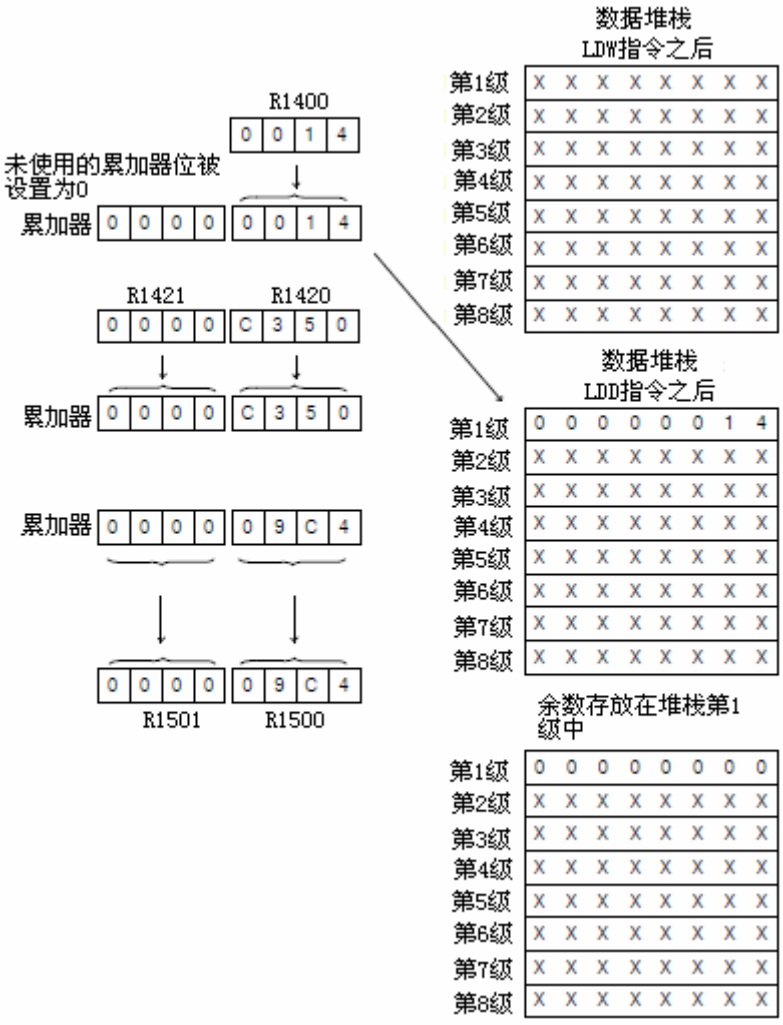
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，LDD 指令将 R1420 和 R1421 中的数据读入累加器，累加器原有数据被压入数据堆栈。SBDIV 指令将累加器中的二进制数除以堆栈第 1 级中的二进制数，结果写入到 R1500 和 R1501 中。



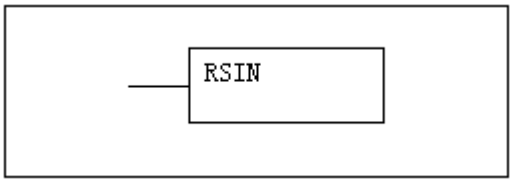
语句视图

```
LD I1
LDW R1400
LDD R1420
SBDIV
OUTD R1500
```



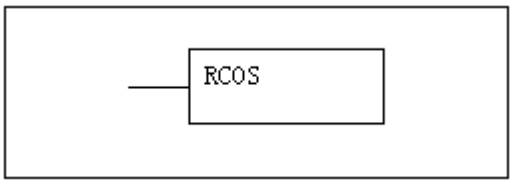
9.49 实数正弦指令（RSIN）

RSIN 指令求出累加器中实数的正弦函数值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



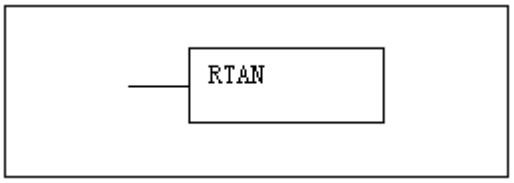
9.50 实数余弦指令（RCOS）

RCOS 指令求出累加器中实数的余弦函数值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



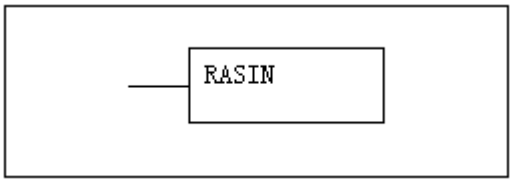
9.51 实数正切指令（RTAN）

RTAN 指令求出累加器中实数的正切函数值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



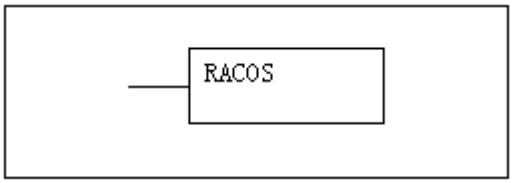
9.52 实数反正弦指令（RASIN）

RASIN 指令将累加器中的正弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



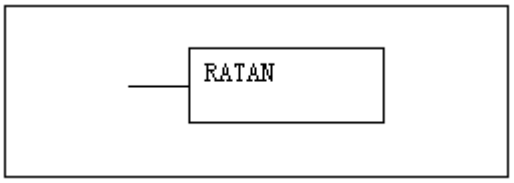
9.53 实数反余弦指令（RACOS）

RACOS 指令将累加器中的余弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



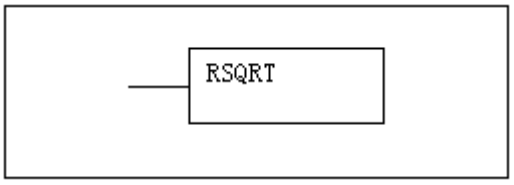
9.54 实数反正切指令（RATAN）

RATAN 指令将累加器中的正切函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



9.55 实数平方根指令（RSQRT）

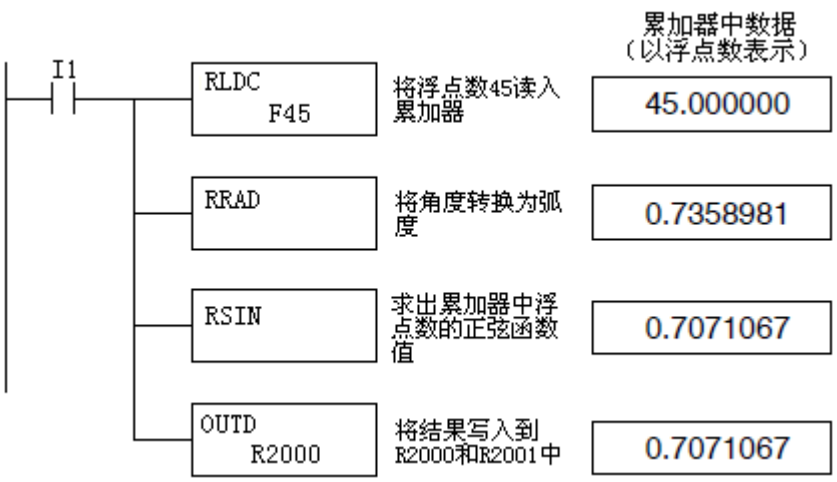
RSQRT 指令对累加器中的实数求平方根，结果存放在累加器中。原始数据和运算结果都应是 IEEE 32 位格式。



受影响的标志线圈	描述
SP53	操作数超出累加器处理范围时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

注意：平方根指令在有些场合很有用，比如将孔板测得的流量数据求取平方根用于 PID 回路的 PV 值，但是 PID 回路内置平方根功能，如果勾选，就不需再用平方根指令。

下面的例子中，求出 45 度的正弦函数值，由于这些高级功能指令的操作数是实数，因此使用 RLDC 指令读入实数。三角函数指令的运算对象须是弧度，因此用 RRAD 指令将角度转化为弧度，RSIN 指令求出正弦函数值，OUTD 指令将结果写入到 R2000 和 R2001 中，由于结果是 32 位的长度，因此需要双字节输出指令。

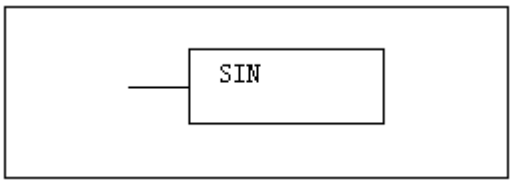


语句视图

```
LD  I1
RLDC  F45
RRAD
RSIN
OUTD  R2000
```

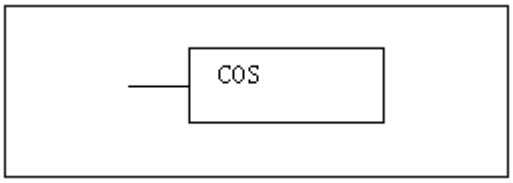
9.56 BCD 数正弦指令（SIN）

SIN 指令求出累加器中 BCD 数的正弦函数值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



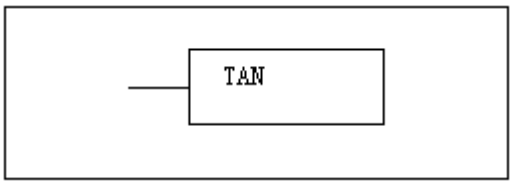
9.57 BCD 数余弦指令（COS）

COS 指令求出累加器中 BCD 数的余弦函数值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



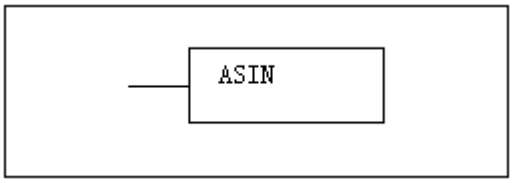
9.58 BCD 数正切指令（TAN）

TAN 指令求出累加器中 BCD 数的正切函数值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



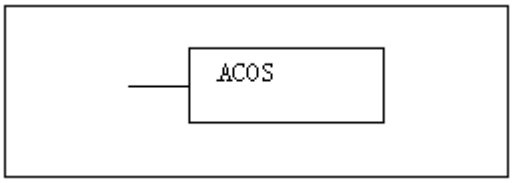
9.59 BCD 数反正弦指令（ASIN）

ASIN 指令将累加器中的正弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



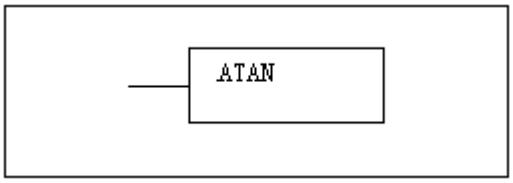
9.60 BCD 数反余弦指令（ACOS）

ACOS 指令将累加器中的余弦函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



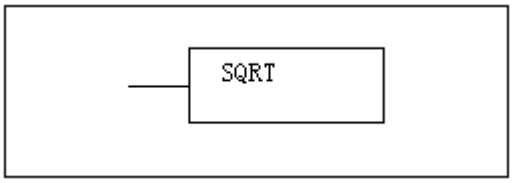
9.61 BCD 数反正切指令（ATAN）

ATAN 指令将累加器中的正切函数值转换成角度值，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。



9.62 BCD 数平方根指令（SQRT）

SQRT 指令对累加器中的 BCD 数求平方根，结果存放在累加器中。原始数据和运算结果都应是 BCD 格式。

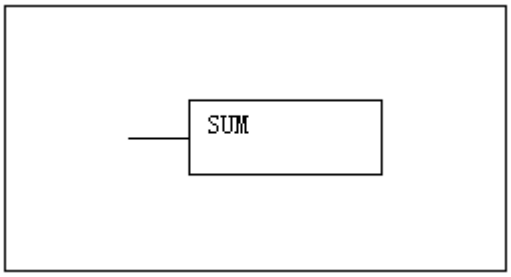


受影响的标志线圈	描述
SP53	操作数超出累加器处理范围时 ON
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

第 10 章 位操作指令

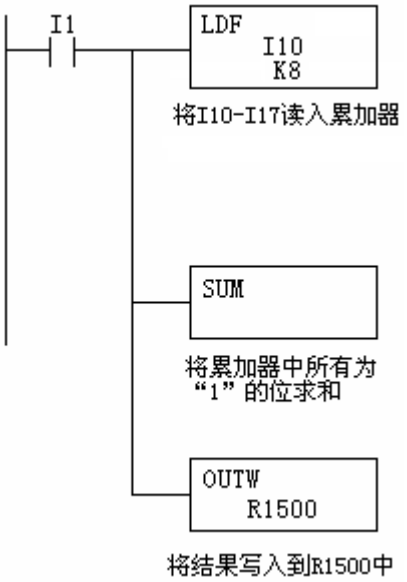
10.1 ON 位求和指令（SUM）

求出累加器中所有为“1”（ON）的位的总数，结果以十六进制形式存放在累加器中。



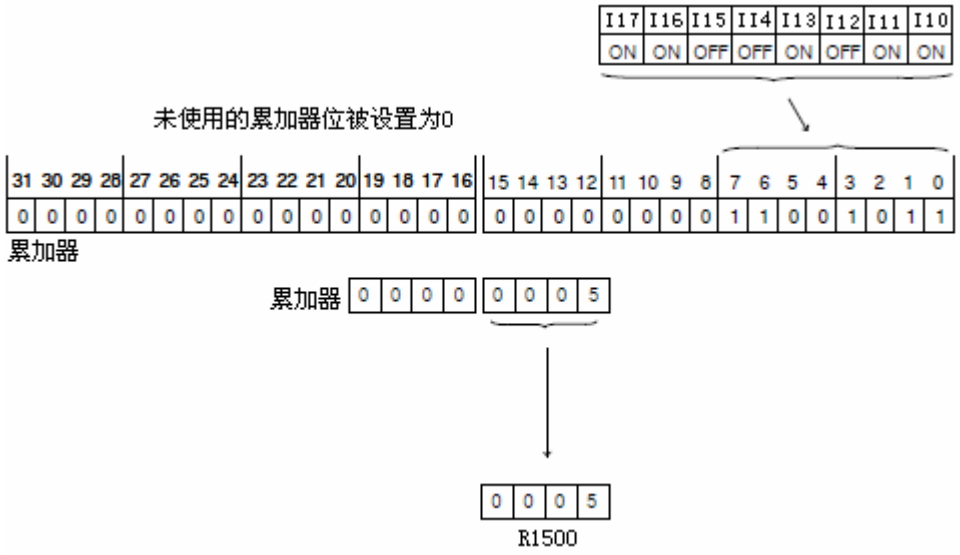
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON

下面的例子中，当 I1 为 ON 时，LDF 指令将 I10-I17 读入累加器，SUM 指令将累加器中所有为“1”的位求和，结果存放在累加器中，OUTW 指令将结果写入到 R1500 中。



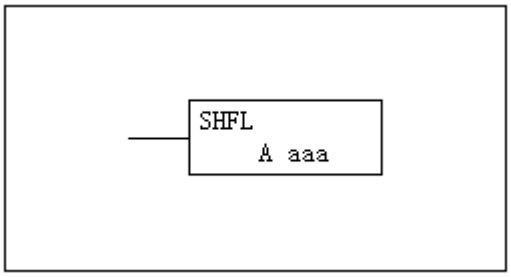
语句视图

```
LD  I1
LDF  I10  K8
SUM
OUTW  R1500
```



10.2 左移指令（SHFL）

SHFL 指令是一个 32 位指令，将累加器中的数据向左移位，移动位数由指令指定，移动后空出的位以“0”填充，移出的位丢弃。

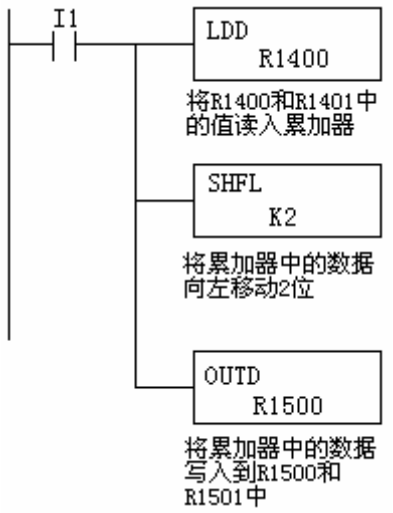


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

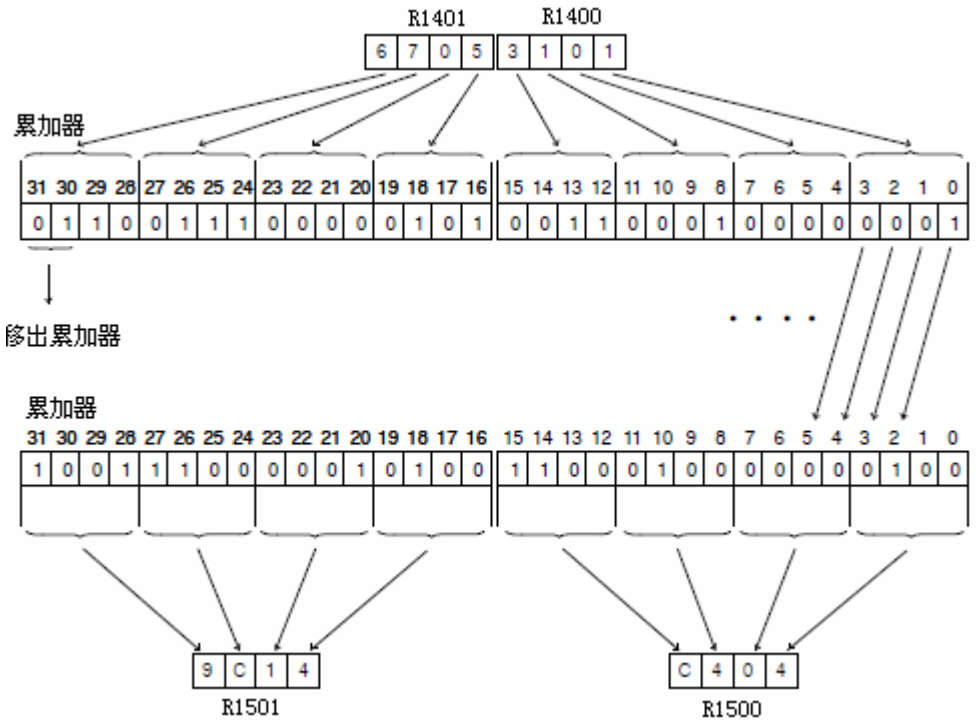
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。SHFL 指令将累加器中的数据向左移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。



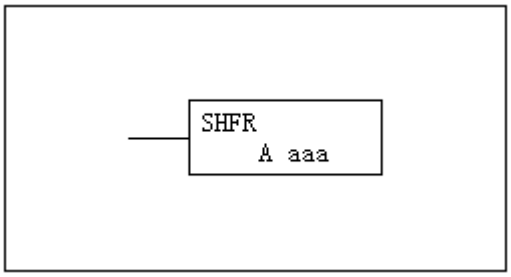
语句视图

```
LD I1
LDD R1400
SHFL K2
OUTD R1500
```



10.3 右移指令（SHFR）

SHFR 指令是一个 32 位指令，将累加器中的数据向右移位，移动位数由指令指定，移动后空出的位以“0”填充，移出的位丢弃。

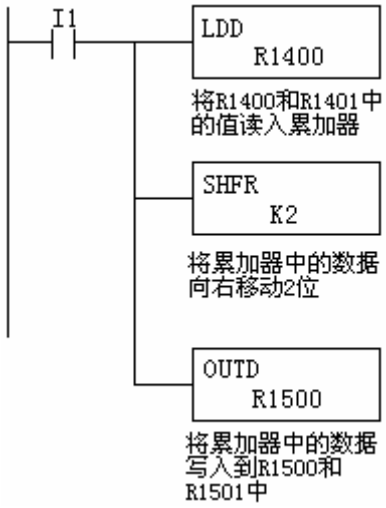


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

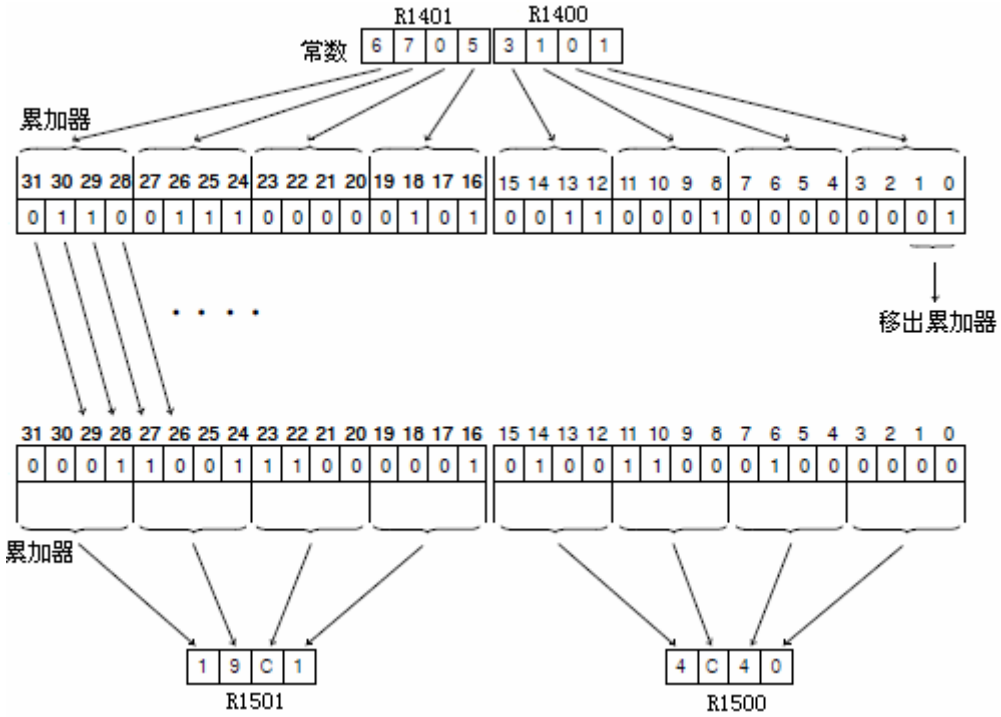
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。SHFR 指令将累加器中的数据向右移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。



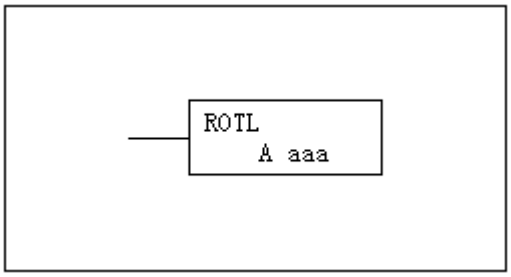
语句视图

```
LD  I1
LDD R1400
SHFR K2
OUTD R1500
```



10.4 循环左移指令（ROTL）

ROTL 指令是一个 32 位指令，将累加器中的数据循环向左移位，移动位数在指令中指定。

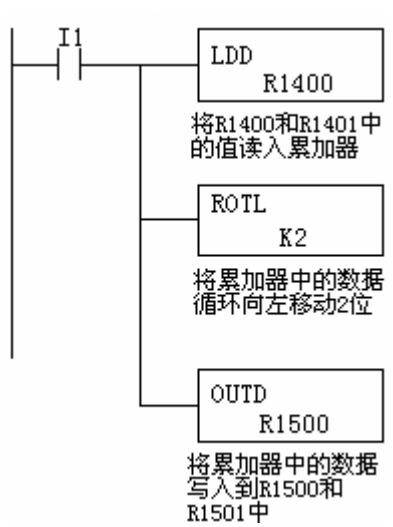


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

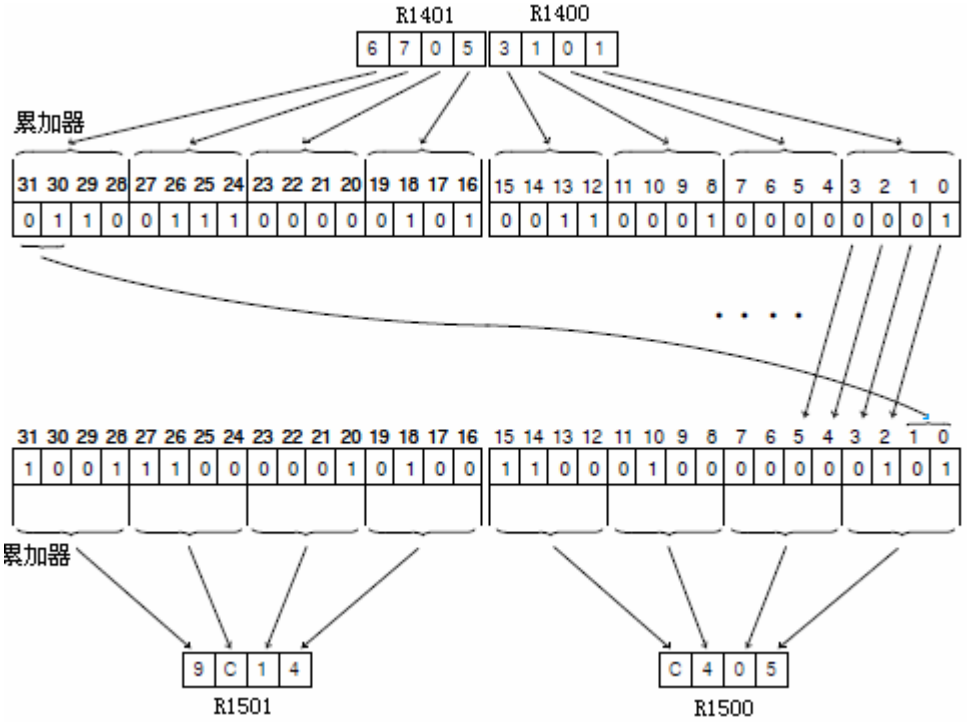
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。ROTL 指令将累加器中的数据循环向左移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。



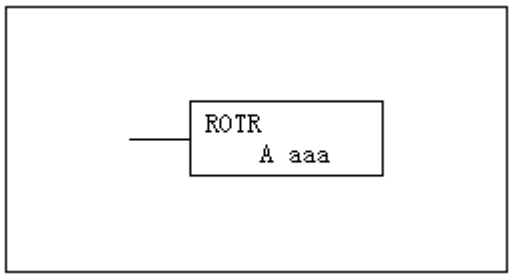
语句视图

```
LD I1
LDD R1400
ROTL K2
OUTD R1500
```



10.5 循环右移指令（ROTR）

ROTR 指令是一个 32 位指令，将累加器中的数据循环向右移位，移动位数在指令中指定。

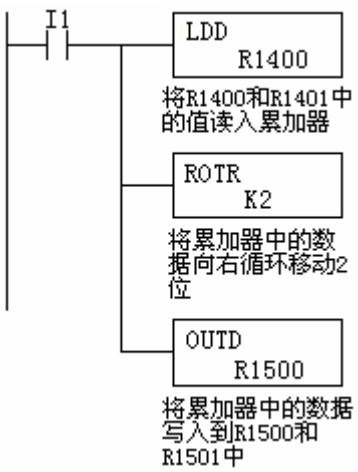


操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-32

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	累加器中的数据为负数时 ON

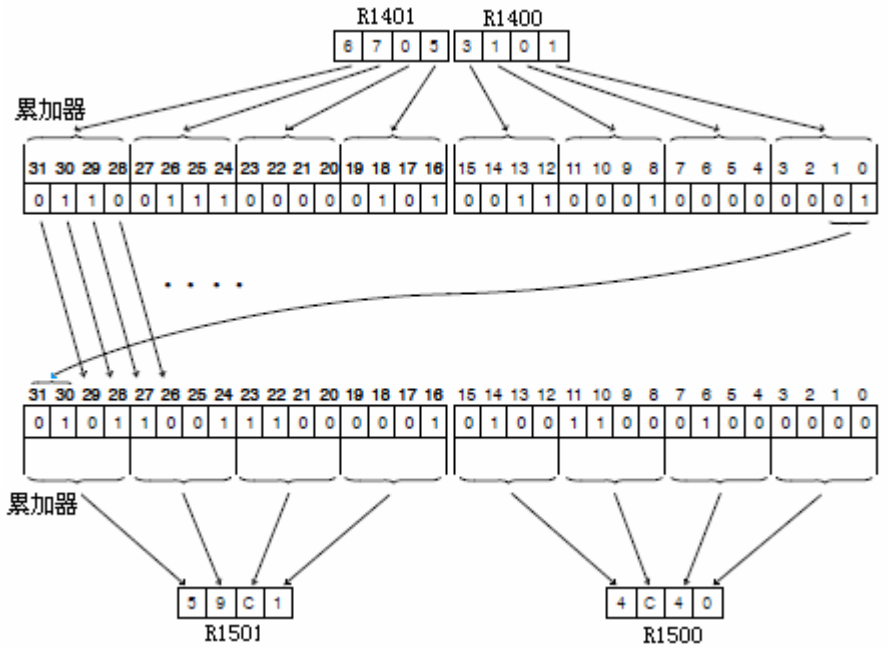
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 中的数据读入累加器。ROTR 指令将累加器中的数据循环向右移动 2 位，OUTD 指令将累加器中的数据写入到 R1500 和 R1501 中。

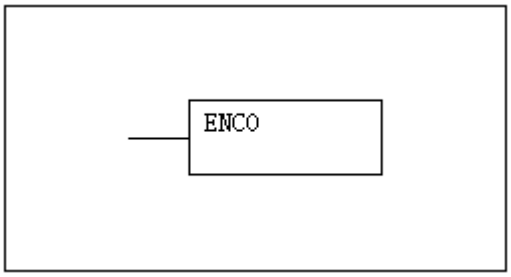


语句视图

```
LD I1
LDD R1400
ROTR K2
OUTD R1500
```



10.6 编码指令（ENCO）

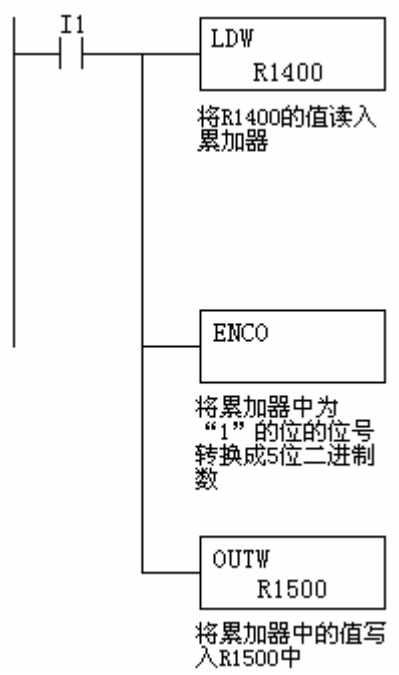


ENCO 指令将累加器中数值为“1”的一位的位号值转换成对应的 5 位二进制数。如果最高位是“1”（位号为 31），ENCO 指令就将十六进制数 1F（十进制数 31）放入累加器。如果被编码的数是 0000 或 0001，ENCO 指令就将 0 放入累加器。如果被编码的数据有多个位是“1”，则最低一个为“1”的位被编码，并且 SP53 被设置为 ON。累加器中的数据为 0 时，SP53 也被设置为 ON。

受影响的标志线圈	描述
SP53	不可进行运算处理时 ON

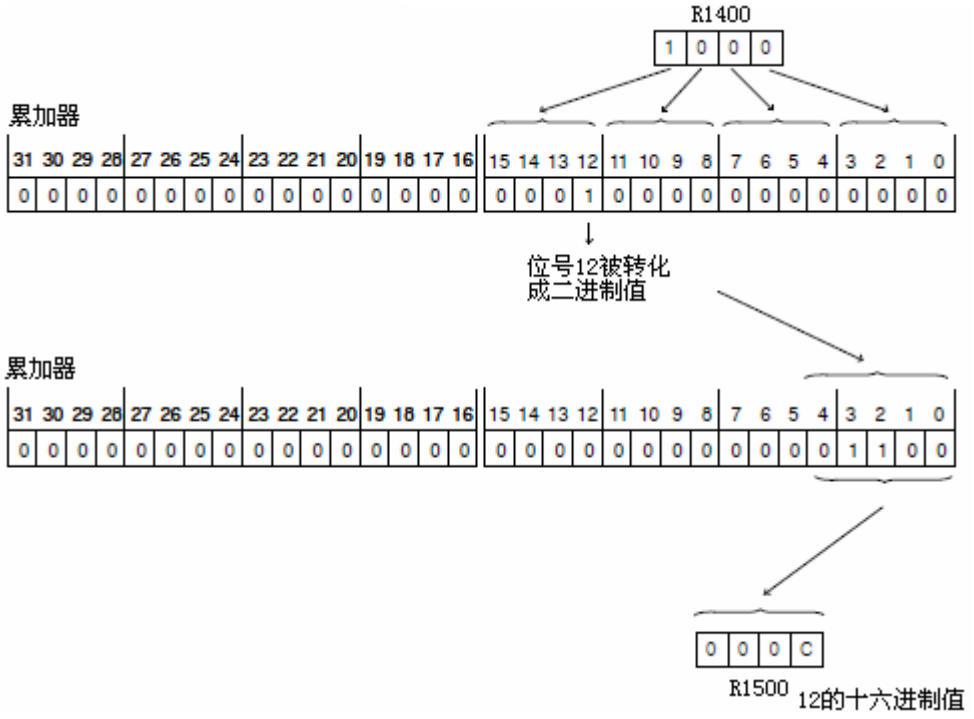
注意：在其它使用相同标志线圈的指令执行前，状态标志有效。

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器，ENCO 指令将累加器中为“1”的位的位号值转换成对应的 5 位二进制数，放入累加器中，OUTW 指令将累加器中的数据写入到 R1500 中。



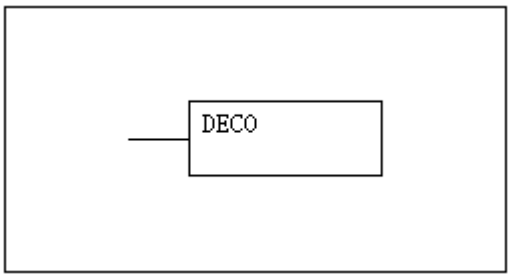
语句视图

```
LD I1
LDW R1400
ENCO
OUTW R1500
```

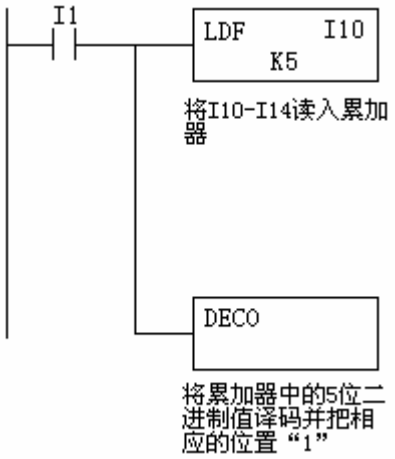


10.7 译码指令（DECO）

DECO 指令将累加器中低 5 位的二进制编码转换成 0-31 之间的数，使累加器相应的位为“1”。如果累加器中数据是 F（十六进制），位 15 将被置位 “1”。如果被译码的数据大于 31，则将其除以 32，直到其小于 32，然后再将其译码。

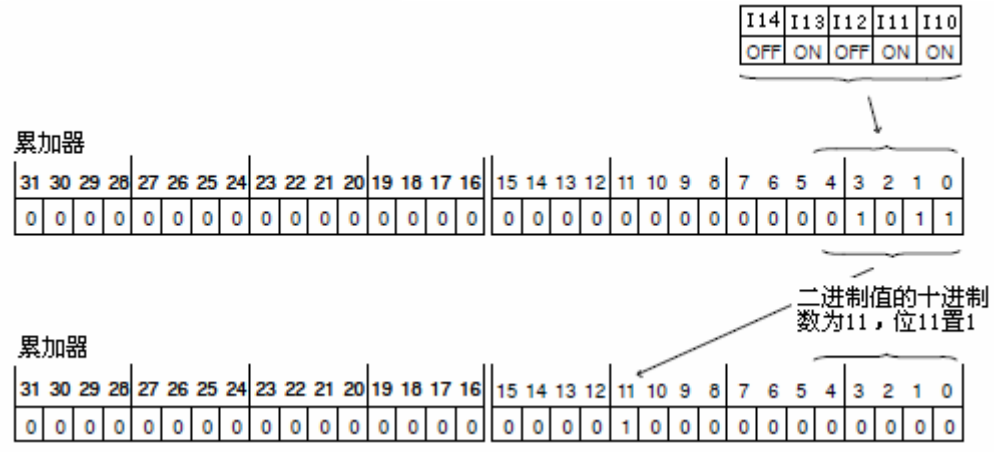


下面的例子中，当 I1 为 ON 时，LDF 指令将 I10-I14 读入累加器，DECO 指令将累加器中的 5 位二进制数据译码，将相应的位置 “1”。



语句视图

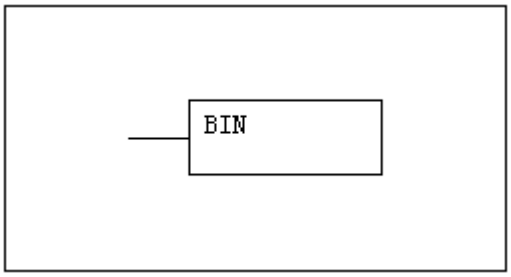
```
LD I1
LDF I10 K5
DECO
```



第 11 章 数据变换指令

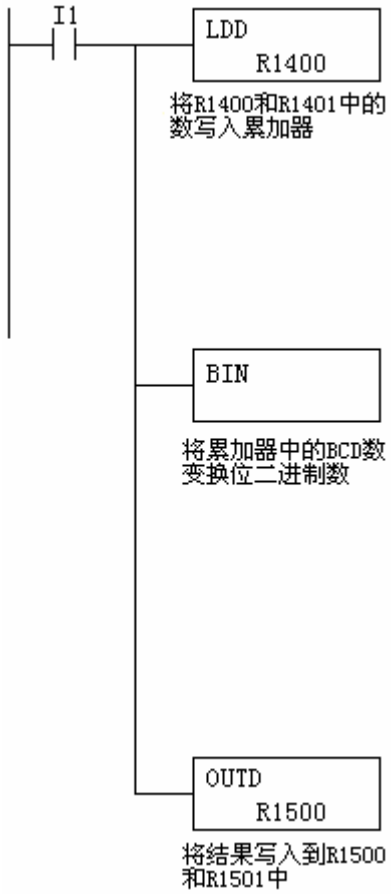
11.1 BIN 码变换指令（BIN）

BIN 指令将累加器中的 BCD 数转换成对应的二进制数，结果存放在累加器中。



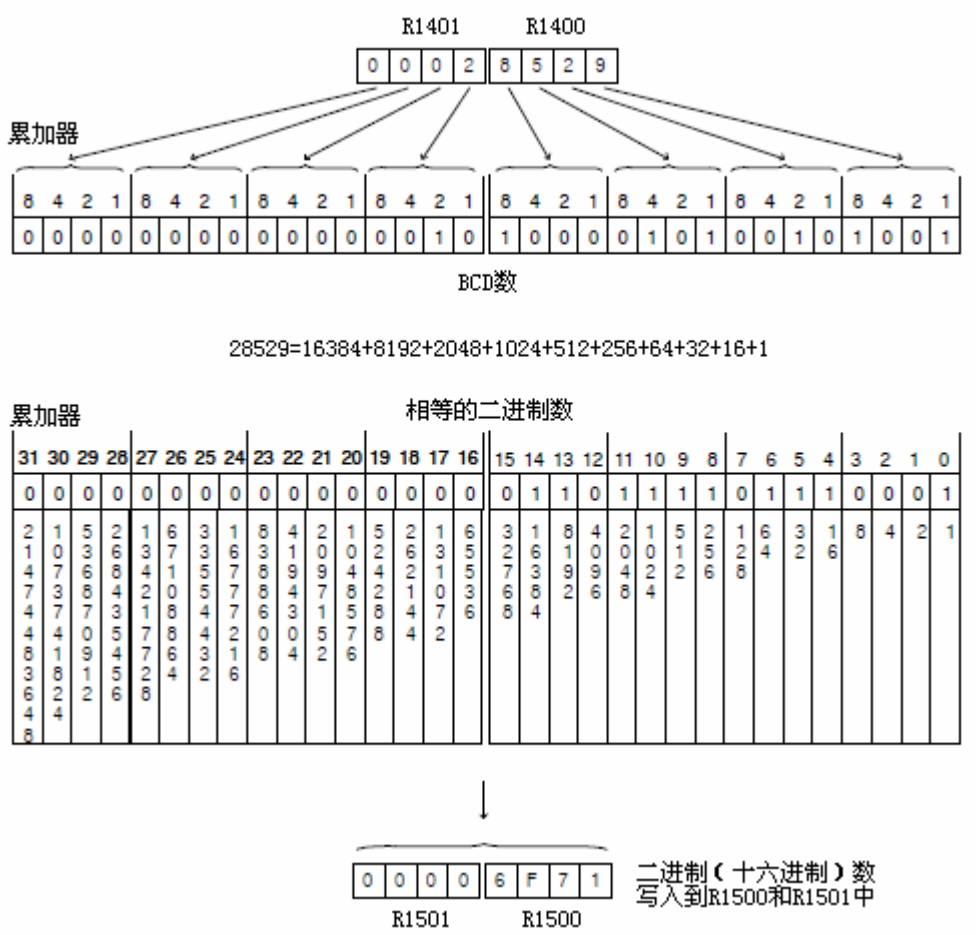
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 的数据读入累加器，BIN 指令将累加器中的数据转换成二进制数(十六进制数),OUTD 指令将结果写入到 R1500 和 R1501 中。查看 R1500 和 R1501 中的数据，显示的是十六进制数。



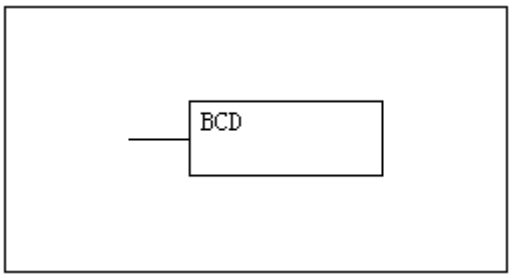
语句视图

```
LD  I1
LDD  R1400
BIN
OUTD  R1500
```


R1401
R1400
0 0 0 2
8 5 2 9
累加器
8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1
BCD数
28529=16384+8192+2048+1024+512+256+64+32+16+1
累加器
相等的二进制数
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1
2147483648 1073741824 536870912 268435456 134217728 67108864 33554432 16777216 8388608 4194304 2097152 1048576 524288 262144 131072 65536 32768 16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1
0 0 0 0 6 F 7 1
R1501 R1500
二进制（十六进制）数
写入到R1500和R1501中

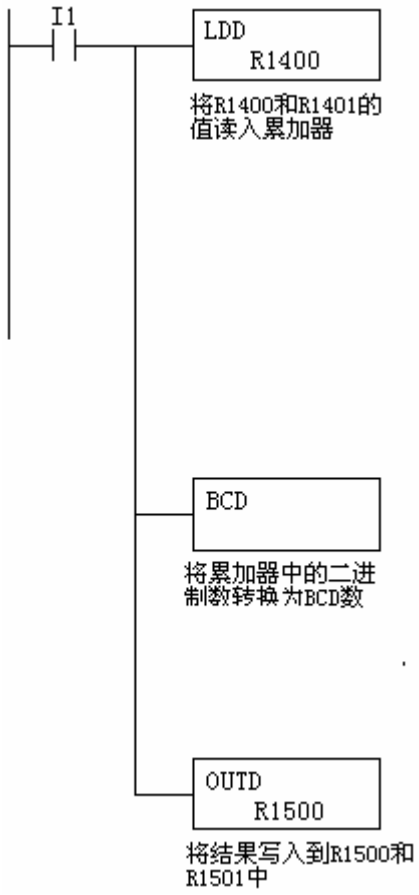
11.2 BCD 码变换指令（BCD）

BCD 指令将累加器中的二进制数转换成对应的 BCD 数，结果存放在累加器中。



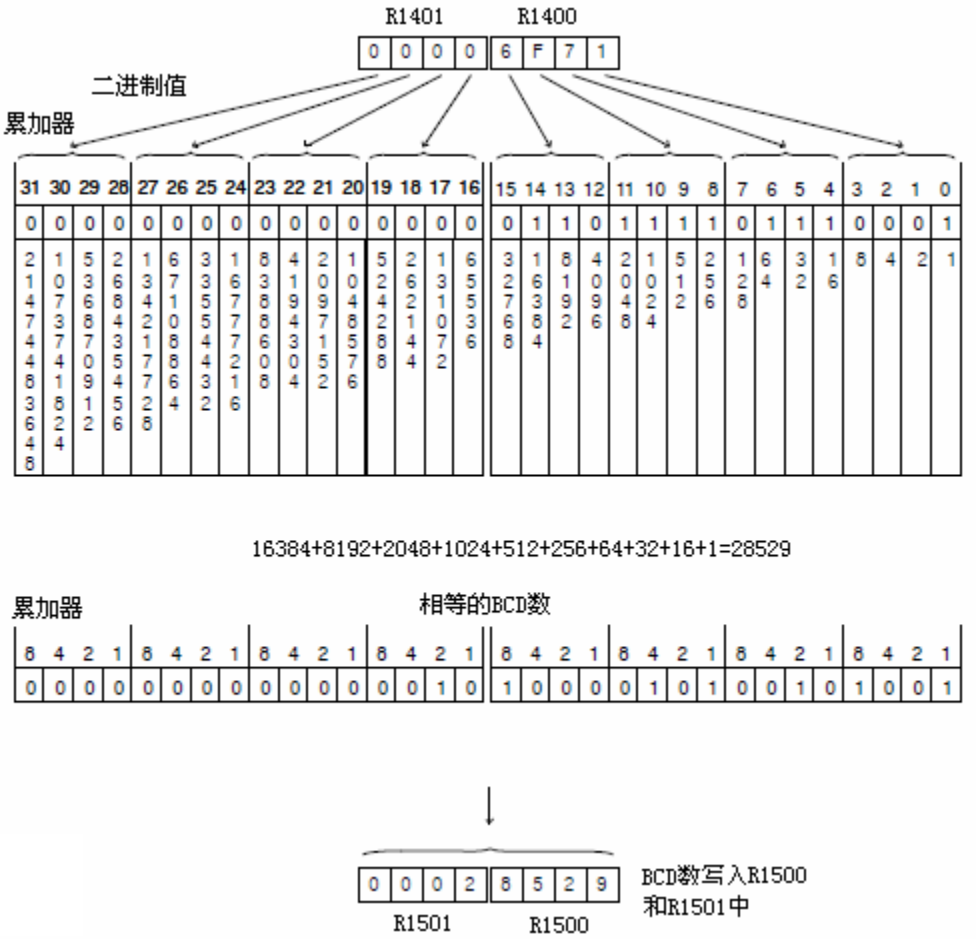
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP75	BCD 运算时，运算的结果不是 BCD 时 ON

下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 的二进制数（十六进制数）读入累加器,BCD 指令将累加器中的二进制数据转换成 BCD 数,OUTD 指令将结果写入到 R1500 和 R1501 中。



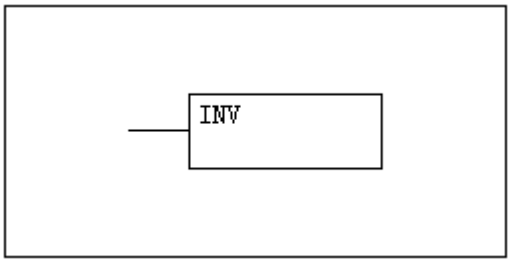
语句视图

```
LD  I1
LDD  R1400
BCD
OUTD  R1500
```

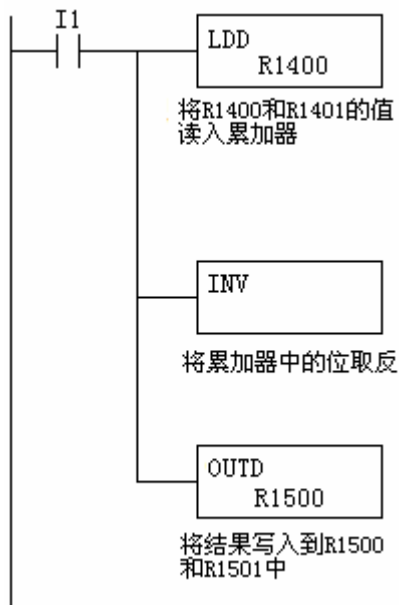

R1401
R1400
0 0 0 0 6 F 7 1
二进制值
累加器
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1
2147483648 1073741824 536870912 268435456 134217728 67108864 33554432 16777216 8388608 4194304 2097152 1048576 524288 262144 131072 65536 32768 16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1
16384+8192+2048+1024+512+256+64+32+16+1=28529
累加器
相等的BCD数
8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1
0 0 0 2 8 5 2 9
R1501
R1500
BCD数写入R1500
和R1501中

11.3 取反指令（INV）

INV 指令将累加器中的位取反，即 0 变 1，1 变 0，结果存放在累加器中。

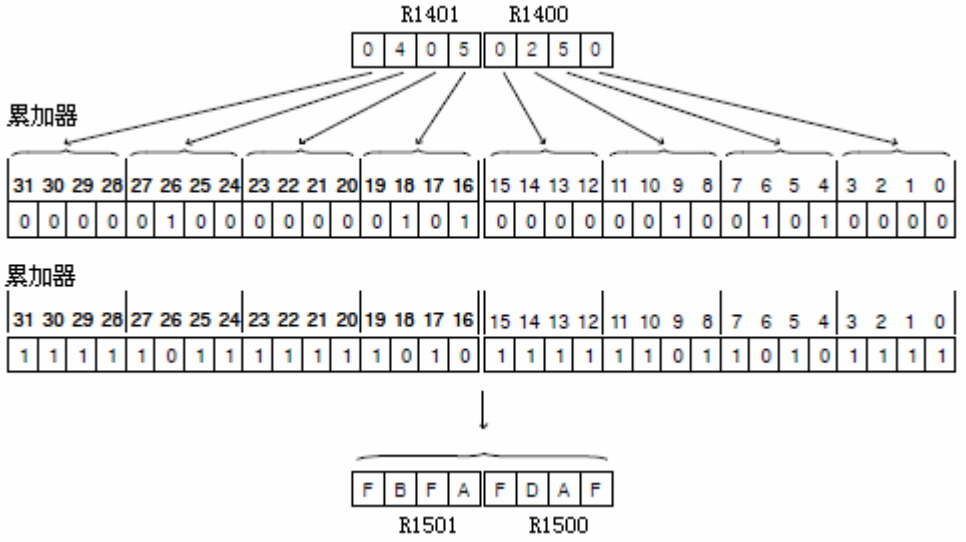


下面的例子中，当 I1 为 ON 时，LDD 指令将 R1400 和 R1401 的数据读入累加器，INV 指令将累加器中的位取反，结果写入到 R1500 和 R1501 中。



语句视图

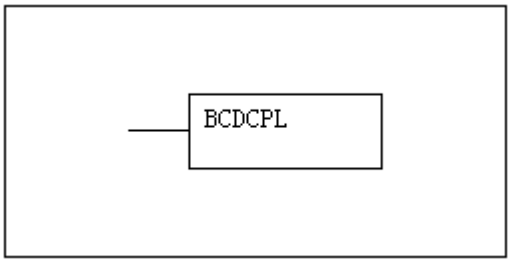
```
LD  I1
LDD R1400
INV
OUTD R1500
```



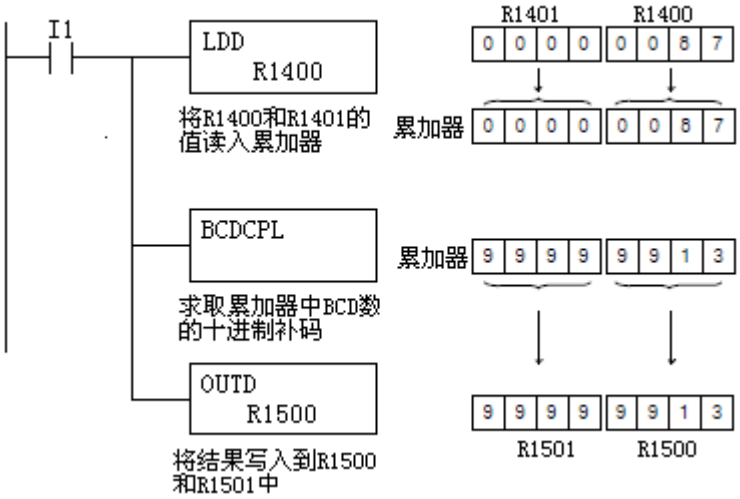
11.4 十进制补码变换（BCDCPL）

BCDCPL 指令对累加器中的 8 位 BCD 数求取 100000000 的补数，结果存放在累加器中。指令的算式是：

100000000
－ 累加器中数据
十进制补码值



下面的例子中，当 I1 为 ON 时，将 R1400 和 R1401 中的数据读入累加器，BCDCPL 指令求取累加器中数据的十进制补码，结果写入到 R1500 和 R1501 中。



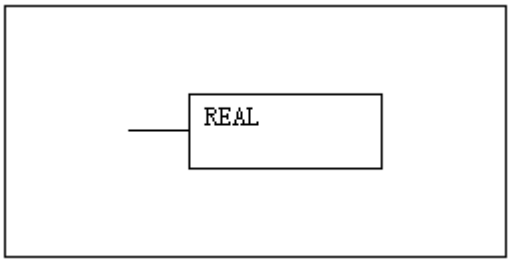
语句视图

```
LD I1
LDD R1400
BCDCPL
OUTD R1500
```

注意：如果程序中有减算指令导致借位的（有符号标志位标出），BCDCPL 指令可用来计算出两个数据之间的绝对差。

11.5 二进制到实数变换指令（REAL）

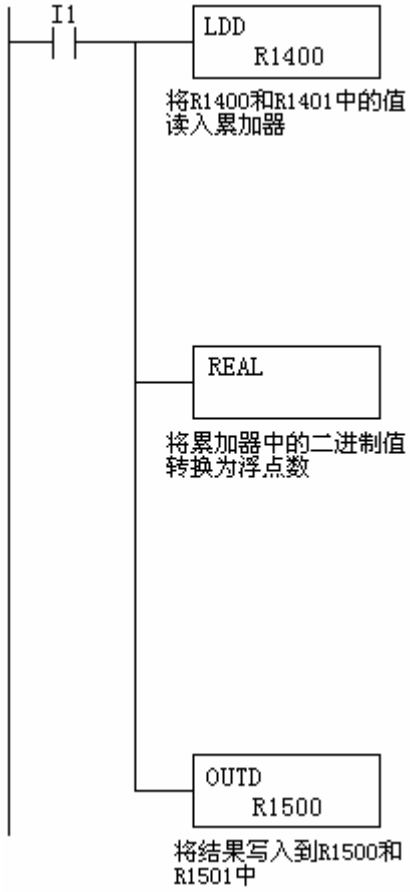
将累加器中的二进制数转换为实数，结果存放在累加器中。二进制数和实数都使用累加器的 32 位。



注意：本指令用于无符号数，不能用于有符号数。

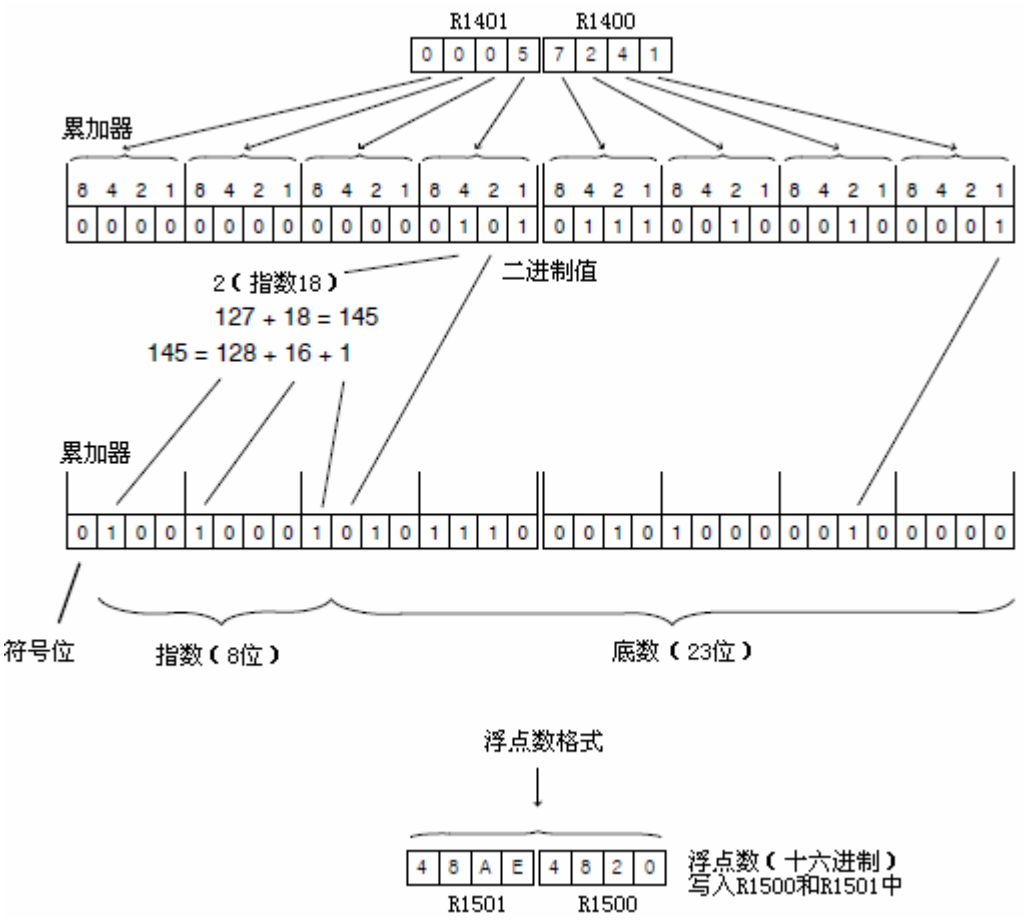
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

下面的例子中，当 I1 为 ON 时，将 R1400 和 R1401 的数据读入累加器，REAL 指令将累加器中的二进制数转换为实数。通过将二进制数的最高位位号加上 127（十进制），将其转换成实数指数，剩余的位是尾数，见下图。然后将结果写入到 R1500 和 R1501 中。查看 R1500 和 R1501 中的数，显示的是十六进制数。



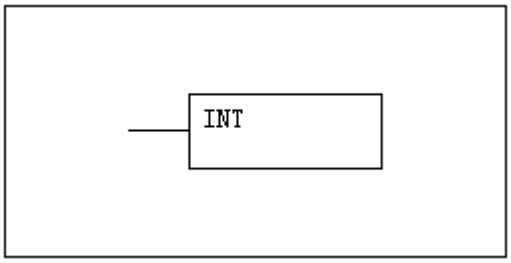
语句视图

```
LD  I1
LDD  R1400
REAL
OUTD  R1500
```


R1401
R1400
0 0 0 5
7 2 4 1
累加器
8 4 2 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
0 1 1 1 0 0 1 0 0 0 1 0 0 0 0 1
2（指数18）
二进制值
127 + 18 = 145
145 = 128 + 16 + 1
累加器
0 1 0 0 1 0 0 0 1 0 1 0 1 1 1 0
0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0
符号位
指数（8位）
底数（23位）
浮点数格式
4 8 A E
4 8 2 0
R1501
R1500
浮点数（十六进制）
写入R1500和R1501中

11.6 实数到二进制数变换指令（INT）

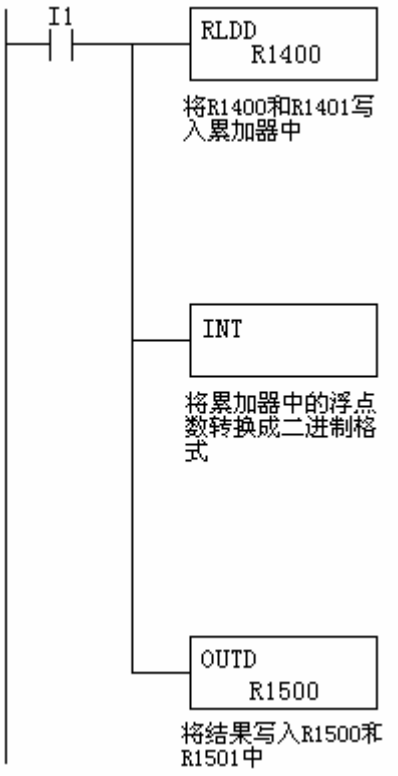
将累加器中的实数转换为二进制数。结果存放在累加器中。二进制数和实数都使用累加器的 32 位。



注意 1：结果的小数部分将向下舍入（例如，14.1→14 或-14.1→-15）。
注意 2：如果实数是负数，将转化成一个有符号数（以二进制补码格式存放）。

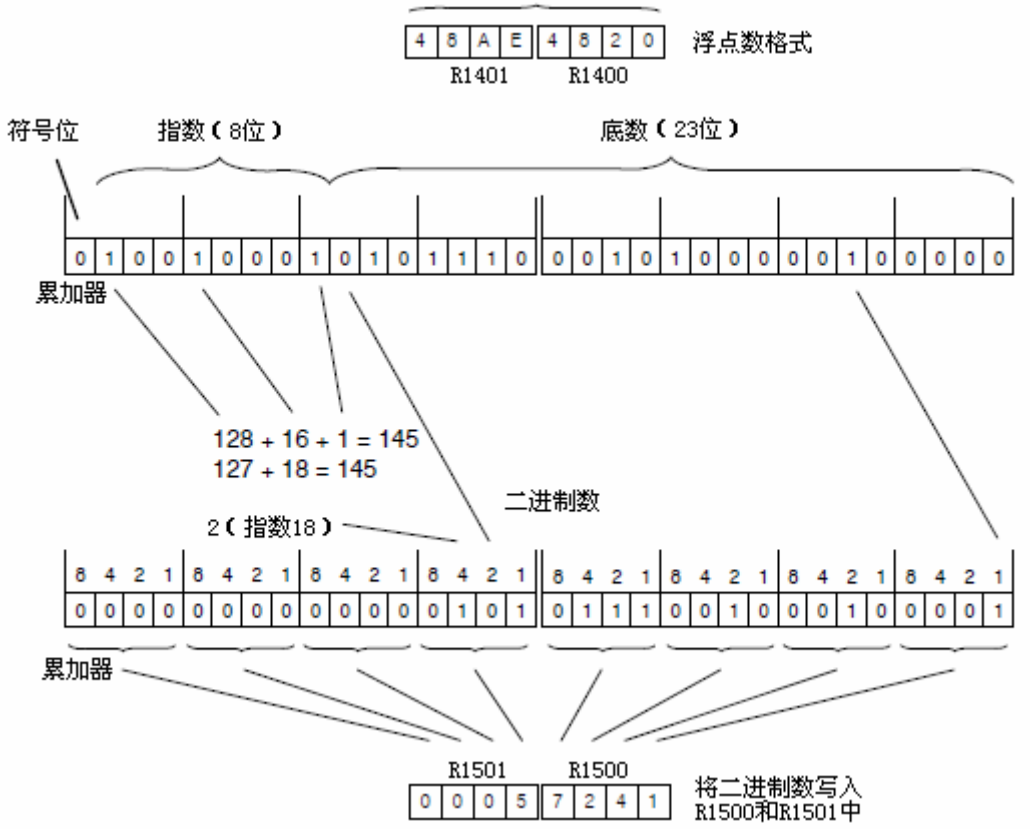
受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON
SP75	当数值不能转换为二进制数时 ON

下面的例子中，当 I1 为 ON 时，将 R1400 和 R1401 的数据读入累加器，INT 指令将累加器中的实数转换为二进制数。然后将结果写入到 R1500 和 R1501 中。查看 R1500 和 R1501 中的数，显示的是十六进制数。



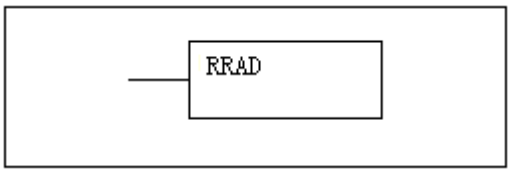
语句视图

```
LD I1
RLDD R1400
INT
OUTD R1500
```



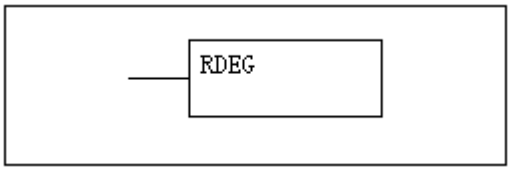
11.7 实数弧度变换指令（RRAD）

RRAD 指令将累加器中实数格式的角度数据变换为相等的实数格式的弧度数据，结果存放在累加器中。



11.8 实数角度变换指令（RDEG）

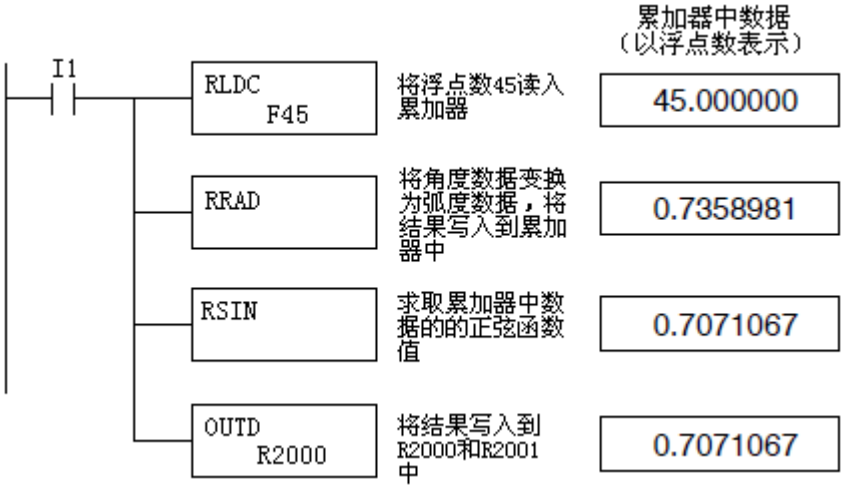
RDEG 指令将累加器中实数格式的弧度数据变换为相等的实数格式的角度数据，结果存放在累加器中。



上面两个指令，是将累加器中的实数从角度变换为弧度，或是从弧度变换为角度。当是角度数据格式时，一个圆的角度是 360°；当是弧度数据格式时，一个圆的弧度是 2π（约 6.28）。被转换和转换后的数据都是正的或负的实数，大于一个圆的角度的数据也可以被转换，同三角函数指令配合使用，非常方便。

受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON
SP72	累加器中的值不是一个有效的实数时 ON
SP73	带符号运算指令执行时，结果溢出时 ON

下面的例子中，求取 45° 的正弦函数。由于三角函数的运算对象只能是弧度，因此用 RRAD 指令将角度变换为弧度，RSIN 指令求取正弦值，将结果写入到 R2000 和 R2001 中。

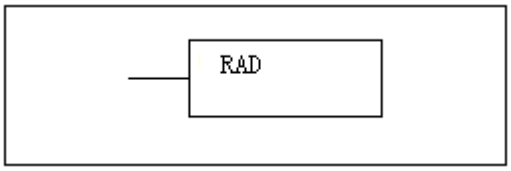


语句视图

```
LD  I1
RLDC  F45
RRAD
RSIN
OUTD  R2000
```

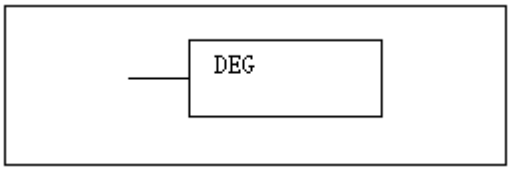
11.9 BCD 数弧度变换指令（RAD）

RAD 指令将累加器中 BCD 格式的角度数据变换为相等的 BCD 格式的弧度数据，结果存放在累加器中。



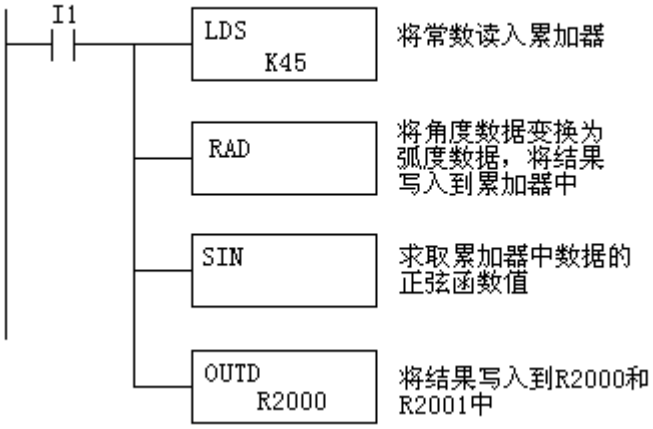
11.10 BCD 数角度变换指令（DEG）

DEG 指令将累加器中 BCD 格式的弧度数据变换为相等的 BCD 格式的角度数据，结果存放在累加器中。



受影响的标志线圈	描述
SP63	指令运算导致累加器中的数据为 0 时 ON
SP70	指令运算导致累加器中的数据为负数时 ON

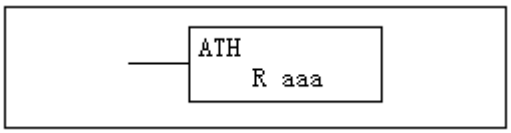
下面的例子中，求取 45° 的正弦函数。由于三角函数的运算对象只能是弧度，因此用 RAD 指令将角度变换为弧度，SIN 指令求取正弦值，将结果写入到 R2000 和 R2001 中。



语句视图

```
LD  I1
LDS K45
RAD
SIN
OUTD R2000
```

11.11 ASCII→HEX 码变换指令（ATH）



ATH 指令将一组 ASCII 数据转换成十六进制格式。ASCII 数据是 2 个数字，而其对应的十六进制码是 1 个数字。这就意味着一组 4 个 R 寄存器的 ASCII 数据变换成相等的十六进制数据只需要 2 个 R 寄存器。指令执行前，需将功能参数读入累加器和数据堆栈中，下面是将 ASCII 数据转换为十六进制数据的步骤：

第 1 步：将 ASCII 数据表的 R 寄存器的数目读入数据堆栈第 1 级中。

第 2 步：将 ASCII 数据表的起始 R 寄存器地址读入累加器中，参数必须是一个十六进制数。

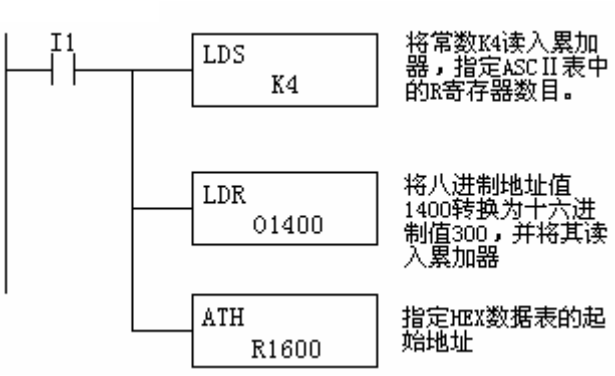
第 3 步：在 ATH 指令中指定十六进制数据表的起始 R 寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

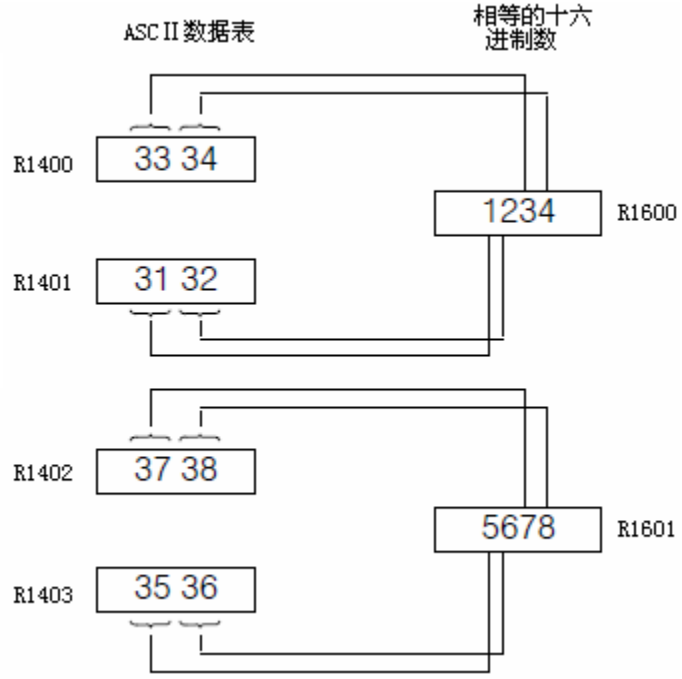
下面的例子中，当 I1 为 ON 时，常数 K4 被读入累加器，下一个读入指令执行时 K4 被压入数据堆栈的第 1 级中。ASCII 数据表的起始寄存器地址（R1400）被读入累加器。ATH 指令中指定了十六进制数据表的起始寄存器地址（R1600）。下表列出了 ATH 变换的有效 ASCII 值。

ATH 变换的有效 ASCII 码与 HEX 的对照表			
ASCII 值	HEX 值	ASCII 值	HEX 值
30	0	38	8
31	1	39	9
32	2	41	A
33	3	42	B
34	4	43	C
35	5	44	D
36	6	45	E
37	7	46	F

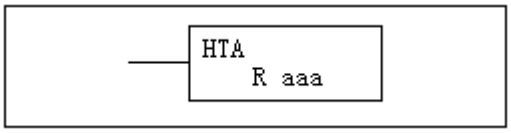


语句视图

```
LD  I1
LDS K4
LDR 01400
ATH R1600
```



11.12 HEX→ASCII 码变换指令（HTA）



HTA 指令将一组十六进制数据转换成 ASCII 码。十六进制码是 1 个数字，而其对应的 ASCII 数据是 2 个数字。这就意味着一组 2 个 R 寄存器的十六进制数据变换成相等的 ASCII 数据需要 4 个 R 寄存器。指令执行前，需将功能参数读入累加器和数据堆栈中，下面是将十六进制数据转换为 ASCII 数据的步骤：

第 1 步：将十六进制数据表的 R 寄存器的数目读入数据堆栈第 1 级中。

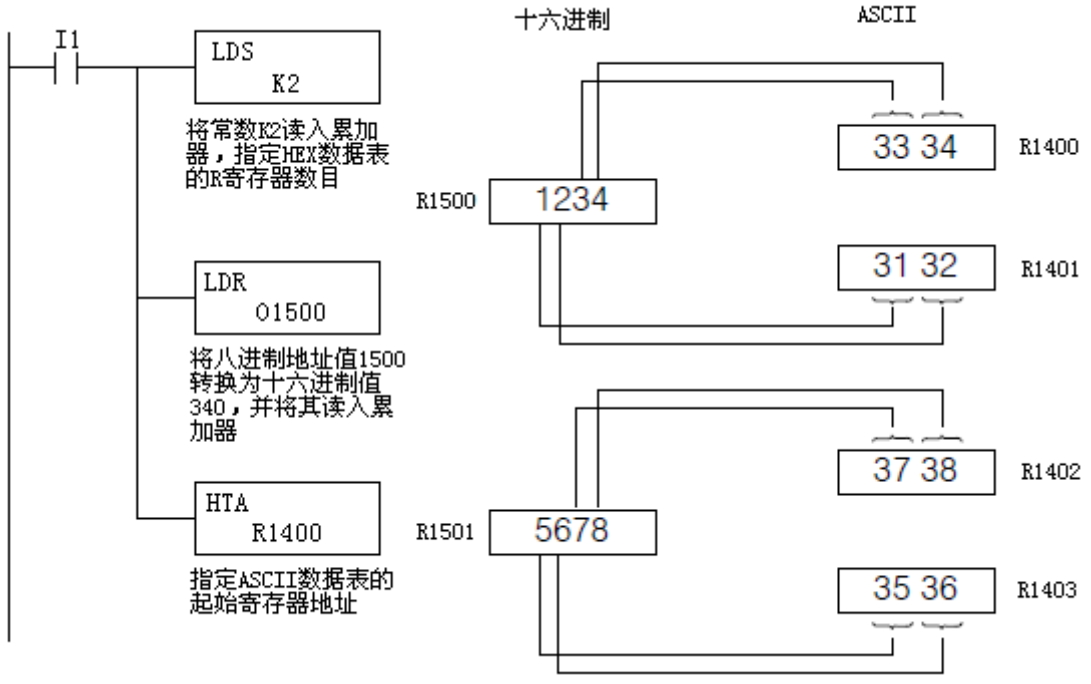
第 2 步：将十六进制数据表的起始 R 寄存器地址读入累加器中，参数必须是一个十六进制数。

第 3 步：在 HTA 指令中指定 ASCII 数据表的起始 R 寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

下面的例子中，当 I1 为 ON 时，常数 K2 被读入累加器，下一个读入指令执行时 K2 被压入数据堆栈的第 1 级中。十六进制数据表的起始寄存器地址（R1500）被读入累加器。HTA 指令中指定了 ASCII 数据表的起始寄存器地址（R1400）。



语句视图

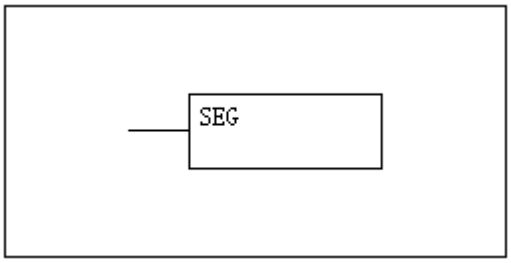
```
LD I1
LDS K2
LDR 01500
HTA R1400
```

下表列出了 HTA 变换的有效 ASCII 值。

ATH 变换的有效 ASCII 码与 HEX 的对照表			
HEX 值	ASCII 值	HEX 值	ASCII 值
0	30	8	38
1	31	9	39
2	32	A	41
3	33	B	42
4	34	C	43
5	35	D	44
6	36	E	45
7	37	F	46

11.13 7 段译码指令（SEG）

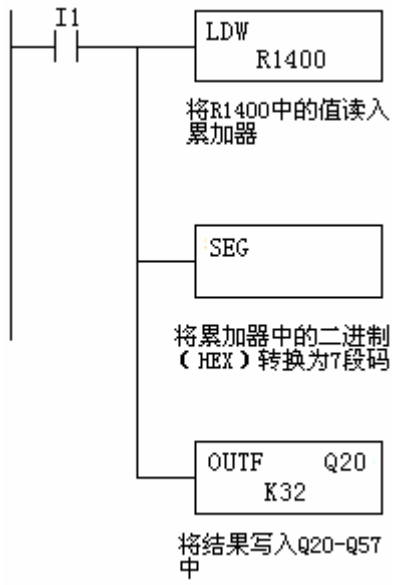
BCD/SEG 指令将累加器中 4 位十六进制数据转换成 7 段码显示格式，结果存放在累加器中。



下面是译码对应表：

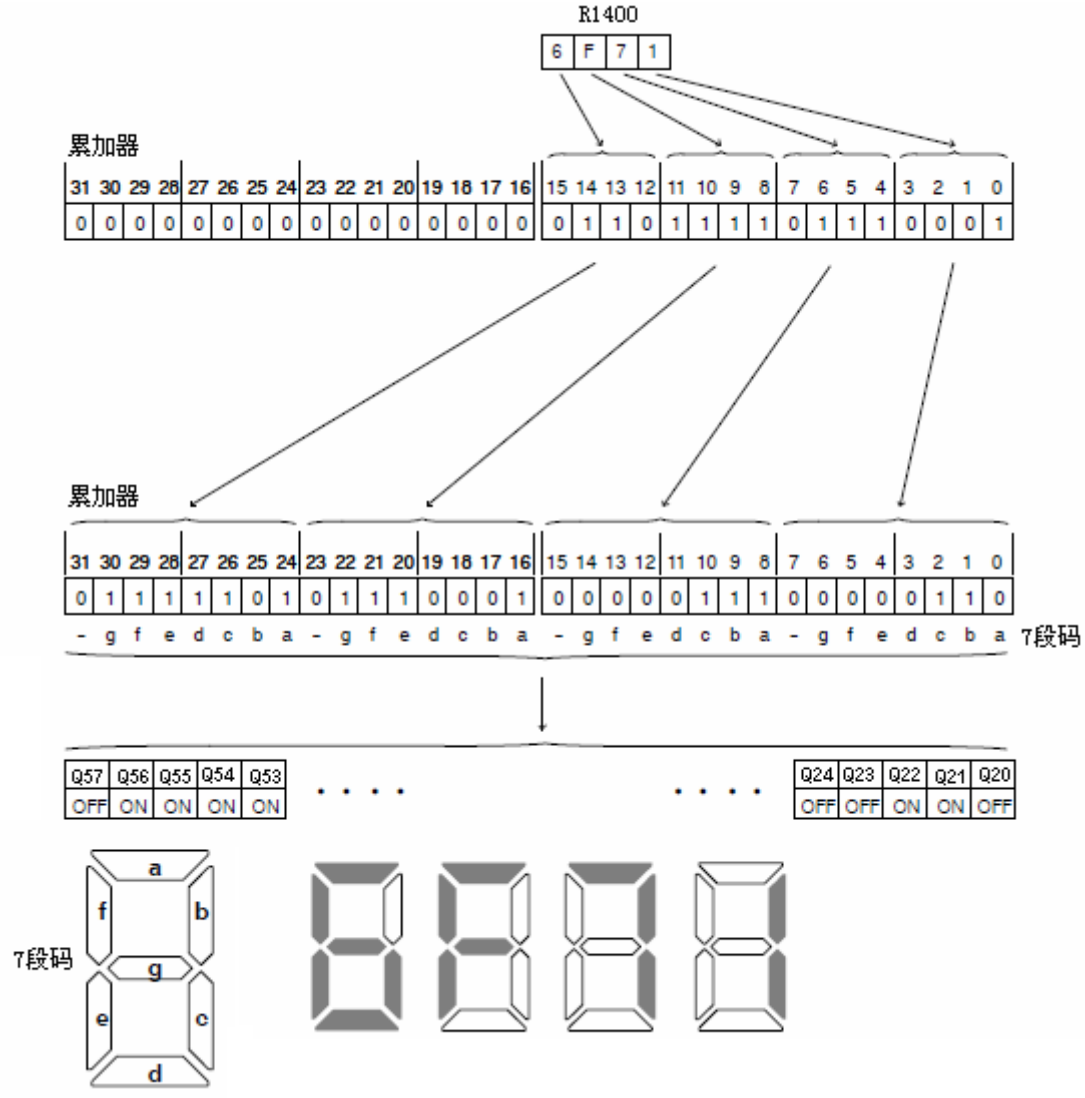
ACC					显示数据	7 段码								HEX 代码
						An								
	a_3	a_2	a_1	a_0		h	g	f	e	d	c	b	a	
0	0	0	0	0	0	0	0	1	1	1	1	1	1	3F
1	0	0	0	1	1	0	0	0	0	0	1	1	0	06
2	0	0	1	0	2	0	1	0	1	1	0	1	1	5B
3	0	0	1	1	3	0	1	0	0	1	1	1	1	4F
4	0	1	0	0	4	0	1	1	0	0	1	1	0	66
5	0	1	0	1	5	0	1	1	0	1	1	0	1	6D
6	0	1	1	0	6	0	1	1	1	1	1	0	1	7D
7	0	1	1	1	7	0	0	1	0	0	1	1	1	27
8	1	0	0	0	8	0	1	1	1	1	1	1	1	7F
9	1	0	0	1	9	0	1	1	0	1	1	1	1	6F
A	1	0	1	0	A	0	1	1	1	0	1	1	1	77
B	1	0	1	1	B	0	1	1	1	1	1	0	0	7C
C	1	1	0	0	C	0	0	1	1	1	0	0	1	39
D	1	1	0	1	D	0	1	0	1	1	1	1	0	5E
E	1	1	1	0	E	0	1	1	1	1	0	0	1	79
F	1	1	1	1	F	0	1	1	1	0	0	0	1	71

下面的例子中，当 I1 为 ON 时，LDW 指令将 R1400 中的数据读入累加器。SEG 指令将累加器中的十六进制数（二进制数）转换为 7 段码格式。OUTF 指令将累加器中的数据写入到 Q20-Q57 中。

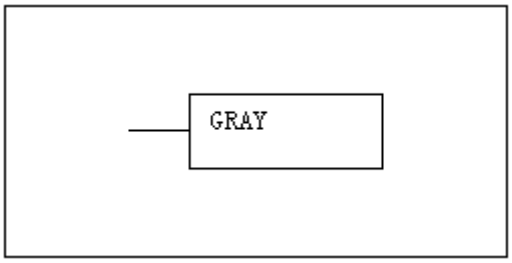


语句视图

```
LD I1
LDW R1400
SEG
OUTF Q20 K32
```

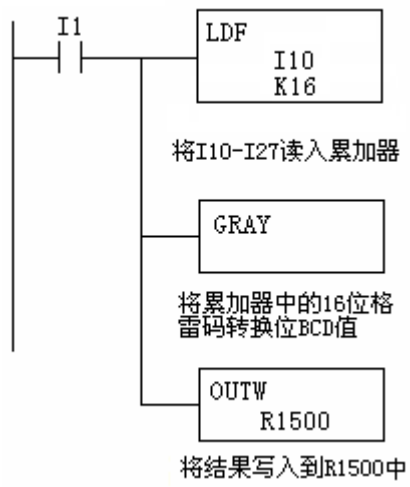


11.14 GRAY→BCD 码变换指令（GRAY）



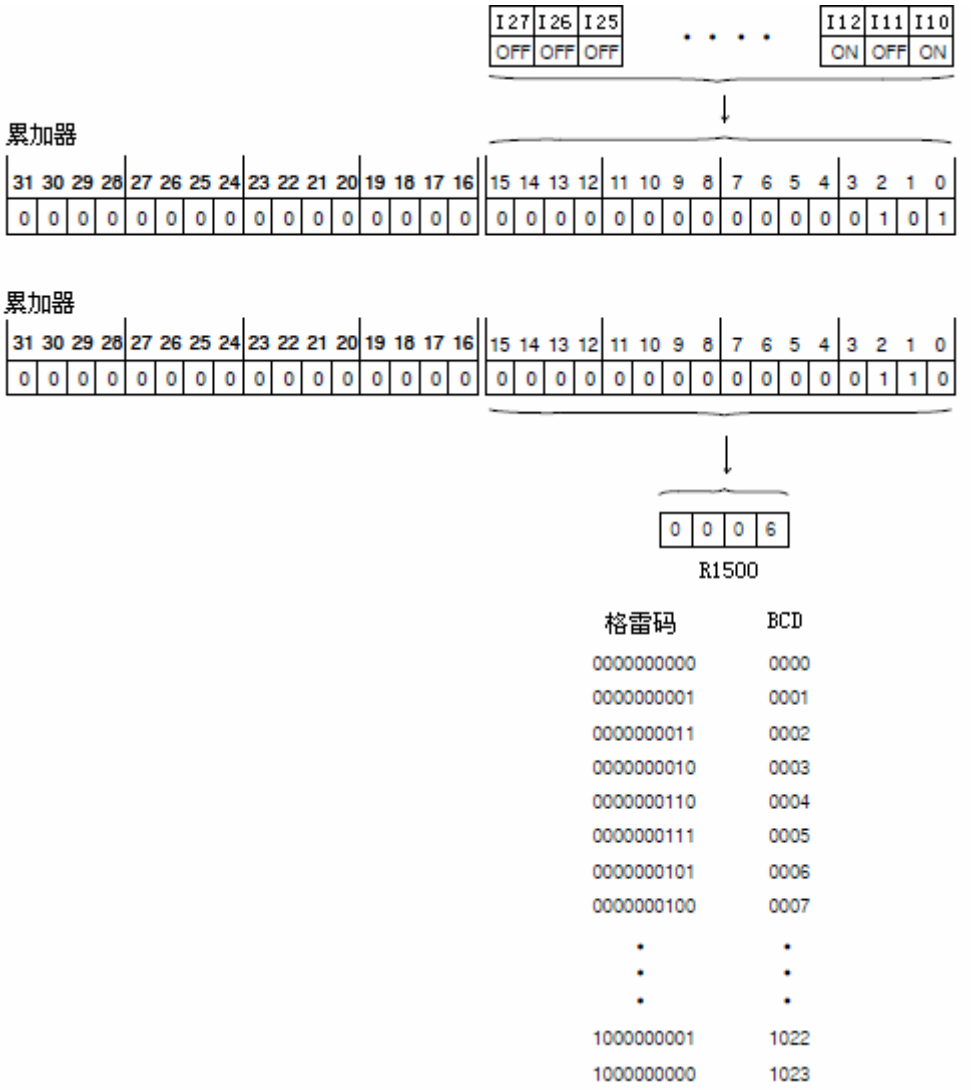
GRAY 指令将一个 16 位格雷码转换为一个 BCD 数。BCD 转换需要累加器中的低 10 位，高 22 位被设置为“0”。本指令用于使用格雷码的设备(通常是编码器),可将分辨率为 512 或 1024 的格雷码直接变换为对应的 BCD 码，而对分辨率为 360 和 720 的格雷码，转换后要分别减去 76 和 152，才能得到正确的 BCD 码。

下面的例子中，当 I1 为 ON 时，LDF 指令将 I10-I27 读入累加器，GRAY 指令将累加器中的数据转换成 BCD 形式，将结果写入到 R1500 中。



语句视图

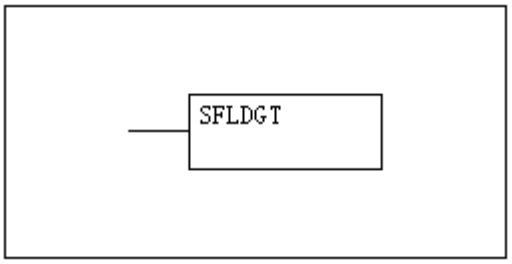
```
LD I1
LDF I10 K16
GRAY
OUTW R1500
```



格雷码	BCD
0000000000	0000
0000000001	0001
0000000011	0002
0000000010	0003
0000000110	0004
0000000111	0005
0000000101	0006
0000000100	0007
•	•
•	•
•	•
1000000001	1022
1000000000	1023

11.15 位替换指令（SFLDGT）

SFLDGT 指令将数据堆栈第 1 级中存放的数据按累加器中存放的位指定数据进行排列处理。



SFLDGT 指令执行之前，需要将功能参数分别读入数据堆栈第 1 级和累加器中。下面是实现位替换功能的步骤：

第 1 步：将要进行位排列处理的数据读入数据堆栈的第 1 级中。

第 2 步：将位指定数据读入累加器，位指定数据的有效范围是 1-8（“1”是最低位，“8”是最高位）。

第 3 步：写入 SFLDGT 指令。

注意： 1. 如果位指定数据是 1-8 之外的数（0 或 9-F）时，相应位将被设置为 0，见下面的例子。

2. 当位指定数据有重复时，累加器的高位指定有效，见下面的例子。

位替换图解

最多有 8 位可被替换，数据堆栈第 1 级中存放的数据是要进行位排列处理的数据，这个数据按照累加器中的位指定数据进行重新排列，结果存入累加器中，见右图。

下面的例子中，当 I1 为 ON 时，数据堆栈第 1 级中的数据位按照累加器中的数据进行重新排列。

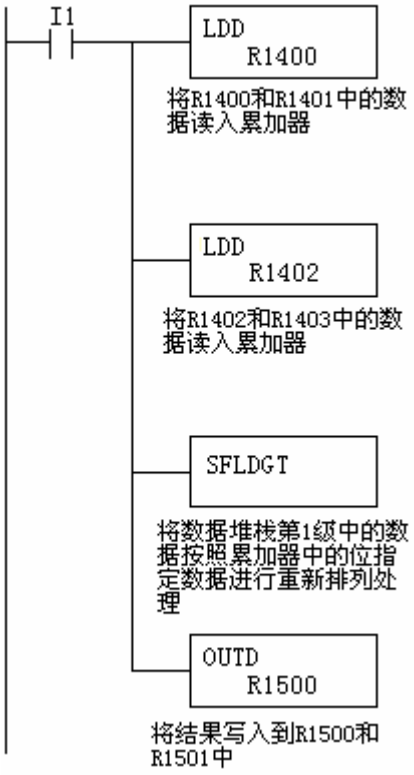
例 A 演示了当位指定数据没有 0 或 9-F，并且没有重复的数据时的位替换。

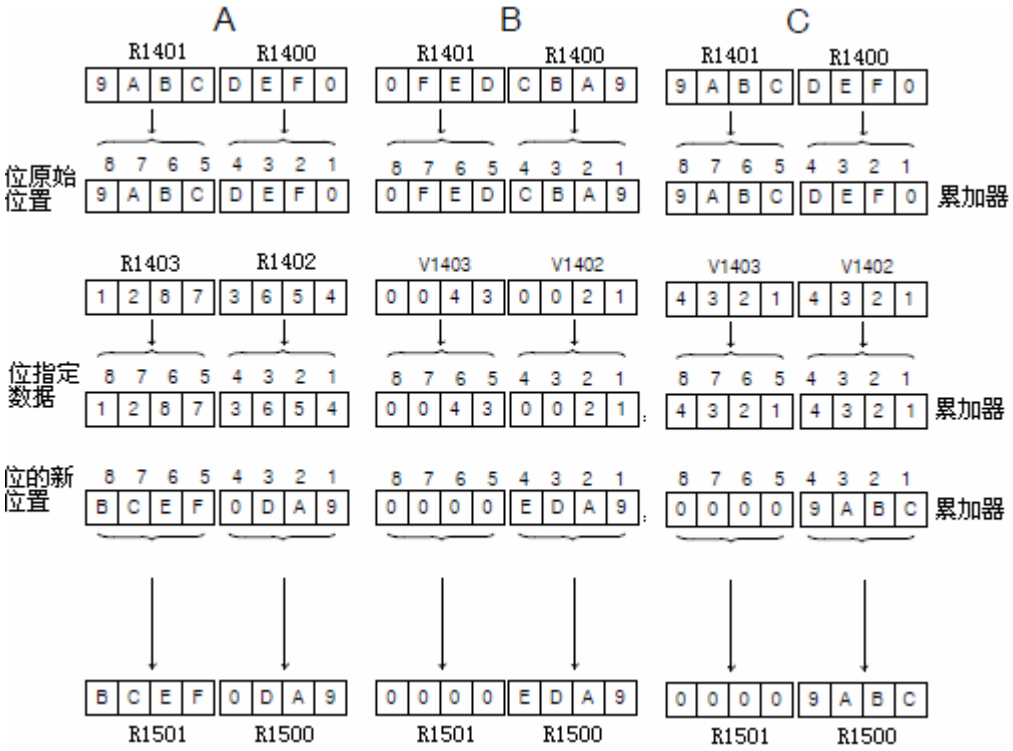
例 B 演示了位指定数据中有 0 或 9-F 时的位替换。注意，当替换指令执行时，相应位数据被设置为“0”。

例 C 演示了当位指定数据有重复时，当替换指令执行时，累加器中的高位指定有效。

语句视图

```
LD I1
LDD R1400
LDD R1402
SFLDGT
OUTD R1500
```

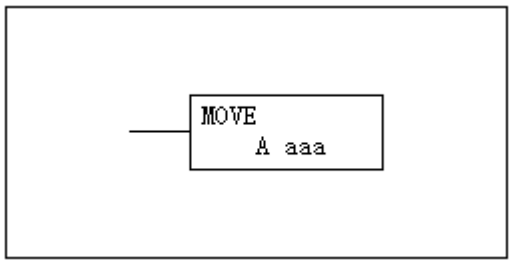




第 12 章 表检索指令

12.1 数据组传送指令（MOVE）

MOVE 指令将一组最多 4095 个数据从一个 R 寄存器表传送到另一个相同长度的寄存器表中。指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：



第 1 步：将要被传送的数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FFF。

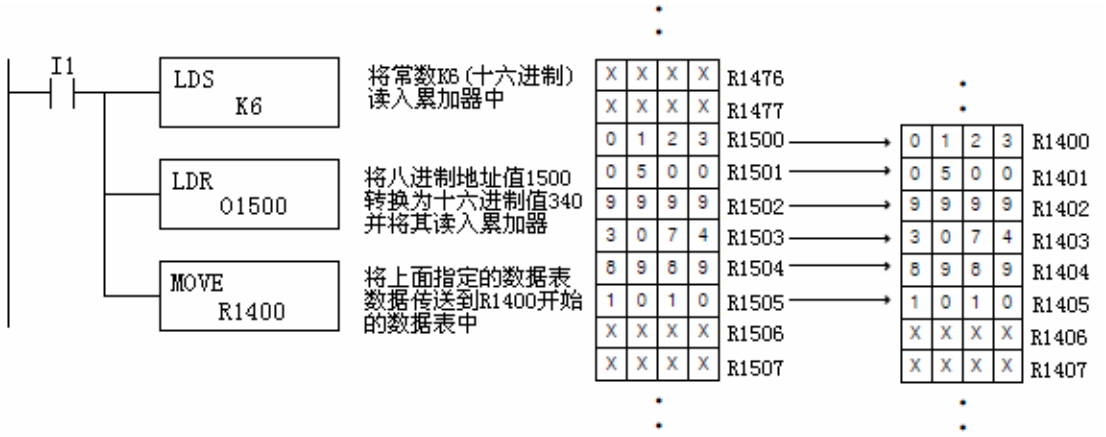
第 2 步：将数据表的起始寄存器地址读入累加器，这个数必须是十六进制数。

第 3 步：写入 MOVE 指令，指定目标数据表的起始寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）
P 指针 P	所有（附录 1）

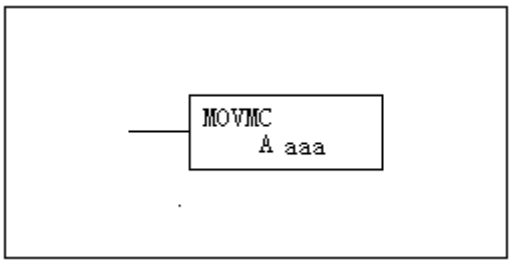
下面的例子中，当 I1 为 ON 时，常数 K6 被读入累加器，这个数据指定了数据表的长度，当第二个读入指令执行时，常数 K6 被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1500（R1500）读入累加器，这是数据表的起始地址。MOVE 指令中指定要传送的目标寄存器数据表的起始地址。



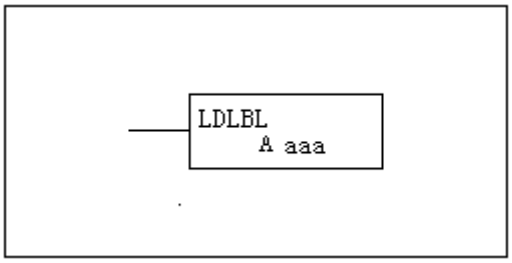
语句视图

```
LD  I1
LDS K6
LDR O1500
MOVE R1400
```

12.2 程序存储器－数据寄存器之间的数据传送指令（MOVMC）



12.3 数据标记地址读出指令（LDLBL）



MOVMC 指令用于程序存储器向数据寄存器传送数据。当数据由程序存储器到数据寄存器传送时，LDLBL 指令同 MOVMC 指令一起使用。

程序存储器向数据寄存器传送数据之前，需将功能参数分别读入数据堆栈的第 1、第 2 级以及累加器中。下面是实现此指令功能的步骤：

第 1 步：将要传送的字的数目（最多 255 个）读入数据堆栈的第 2 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据源区域（十六进制）及目标数据表的起始寄存器地址的偏差值读入数据堆栈的第 1 级中。

第 3 步：将数据由程序存储器传送到 R 寄存器时，将数据源标记地址读入累加器。

第 4 步：写入 MOVMC 指令，指定目标数据表地址。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	K	1-FFFF

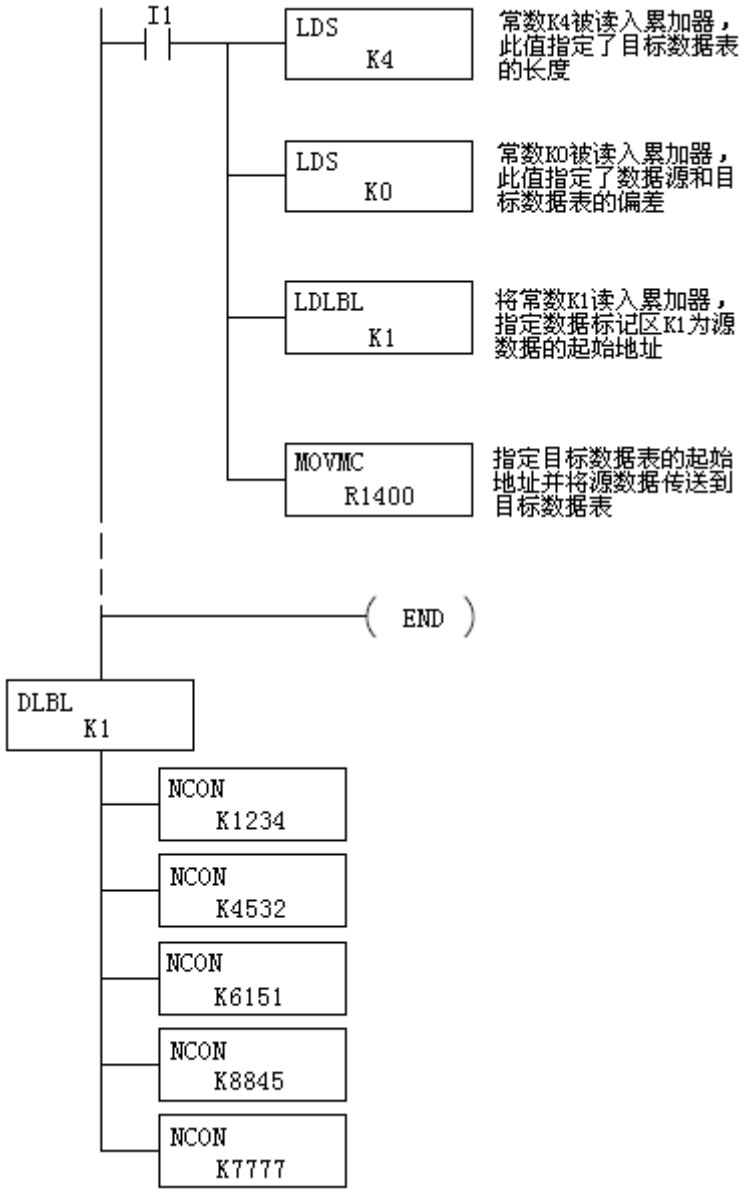
受影响的标志线圈	描述
SP53	表指针错误时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。

警告：本指令的偏差值从 0 开始，也可以是不会导致数据源区域之外的数据被传送到目标数据表的任何数据。当偏差值超出数据源区域的范围，未知的数据将被传送到目标数据表中。

将数据从数据标记区传送到 R 寄存器

下面的例子中，数据从数据标记区传送到 R 寄存器。当 I1 为 ON 时，常数 K4 被读入累加器，此值指定了目标数据表的长度，当下面两个读入指令执行时，此值被压入数据堆栈的第 2 级中。常数 K0 被读入累加器，此值指定了数据源区域及目标数据表的偏移量，当 LDLBL 指令执行时，此值被压入数据堆栈的第 1 级中。LDLBL 指令将数据源标记地址读入累加器中。MOVMC 指令指定目标数据表的起始寄存器地址并将数据从数据源传送到 R 寄存器中。



语句视图

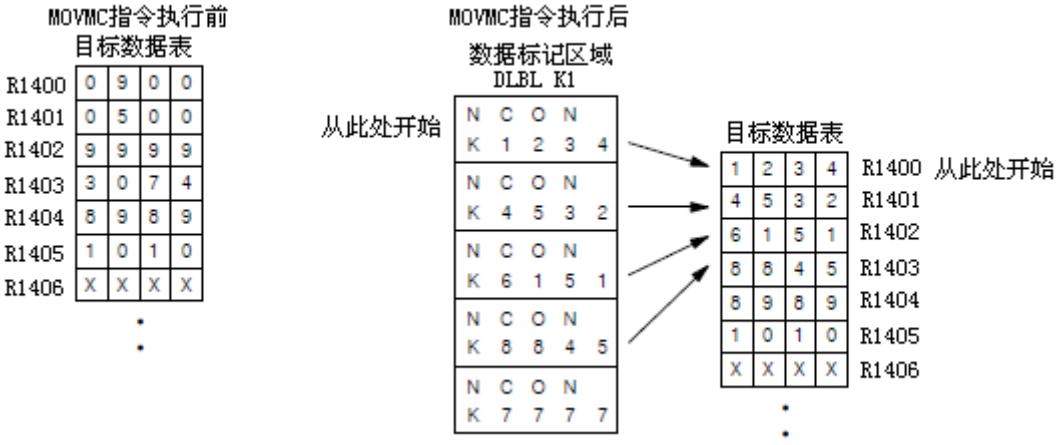
```
LD I1
LDS K4
LDS K0
LDLBL K1
MOVMC R1400
END
DLBL K1
NCON K1234
NCON K4532
NCON K6151
NCON K8845
NCON K7777
```

警告：上例中的偏差值是 0，也可以是不会导致数据源数据区之外的数据被传送到目标数据表的任何数据。当偏差值超出数据源数据区的范围时，未知的数据将被传送到目标数据表中。

下图给出了上例的运行结果。偏移量为 0，4 个字被传送。

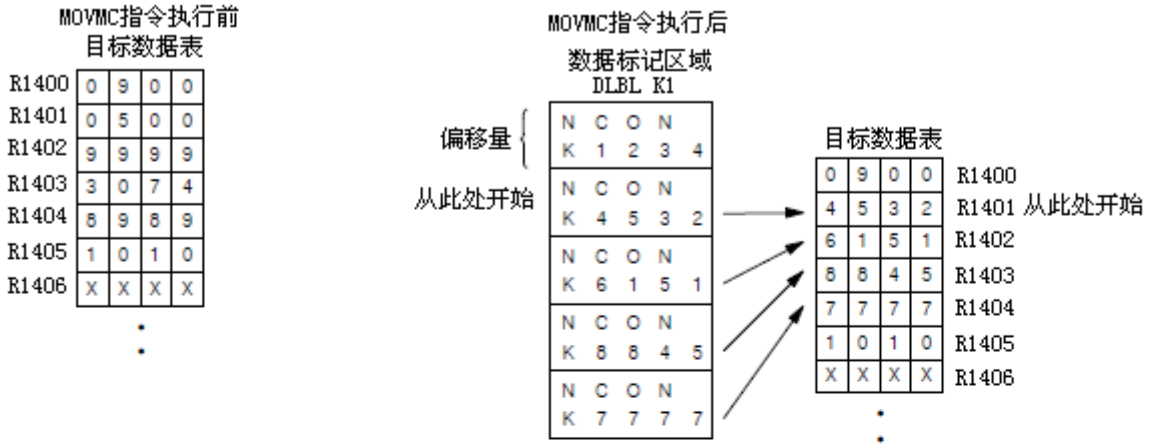
运行举例

偏移量=0，传送 4 字

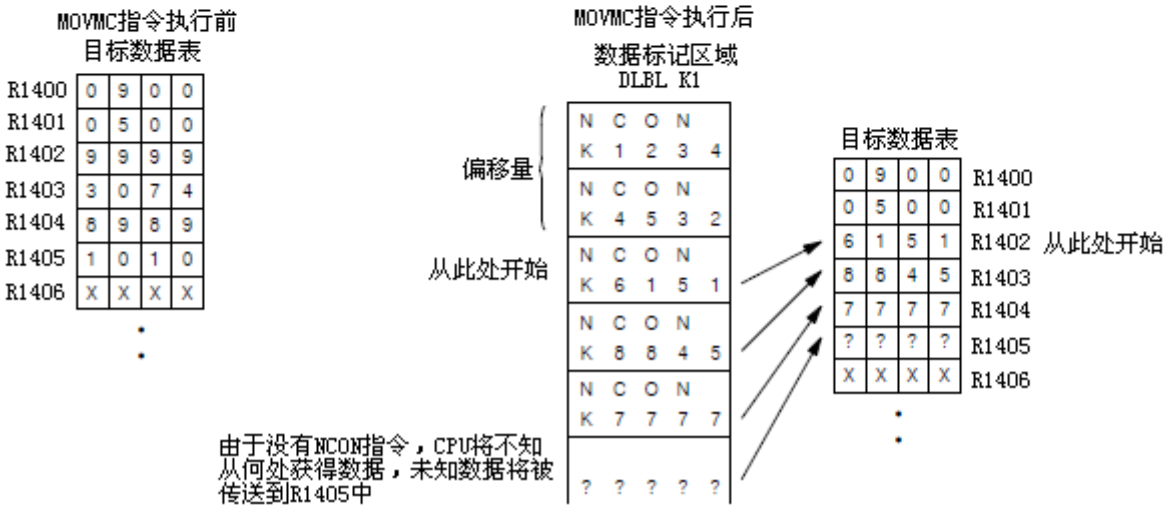


上图中很清楚的演示了偏移量为 0 时的程序运行结果。下图给出了当偏移量为其它数值时的程序运行结果，通过下图可以看到，偏移量是数据源区域及目标数据表的偏移量，并且，当使用了不恰当的偏移量（本例中为 2）时，会有未知的数据被传送到目标数据表。

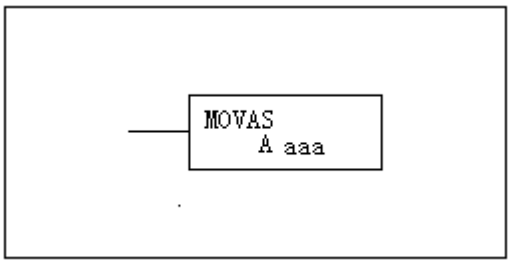
偏移量=1，传送 4 字



偏移量=2，传送 4 字



12.4 登记数据寄存器传送指令（MOVAS）

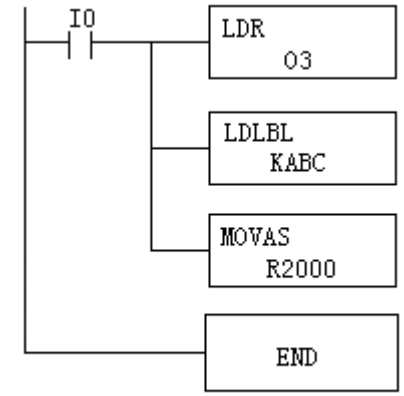


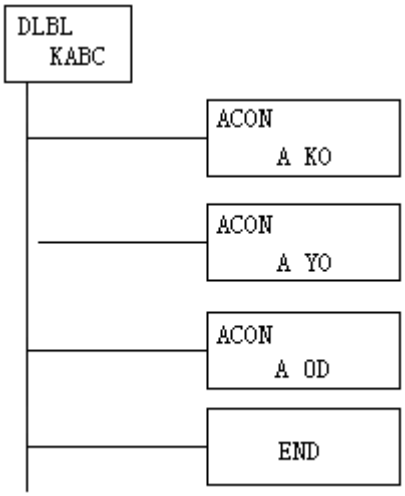
MOVAS 指令用于将程序存储器中登记的数据，向数据寄存器中成组传送。

1. 使用此指令，首先要在程序存储器中有 DLBL 登记的标记名，在其后有用 ACON 或 NCON 登记的数据。
2. 执行此指令，在数据堆栈第 1 级存入传送字数，累加器中存入用 LDLBL 指令读入的数据标记名。
3. 程序存储器中登记的数据，1 个指令（1 个地址）是 16 位数据。
   注意：程序存储器地址是 10 进制数，但传送字数用 8 进制设定。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

下面的例子中，条件 I0 成立时，数据标号 ABC 内登记的数据由 MOVAS 指令传送到指定的 R2000-R2002 中。





语句视图

```
LD I0
LDR O3
LDLBL KABC
MOVAS R2000
END
DLBL KABC
ACON A KO
ACON A YO
ACON A OD
END
```

12.5 任意位置位指令（BSET）

BITSET 指令将 R 寄存器表中的指定位置位。

12.6 任意位复位指令（BITRST）

BITRST 指令将 R 寄存器表中的指定位复位。

下面是实现这两个指令功能的步骤：

第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 1 级中。这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始地址读入累加器中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数据。

第 3 步：写入 BITSET 指令或 BITRST 指令，指定想要置位或复位的位号。位号按八进制编号，数据表第 1 个位的位号是“0”。

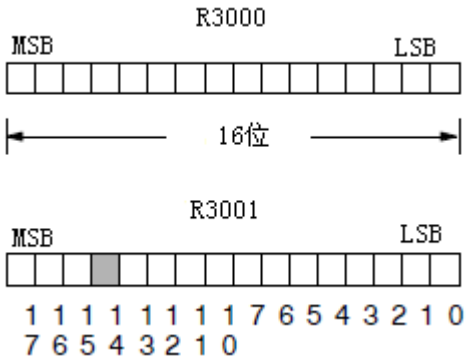
小技巧：每个 R 寄存器包含 16 位，因此，数据表的第 1 个寄存器位号范围是 0-17（八进制）。例如，假定数据表长度是 6 字=（6×16）位=96 位（十进制），或 140 位（八进制）。位的编号范围是 0-137。当指定的位超出这个范围时，SP53 将被置为 ON。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

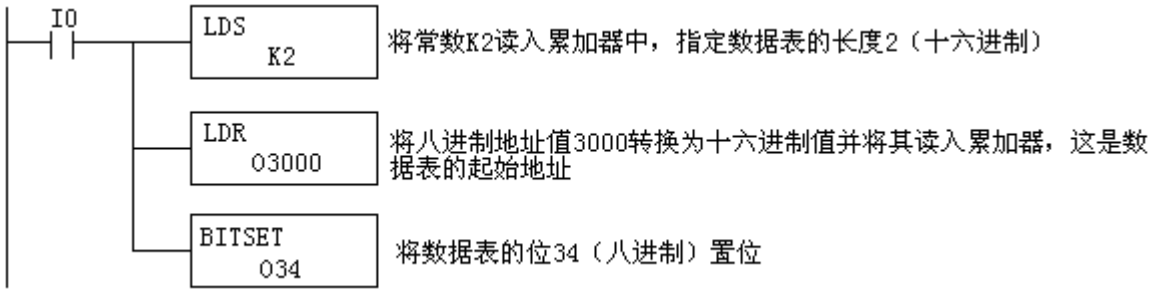
受影响的标志线圈	描述
SP53	指定的位超出表范围时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。

举例，假设数据表起始地址是 R3000，2 字长，如右图所示。每个字包含 16 位，或 0-17（八进制）。要将第二个字的位 12 置位，其八进制编号为 14。计算其在数据表中的编号 17+14=34（八进制），下面的程序演示了如何将这个位置位。



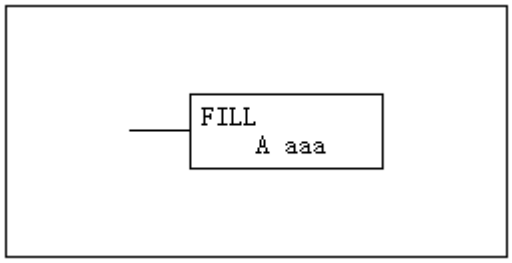
下面的例子中，使用 I0 来触发 BITSET 指令。首先，将数据表长度值（2 字）读入数据堆栈的第 1 级中。然后，将数据表的起始地址读入累加器中。由于 R3000 是一个八进制地址值，我们使用 LDR 指令，将地址读入累加器的同时还将其转换成了十六进制数据。最后，写入 BITSET（或 BITRST）指令，指定位在数据表中的八进制编号（位 34）。



语句视图

```
LD I0
LDS K2
LDR O3000
BITSET O34
```

12.7 同一数据成组写入指令（FILL）



FI11 指令将同一个数据（R 寄存器或 4 字常数）写入到一组（最多 255 个）R 寄存器中。指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中。下面是实现这一指令功能的步骤：

第 1 步：将被写入数据的 R 寄存器的数目读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将被写入数据表的起始 R 寄存器地址写入累加器，这个数必须是十六进制数。

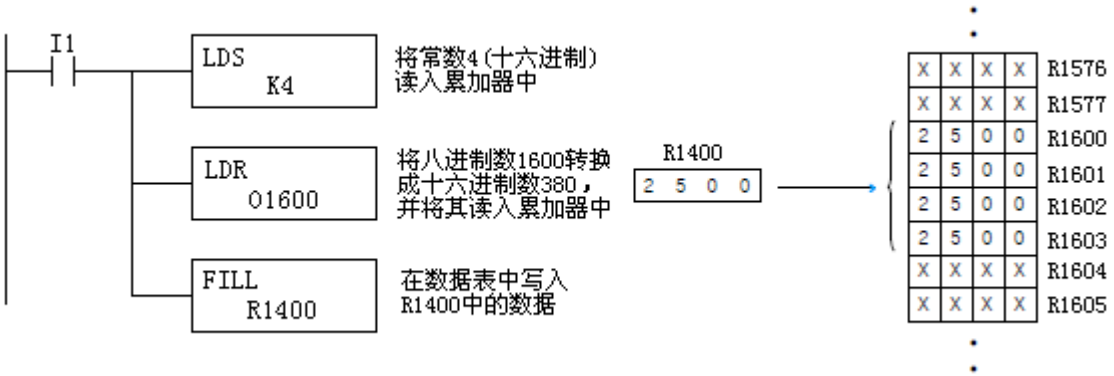
第 3 步：写入 FILL 指令，并指定要写入的数据。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
常数	K	0-FFFF

受影响的标志线圈	描述
SP53	R 寄存器地址超出范围时 ON

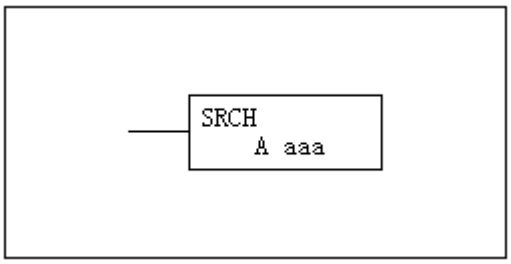
下面的例子中，当 I1 为 ON 时，LDS 指令将常数 K4 读入累加器中，此常数指定了被写入数据的 R 寄存器数目。LDR 指令将八进制地址值 1600 写入累加器，并将累加器中原有的数据常数 4（十六进制）压入堆栈的第 1 级中，这个地址值是被写入数据表的起始寄存器地址。FILL 指令指定要写入的数据并将其写入到数据表中。



语句视图

```
LD I1
LDS K4
LDR 01600
FILL R1400
```

12.8 同一数据检索指令（SRCH）



SRCH 指令用于在一组（最多 255 个）R 寄存器中检索一个指定的数据。指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级、第 2 级和累加器中。下面是实现此指令功能的步骤：

第 1 步：将要被检索的数据表的长度读入数据堆栈的第 2 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将要被检索的数据表的起始 R 寄存器地址读入数据堆栈的第 1 级中，这个数必须是十六进制数。

第 3 步：将起始寄存器地址的偏移量读入累加器中，偏移量指定从何处开始检索，必须是十六进制数。

第 4 步：写入 SRCH 指令，并指定要检索的第一个数据。

执行结果：指令执行结果是包含被检索数据的寄存器在数据表中相对于起始地址的偏差量被写入累加器中。如果被检索数据所在的地址在偏移量指定检索的区域之外或是未找到要检索的数据，SP53 被置 ON。如果未找到要检索的数据，累加器中将被写入“0”。结果是一个十六进制数。

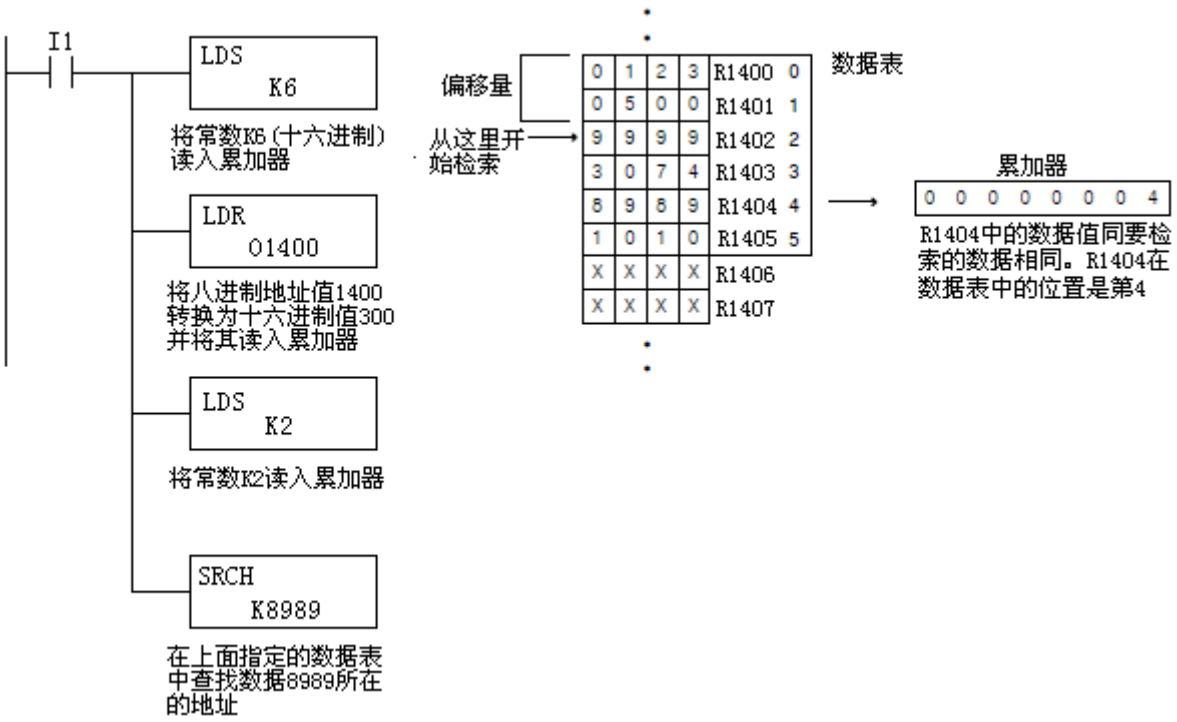
小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	0-FFFF

受影响的标志线圈	描述
SP53	表中没有与要搜索数据相同的数据时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。指令的指针从 0 开始，保存在累加器。

下面的例子中，当 I1 为 ON 时，常数 K6 被装入累加器，这个数据指定了数据表的长度，当第二个读入指令执行时，常数 K6 被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器，这是数据表的起始地址，当第三个读入指令执行时，这个地址值被压入数据堆栈的第 1 级中，第 1 级中的原有数据 K6 被压入第 2 级中。第三个读入指令将常数 K2 读入累加器，这个数据是数据表起始地址的偏移量。SRCH 指令中指定要检索的数据。如果被检索的数据等于某个寄存器中的数据，那这个寄存器的位置相对于数据表起始地址的偏差值被读入累加器中。

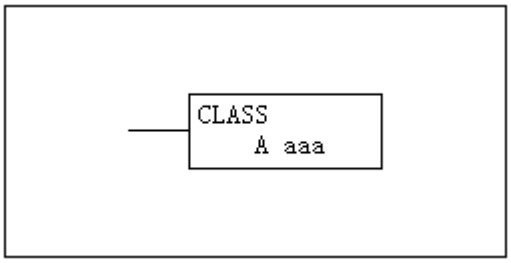


语句视图

```
LD I1
LDS K6
LDR O1400
LDS K2
SRCH K8989
```

12.9 数据级别分类指令（CLASS）

CLASS 指令用于在指定的数据表中检索第一个大于指定值的数据在数据表中的位置。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：



注意：本指令没有偏移量，与 SRCH 指令不同。

第 1 步：将要被检索的数据表的长度（最多 255 个寄存器）读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始寄存器地址读入累加器，这个数必须是十六进制数。

第 3 步：写入 CLASS 指令，指定要比较的数据。

执行结果：第一个包含比指定数据大的寄存器在数据表中的相对于起始地址的偏差值被写入累加器中。如果大于指定数据的数据没有找到，则在累加器中写入 0，结果是一个十六进制数。

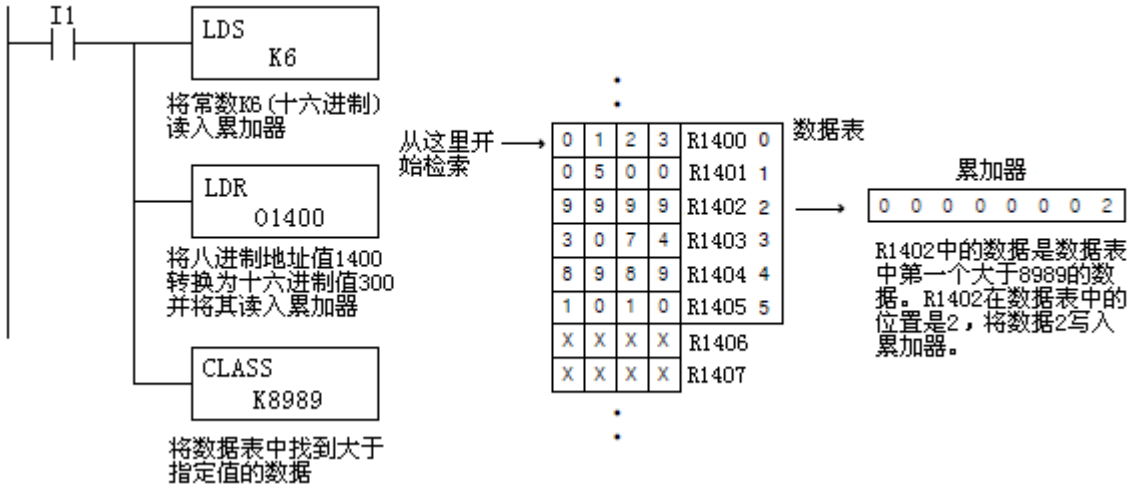
小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）
常数 K	0-FFFF

受影响的标志线圈	描述
SP53	表中没有找到比要搜索数据大的数据时 ON

注意：在其它使用相同标志线圈的指令执行前，状态标志有效。指令的指针从 0 开始，保存在累加器。

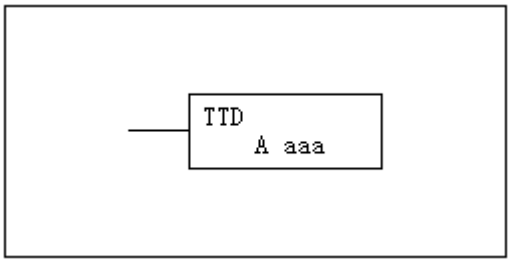
下面的例子中，当 I1 为 ON 时，常数 K6 被读入累加器，这个数据指定了数据表的长度，当第二个读入指令执行时，常数 K6 被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器，这是数据表的起始地址。CLASS 指令中指定要比较的数据。如果寄存器中的数据大于要比较的数据，在这个寄存器在数据表中的相对于起始地址的偏差值被写入累加器中。如果数据表中的数据没有大于指定的数据，则在累加器中写入“0”，将 SP53 设置为 ON。



语句视图

```
LD I1
LDS K6
LDR 01400
CLASS K8989
```

12.10 指针加法取出指令（TTD）



TTD 指令将指定数据表中一个数据取出放入一个寄存器中，同时将指针值加 1。数据表的第一个寄存器存放的是一个指针值，用于指示将要被取出数据的寄存器。条件成立时，此指令每次扫描运行一次。当指针值等于数据表的最后一个地址时将复位为“1”。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器被用作指针），这个数必须是十六进制数。

第 3 步：写入 TTD 指令，指定目标寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。

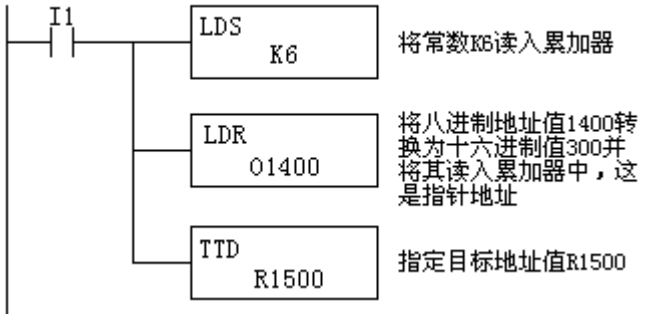
小技巧：指针中的值应设置为数据表操作开始的值。可使用特殊线圈 SP0 或一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

受影响的标志线圈	描述
SP56	当表指针等于表长度时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。指令的指针从 0 开始，当达到表长度时复位。指针复位到 1 而不是 0。

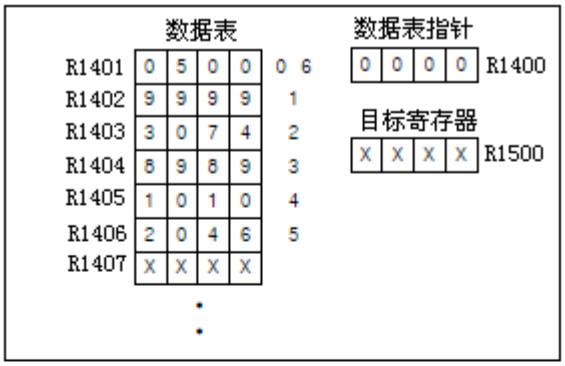
下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址，注意，R1400 被用作指针地址，不是数据源的一部分。写入 TTD 指令，指定目标寄存器地址 R1500。TTD 指令每执行一次，数据表指针值就加“1”。



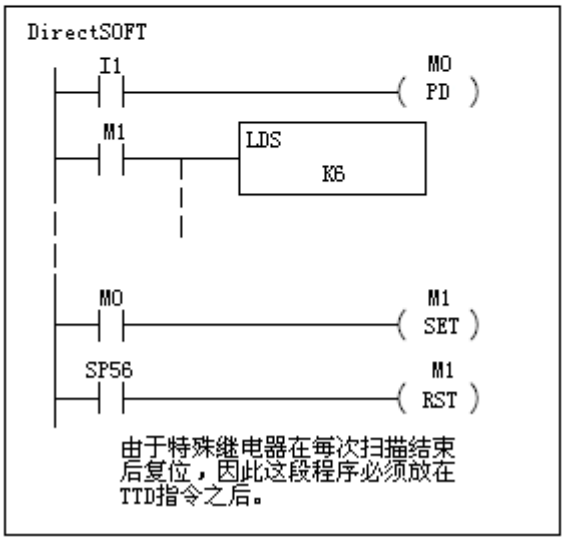
语句视图

```
LD  I1
LDS K6
LDR 01400
TTD R1500
```

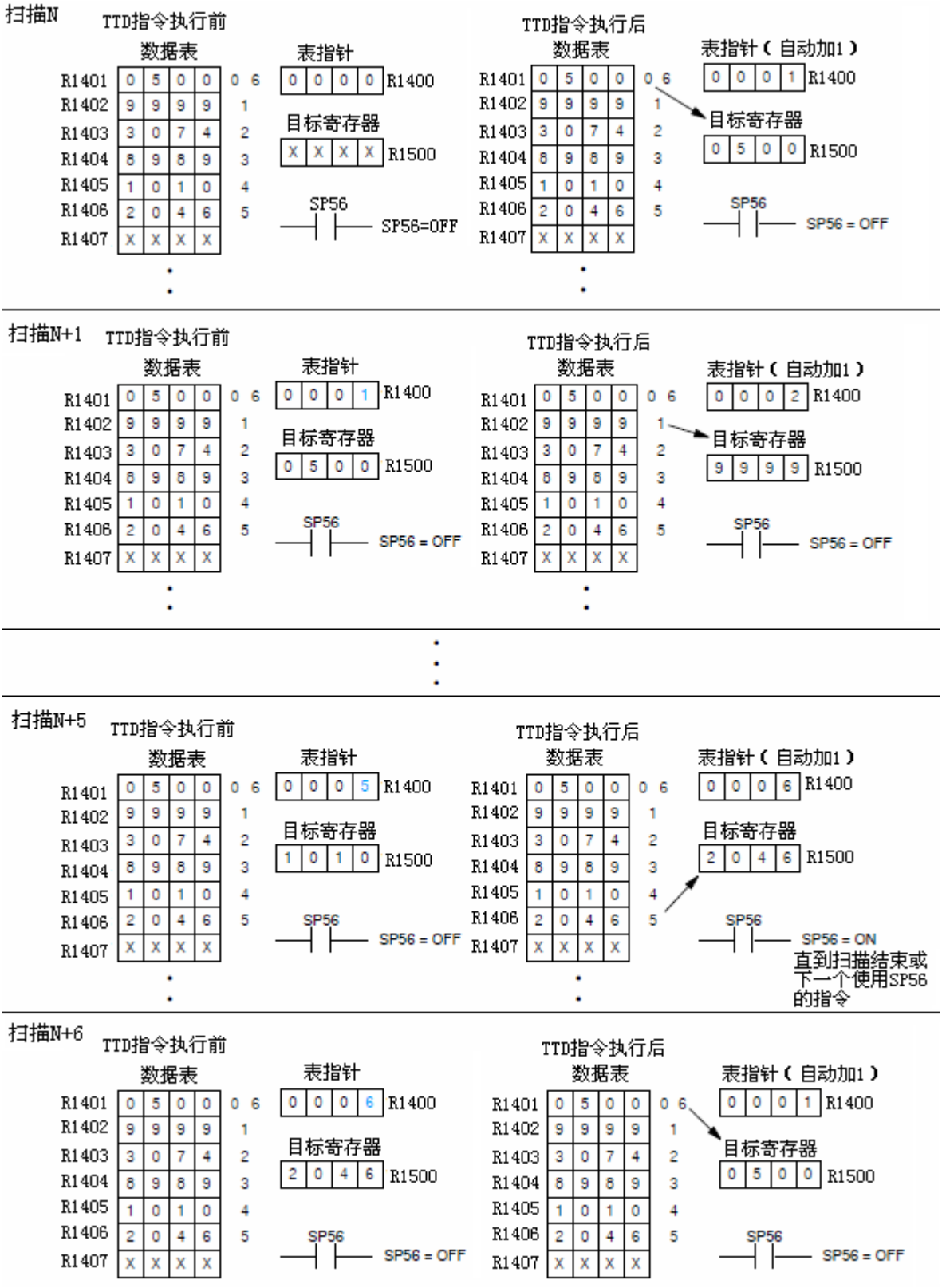
了解数据表是怎么编号的很重要，查看右边的图例，你会发现，当指针值等于 0 时，第一个寄存器地址 R1401 中的数据将被使用，而当指针值等于 6 时，R1401 中的数据也被使用。这是因为只有在 TTD 指令第一次执行时，指针值才会等于 0，第一次执行后，指针的范围就是从 1 到 6，然后再复位为 1。



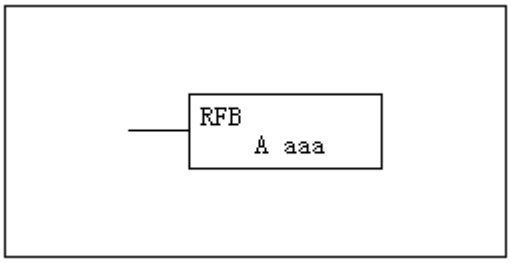
右边的例子中，使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于指针值是自动增加的，因此数据表循环也非常快，如果不想这样，可以使用右图的方法，这样当数据表循环一次后就停下来。这段程序不是必须的，可以选用。



下图演示了上页的程序例子中每次扫描后指针值如何从0-6自动循环，然后从1-6循环，注意：SP56被置为ON后，保持到本次扫描结束。



12.11 指针减法取出指令（RFB）



RFB 指令将指定数据表底部的一个数据取出放入一个寄存器中，同时将指针值减 1。数据表的第一个寄存器存放的是一个指针值，用于指示将要被取出数据的寄存器。条件成立时，此指令每次扫描运行一次。当指针值等于 0 时，指令停止运行。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器被用作指针），这个数必须是十六进制数。

第 3 步：写入 RFB 指令，指定目标寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。

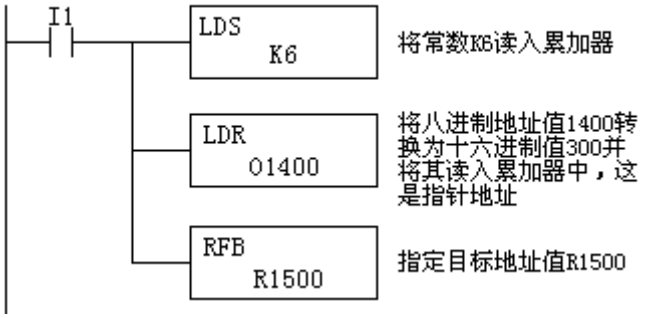
小技巧：指针中的值应设置为数据表操作开始的值。可使用特殊线圈 SP0 或一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

受影响的标志线圈	描述
SP56	当表指针等于 0 时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。指令的指针可以从数据表的任意位置开始，指针不是自动设置，需要在程序中给指针赋值。

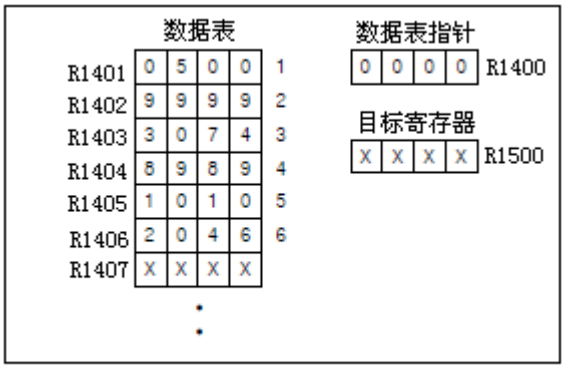
下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址，注意，R1400 被用作指针地址，不是数据源的一部分。写入 RFB 指令，指定目标寄存器地址 R1500。RFB 指令每执行一次，数据表指针值就减“1”。



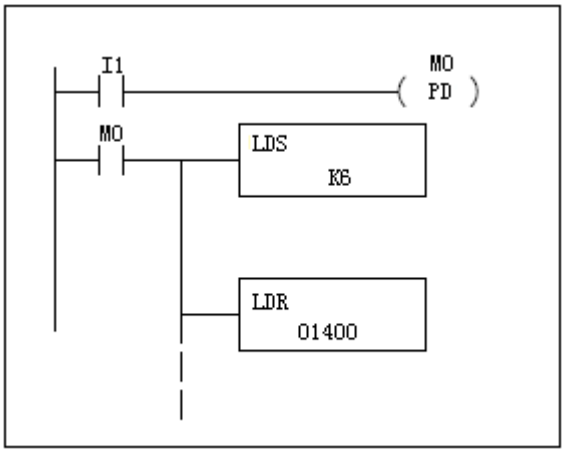
语句视图

```
LD I1
LDS K6
LDR 01400
RFB R1500
```

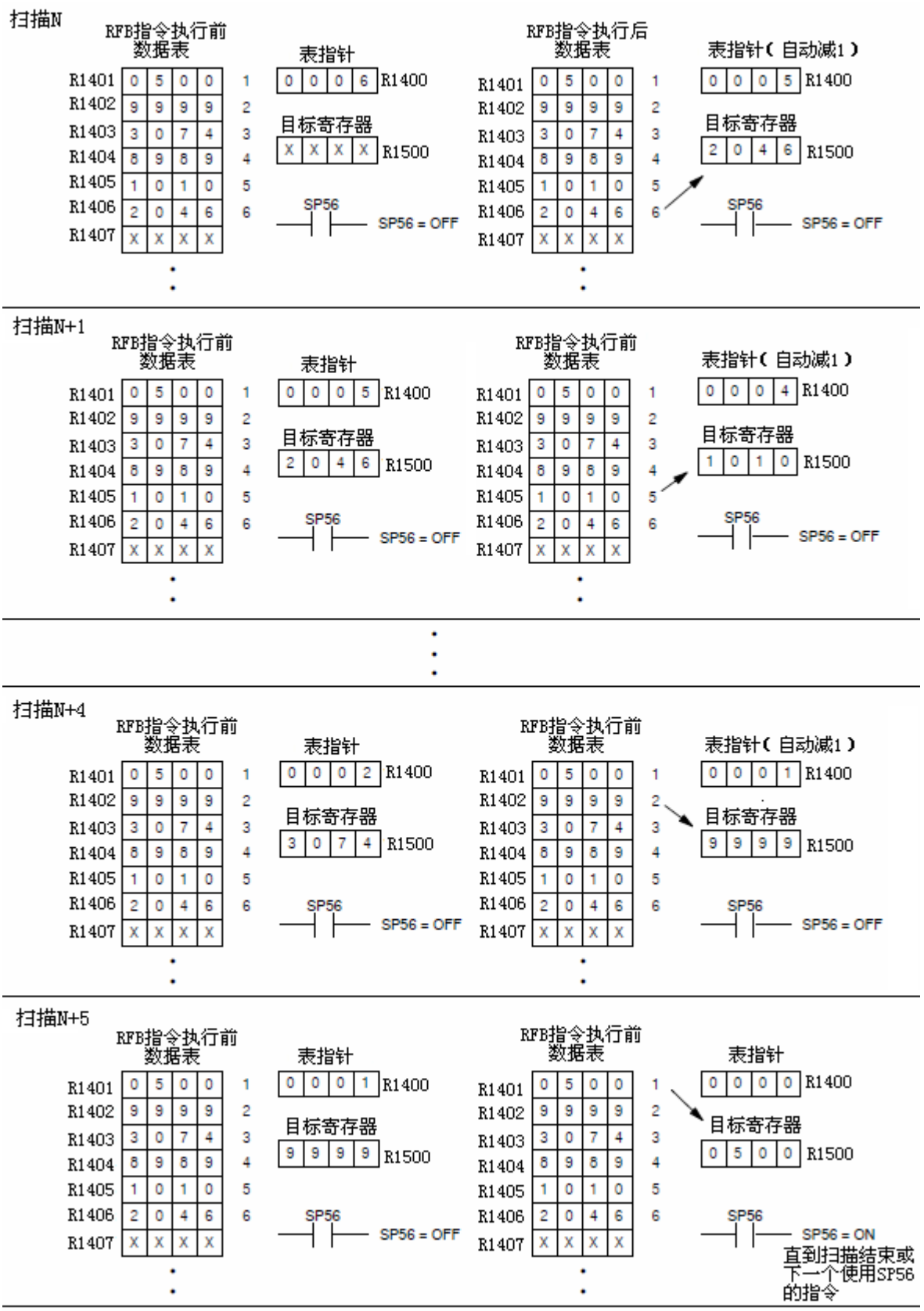
了解数据表是怎么编号的很重要，查看右边的图例，你会发现，当指针值等于 1 时，第一个寄存器地址 R1401 中的数据将被使用，当指针值等于 2 时，R1402 中的数据将被使用，以此类推。



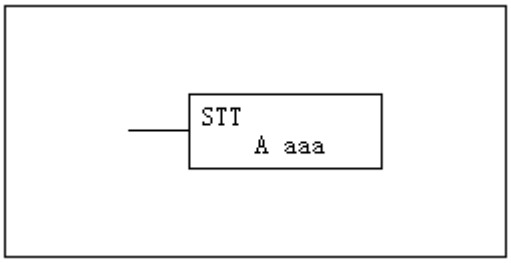
右边的例子中，使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于指针值是自动减少的，因此数据表循环也非常快，如果不想这样，可以如右图的所示的方法使用一次扫描输出指令，这样当输入由 OFF→ON 时，数据仅读入一次。



下图演示了上页的程序例子中每次扫描后指针值如何从 6 到 0 自动循环，注意：SP56 被置为 ON 后，保持到本次扫描结束。



12.12 指针加法存入指令（STT）



STT 指令将一个常数或指定寄存器中的数据存入一个指定的数据表中，同时将数据表指针加 1。当指针值到达数据表的末端，就复位为“1”。数据表的第一个寄存器存放的是指针值，用于指示将要被存入数据的寄存器。条件成立时，此指令每次扫描运行一次。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器被用作指针），这个数必须是十六进制数。

第 3 步：写入 STT 指令，指定数据源寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。

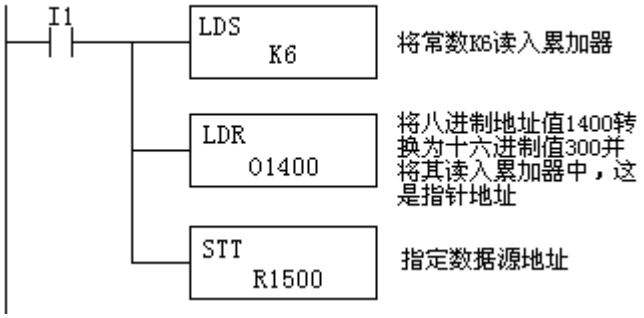
小技巧：指针中的值应设置为数据表操作开始的值，而且它应小于等于数据表的长度。比如，如果数据表长度是 6 字长，那么指针的范围应为 0-6，如果超出这个范围，数据将不移动。可使用一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	0-FFFF

受影响的标志线圈	描述
SP56	当表指针等于表长度时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。指令的指针从 0 开始，当指针值达到表长度时，复位为 1。

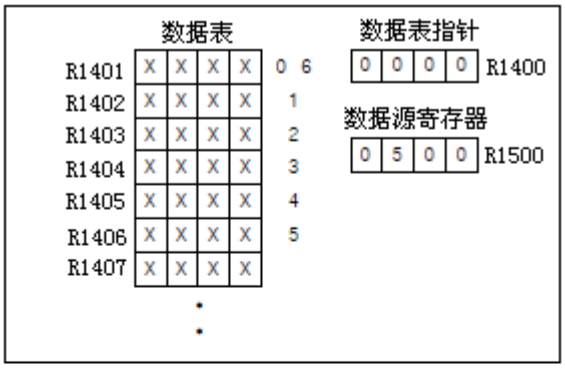
下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址，注意，R1400 被用作指针地址。写入 STT 指令，指定数据源寄存器地址 R1500。STT 指令每执行一次，数据表指针值就加“1”。



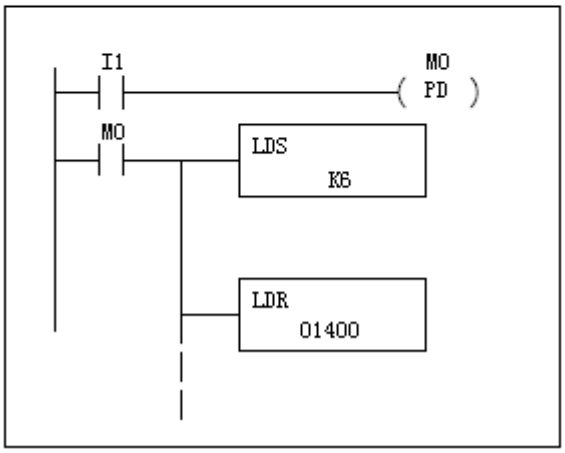
语句视图

```
LD  I1
LDS K6
LDR 01400
STT R1500
```

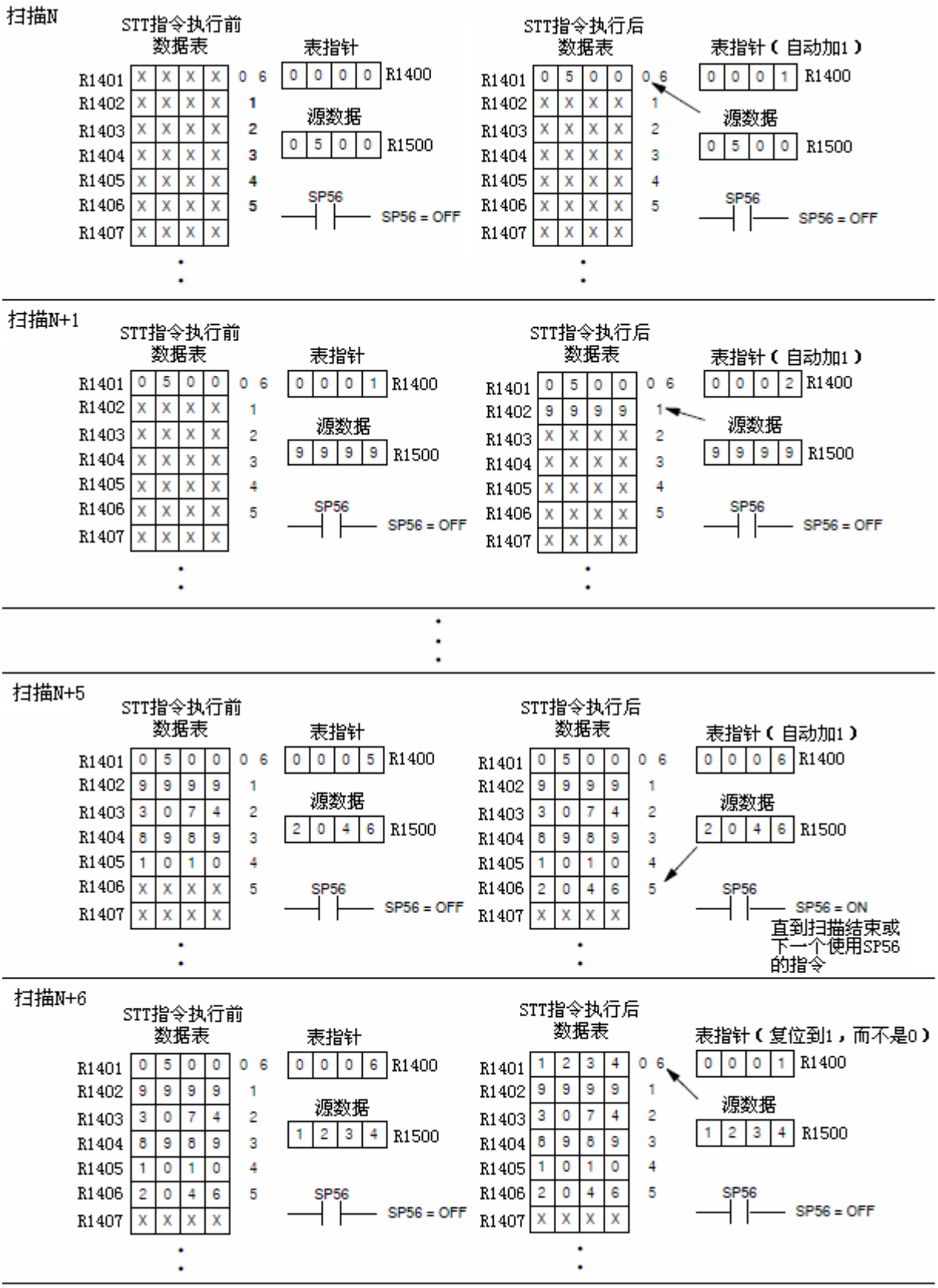
了解数据表是怎么编号的很重要，查看右边的图例，你会发现，当指针值等于 0 时，源数据存入第一个寄存器 R1401 中，而当指针值等于 6 时，源数据也存入 R1401 中。这是因为只有在 STT 指令第一次执行时，指针值才等于 0，第一次执行后，指针的范围就是从 1 到 6，然后再复位为 1。



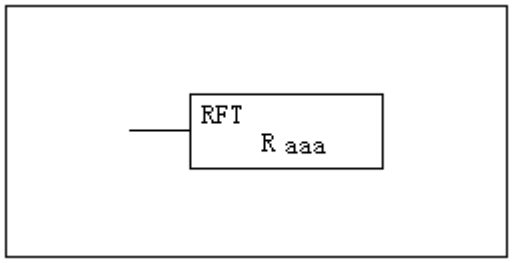
右边的例子中，使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于指针值是自动增加的，因此源数据存入数据表也非常快，如果不想这样，可以使用右图的方法，使用一次扫描输出指令，这样当输入由 OFF→ON 时，数据仅读入一次。



下图演示了上页的程序例子中每次扫描后指令的执行结果，指针值如何从 0 到 6 自动循环，然后从 1 到 6 循环，注意：SP56 被置为 ON 后，保持到本次扫描结束。本例假设有另外一段程序用来改变数据源 R1500 中的数据，并且其执行优先级高于 STT 指令。



12.13 堆栈上托取出指令（RFT）



RFT 指令将指定数据表中的一个数据取出放入一个寄存器中。当数据表的一个数据移出后，其它数据都向上移动一个地址。

数据表的第一个寄存器中存放的是数据表的长度，RFT 指令每运行一次，数据表的长度就减 1。如果数据表的长度等于 0 或大于数据表长度的最大值（由数据堆栈的第 1 级中的数据指定），指令将不执行，SP56 将被置位为 ON。

条件成立时，此指令每次扫描运行一次。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器中存放的是数据表的长度计数值），这个数必须是十六进制数。

第 3 步：写入 RFT 指令，指定目标寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。

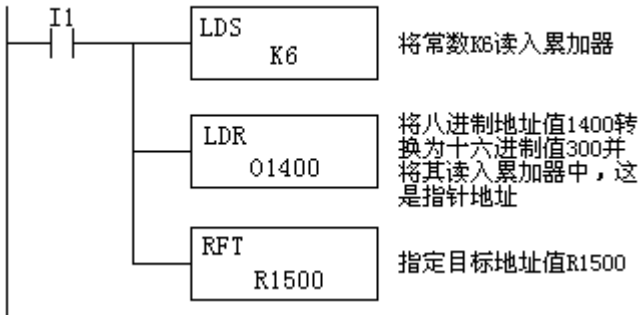
小技巧：数据表长度计数值应设置为指示数据表操作的起始点，而且它应小于等于数据表的长度。比如，如果数据表长度是 6 字长，那么数据表长度的计数值的范围应为 1-6，如果超出这个范围或等于 0，数据将不移动。可使用一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）

受影响的标志线圈	描述
SP56	当表指针等于 0 时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。指令的指针可以从数据表的任意位置开始，指针不是自动设置，需要在程序中给指针赋值。

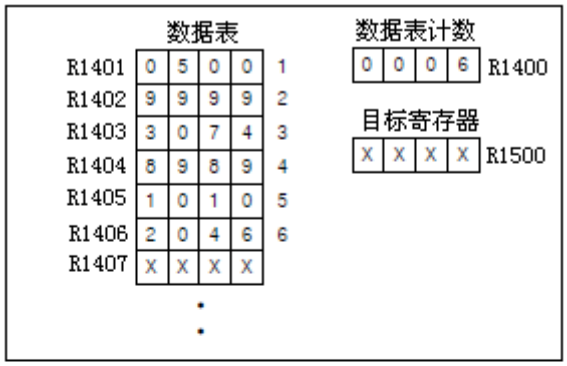
下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址。写入 RFT 指令，指定目标寄存器地址 R1500。RFT 指令每执行一次，数据表长度值就减“1”。



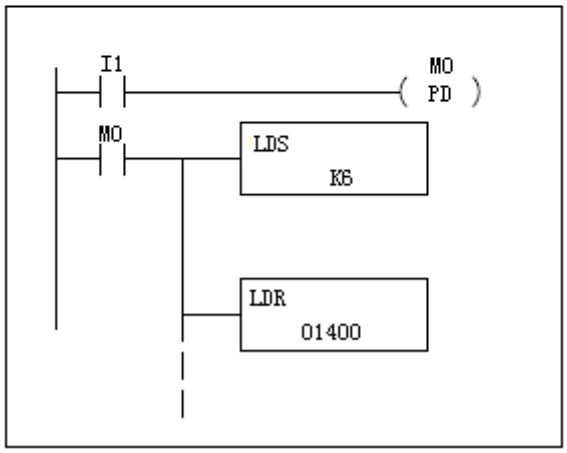
语句视图

```
LD  I1
LDS K6
LDR 01400
RFT R1500
```

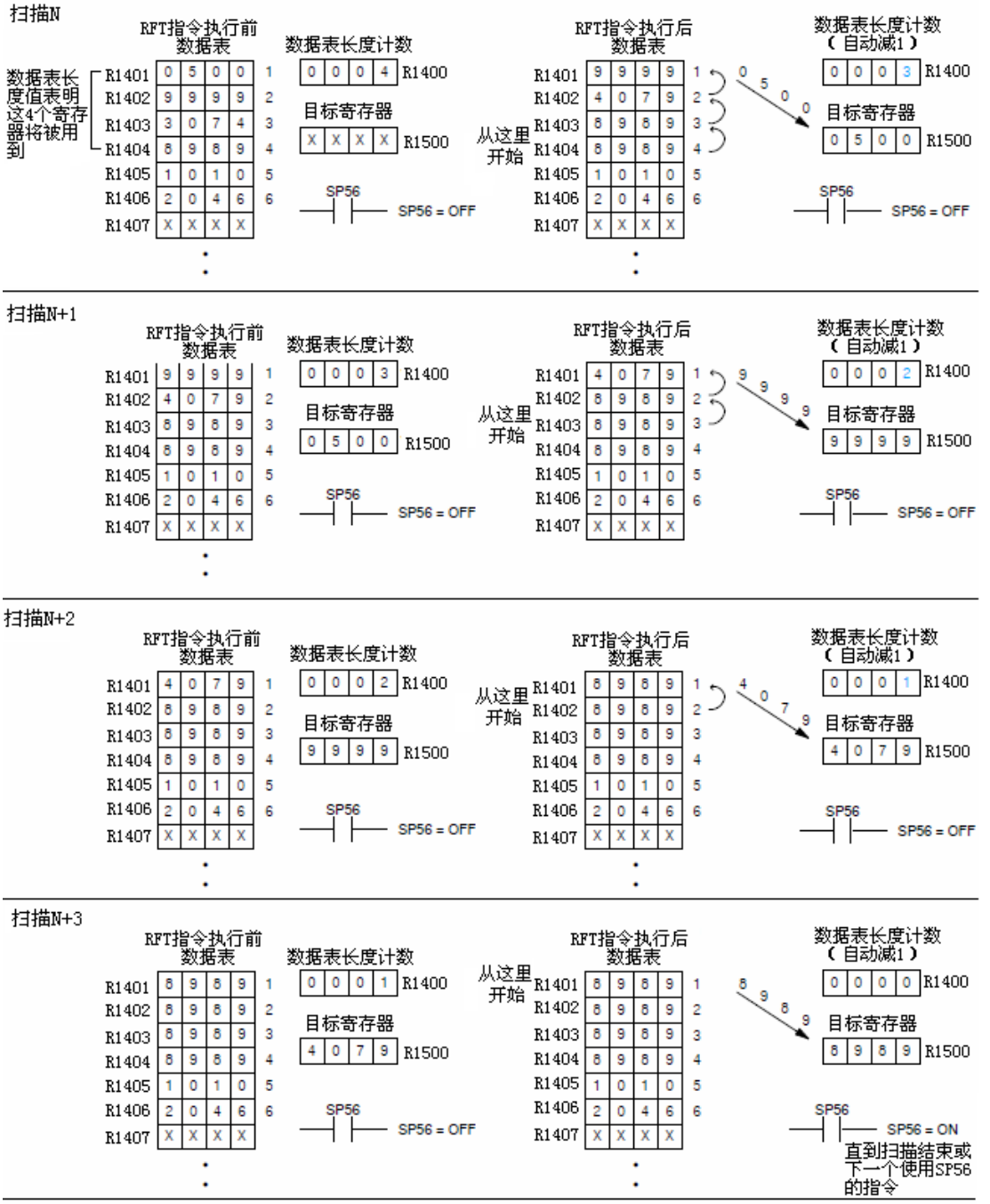
了解数据表是怎么编号的很重要，查看右边的图例，你会发现，数据表编号从数据表顶端开始。例如，如果数据表长度计数值从 6 开始，则指令执行时，所有 6 个地址的值都会变动。



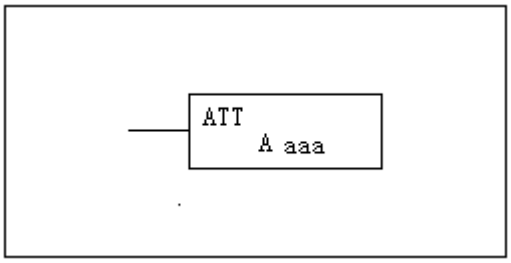
右边的例子中，使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于数据表长度计数值是自动减少的，因此数据表数据移出也非常快，如果不想这样，可以使用右图的方法，使用一次扫描输出指令，这样当输入由 OFF→ON 时，仅取出一个数据。



下图演示了上页的程序例子中每次扫描后指令的执行结果，本例中，设置最初的数据表长度计数值为 4（注意，可将数据表长度计数值设置为任何不大于数据表长度的数值）。数据表长度计数值从 4 到 0 逐渐减少，指令每执行一次，就减少“1”。注意，数据表的最后两个位置，5 和 6，没有移动，还有，当长度计数值等于 0 时，SP56 被置为 ON 后，保持到本次扫描结束。



12.14 堆栈下推存入指令（ATT）



ATT 指令将一个指定寄存器中的数据存入一个指定的数据表中，数据表的其它数据均向下移动一个地址。

条件成立时，此指令每次扫描运行一次。此指令执行之前，需要将功能参数分别读入数据堆栈的第 1 级和累加器中，下面是实现此指令功能的步骤：

第 1 步：将数据表的长度读入数据堆栈的第 1 级中，这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始寄存器地址读入累加器（注意，起始寄存器中存放的是数据表计数值），这个数必须是十六进制数。

第 3 步：写入 ATT 指令，指定源数据寄存器地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

小技巧：当条件成立时，此指令每次扫描执行一次。如果不想指令每次扫描都执行，可对条件使用一次扫描输出指令（PD）。

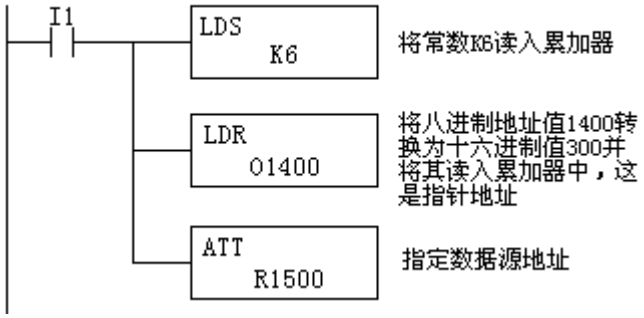
小技巧：数据表计数值应设置为指示数据表操作的起始点，而且它应小于等于数据表的长度。比如，如果数据表长度是 6 字长，那么数据表计数值的范围应为 1-6，如果超出这个范围或等于 0，数据将不移动。可使用一次扫描输出指令（PD），只在一次扫描中设置这个指针值而不影响指令的执行。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	0-FFFF

受影响的标志线圈	描述
SP56	当表指针等于表长度时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。指令的指针可以从数据表的任意位置开始，指针不是自动设置，需要在程序中给指针赋值。

下面的例子中，当 I1 为 ON 时，第一个读入指令将常数 K6 读入累加器中，此值指定了数据表的长度，当第二个读入指令执行时，此值被压入数据堆栈的第 1 级中。第二个读入指令将八进制地址值 1400（R1400）读入累加器中，R1400 是数据表的起始地址，里面存放的数据表计数值。写入 ATT 指令，指定源数据寄存器地址 R1500。ATT 指令每执行一次，数据表计数值就加“1”。



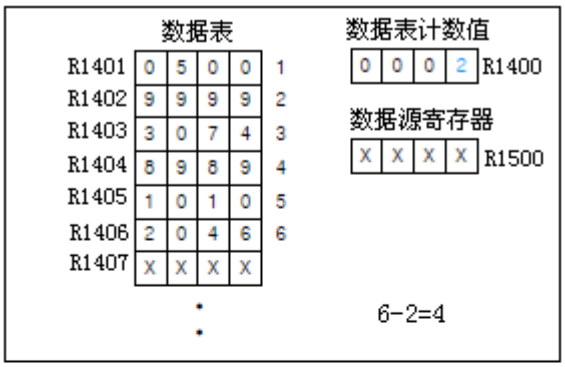
语句视图

```
LD  I1
LDS K6
LDR 01400
ATT R1500
```

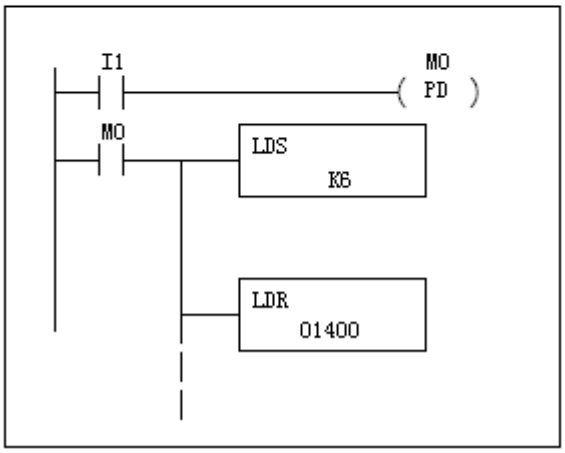
对于 ATT 指令，数据表计数值确定了指令停止执行之前将数据读入数据表的次数，因此，数据表计数值有助于理解系统是如何运用它来控制指令执行的。

右边的例子中，如果数据表计数值被设置为 2，数据表长度是 6 字，那么指令停止执行前还要执行 4 次，很容易计算出：

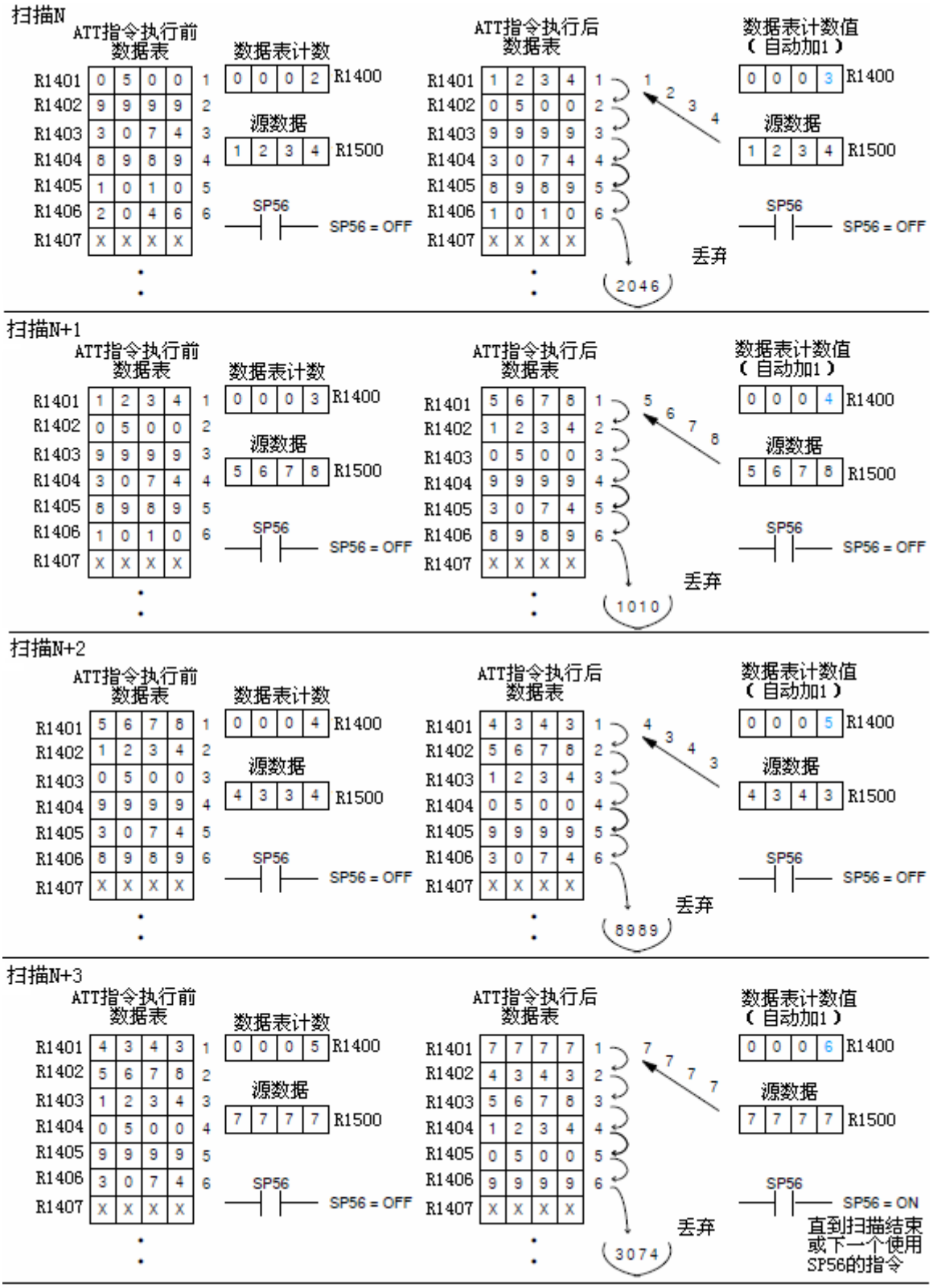
数据表长度－数据表计数值=执行次数



右图的例子中使用了一个常开输入点（I1）来控制执行。CPU 扫描速度非常快，由于数据表计数值是自动增加的，因此源数据存入数据表也非常快，如果不想这样，可以使用右图的方法，使用一次扫描输出指令，这样当输入由 OFF→ON 时，数据仅读入一次。



下图演示了上页的程序例子中每次扫描后指令的执行结果，数据表计数值的初始值被设置为 2，指令执行后它的值自动从 2 增加到 6。注意，当数据表计数值等于 6，也就是等于数据表长度时，SP56 被设置为 ON。本例假设有另外一段程序用来改变数据源 R1500 中的数据，并且其执行优先级高于 ATT 指令。



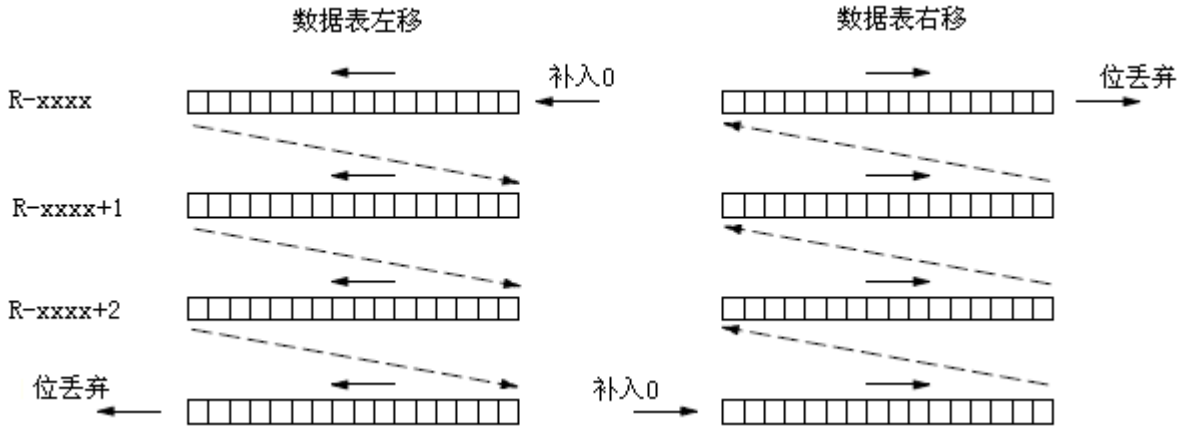
12.15 表左移指令（TSHFL）

TSHFL 指令将 R 寄存器数据表中的所有位左移指定位数。

12.16 表右移指令（TSHFR）

TSHFR 指令将 R 寄存器数据表中的所有位右移指定位数。

下面的说明同时适用于 TSHFL 指令和 TSHFR 指令，数据表就是一系列的 R 寄存器。这两个指令通过整个数据表连续移位。位移出字的边缘，进入相邻字的另一端中。在数据表的两端，位要么被丢弃，要么在数据表中补入 0。下图中数据表是 4 字长。



第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 1 级中。这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始地址读入累加器中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数据。

第 3 步：写入 TSHFL 和 TSHFR 指令，指定要移动的位数。这个数必须是八进制。

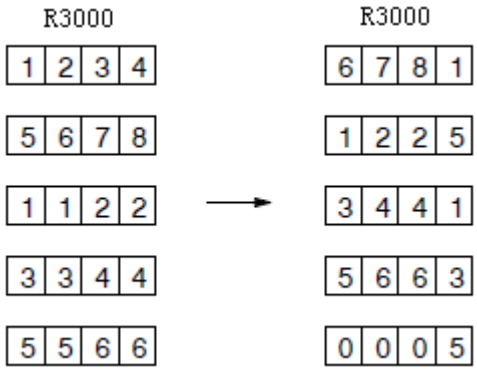
小技巧：每个 R 寄存器包含 16 位，因此，数据表的第 1 个寄存器位号范围是 0-17（八进制）。如果想移动 20 位，八进制就是 24 位。当指定要移动的位数超出数据表的位数时，SP53 将被置为 ON。如果最后移出的位为“1”，SP67 将被置为 ON。

操作数类型		D4-454 范围
	A	aaa
八进制数	0	0-7777

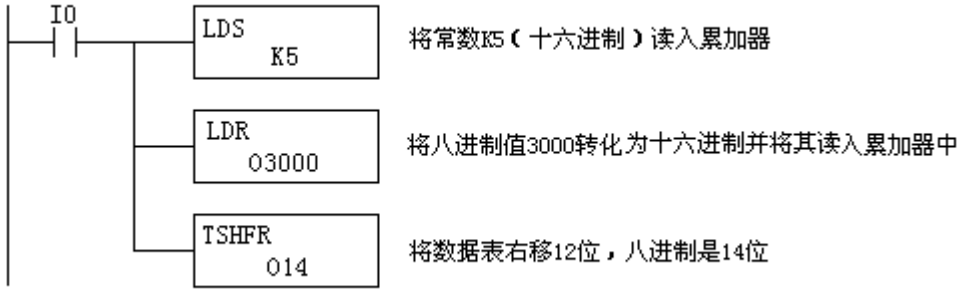
受影响的标志线圈	描述
SP56	当指定要移动的位数超出数据表的位数时 ON
SP67	当最后移出的位为 1 时 ON

注意：在本次扫描结束前或其它使用相同标志线圈的指令执行前，状态标志有效。

右边的例子中，包含的是 BCD 数。假设要将数据表右移 3 个 BCD 字（12 位）。将 12 转换为八进制数 14。使用 TSHFR 指令并指定移动位数是 14，运行结果见右图。注意到 2-3-4 三个数字被丢弃，底部补入了三个 0。



下面的例子中假设 R3000-R3004 中的数据就如上图所示。使用 I0 来触发 TSHFR 指令。首先，将数据表长度值（5 字）读入堆栈的第 1 级中。其次，将数据表的起始地址读入累加器。由于 R3000 是一个八进制数，我们使用 LDR 指令将其转换为十六进制数。最后，写入 TSHFR 指令，并指定要移动的位数，十进制数是 12，八进制数是 14。

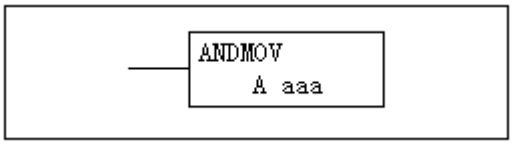


语句视图

```
LD I0
LDS K5
LDR 03000
TSHFR 014
```

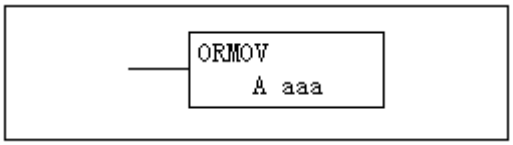
12.17 ACC 逻辑与传送指令（ANDMOV）

ANDMOV 指令将一个数据表中的每个字同累加器中的数据逻辑与运算，然后将运算结果传送到指定地址。



12.18 ACC 逻辑或传送指令（ORMOV）

ORMOV 指令将一个数据表中的每个字同累加器中的数据逻辑或运算，然后将运算结果传送到指定地址。



12.19 ACC 逻辑异或传送指令（XORMOV）

XORMOV 指令将一个数据表中的每个字同累加器中的数据逻辑异或运算，然后将运算结果传送到指定地址。

下面的说明同时适用于 ANDMOV 指令、ORMOV 和 XORMOV 指令，数据表就是一系列的 R 寄存器。这些指令将累加器中的内容与每个字进行逻辑运算，然后将运算结果传送到指定的地址中。

第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 2 级中。这个数必须是十六进制数，范围是 0-FF。

第 2 步：将数据表的起始地址读入数据堆栈的第 1 级中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数据。

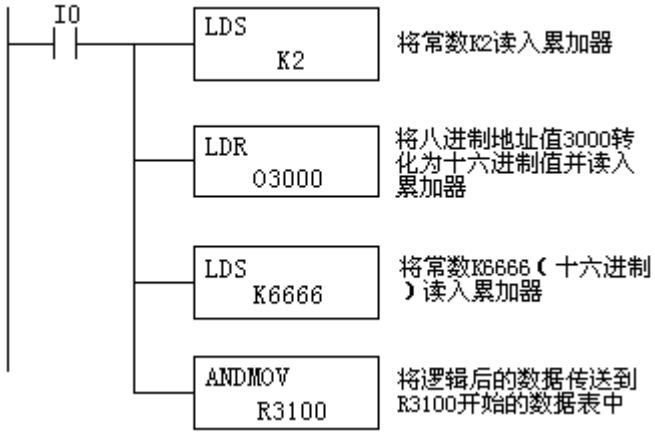
第 3 步：将要同数据表中数据逻辑运算的 BCD/十六进制数读入累加器。

第 4 步：写入 ANDMOV、ORMOV、XORMOV 指令，指定目标数据表地址，这个新数据表长度自动等于原始数据表。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

右边的例子中，将 R3000 开始的两字数据表同 K6666 逻辑与并将结果传送到以 R3100 开头的数据表中。

下面的程序完成了上例中的 ANDMOV 操作，假设 R3000-R3001 中的数据已经存在。首先，将数据表长度值（2 字）读入累加器中。其次，将数据表的起始地址读入累加器中，使用 LDR 指令。然后，把要与数据表数据逻辑与的数据读入累加器。在 ANDMOV 指令中，指定目标数据表的起始地址 R3100。

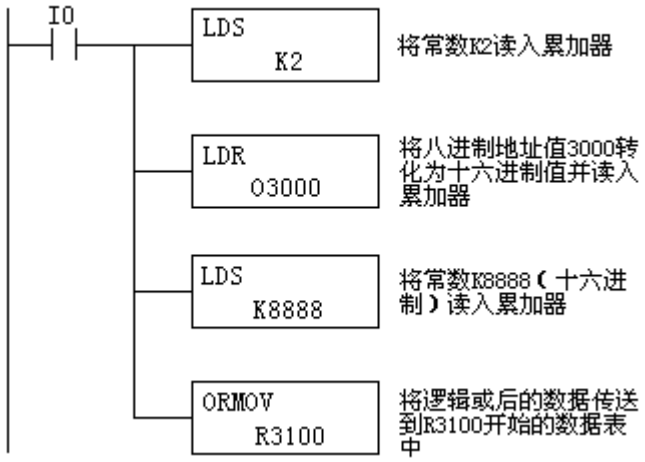


语句视图

```
LD I0
LDS K2
LDR O3000
LDS K6666
ANDMOV R3100
```

右边的例子中，将 R3000 开始的两字数据表同 K8888 逻辑或并将结果传送到以 R3100 开头的数据表中。

下面的程序完成了上例中的 ORMOV 操作，假设 R3000-R3001 中的数据已经存在。首先，将数据表长度值（2 字）读入累加器中。其次，将数据表的起始地址读入累加器中，使用 LDR 指令。然后，把要与数据表数据逻辑或的数据读入累加器。在 ORMOV 指令中，指定目标数据表的起始地址 R3100。



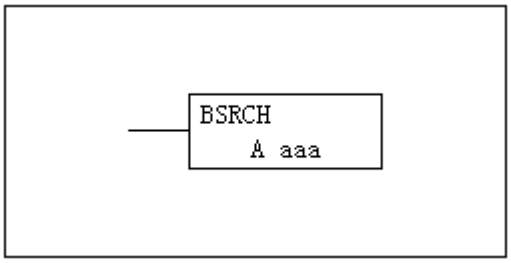
语句视图

```
LD I0
LDS K2
LDR O3000
LDS K8888
ORMOV R3100
```

右边的例子中，将 R3000 开始的两字数据表同 K3333 逻辑异或并将结果传送到以 R3100 开头的数据表中。

12.20 多字节数据检索指令（BSRCH）

BSRCH 指令在一个数据表中检索一个数据块。指令执行前，要将功能参数分别读入堆栈的第 1 级、第 2 级、第 3 级和累加器中。如果要检索的数据块被找到，它的起始地址将被存放在累加器中。如果没有找到，SP53 将被置位为 ON。



操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
指针	P	所有（附录 1）

受影响的标志线圈	描述
SP53	当表指针出错时 ON

下面是实现这个指令功能的步骤：

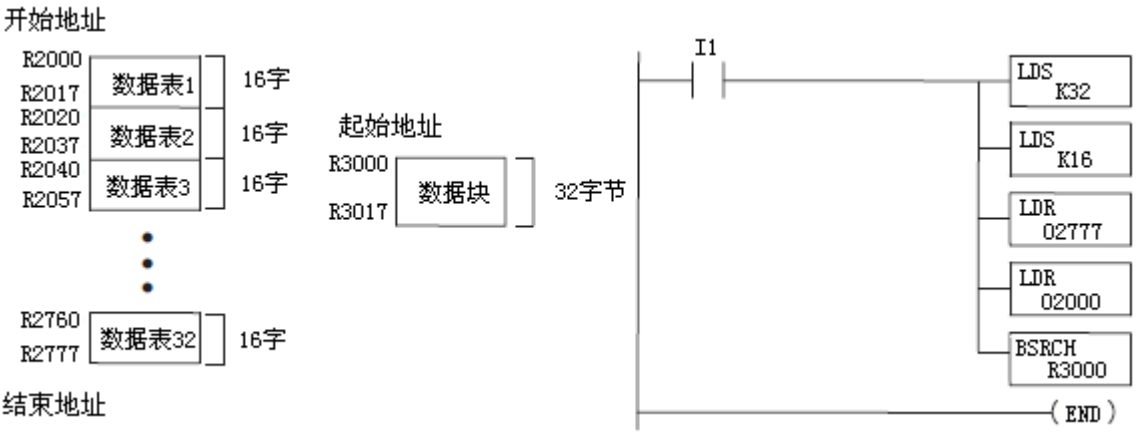
第 1 步：读入要检索的数据块字节数，这个数必须是十六进制数，范围是 0-FF。

第 2 步：读入每个数据表的长度（字数），BSRCH 指令将检索相邻的多个数据表，这个参数必须是十六进制数，范围是 0-FF。

第 3 步：读入检索对象数据表的结束地址。这个参数必须是十六进制数，可以使用 LDR 指令将八进制地址值转化为十六进制数据。

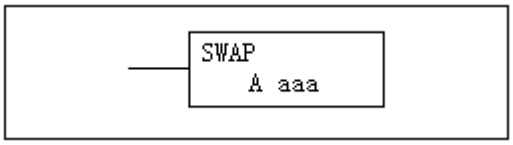
第 4 步：读入检索对象数据表的开始地址。这个参数必须是十六进制数，可以使用 LDR 指令将八进制地址值转化为十六进制数据。

第 5 步：写入 BSRCH 指令，指定要检索数据块的起始地址。



12.21 交换指令（SWAP）

SWAP 指令将两个相同长度的数据表数据互换。



下面是实现此指令功能的步骤：

第 1 步：将数据表的长度（R 寄存器的个数）读入数据堆栈的第 1 级中。这个数必须是十六进制数，范围是 0-FF，注意，两个数据表的长度必须相等。

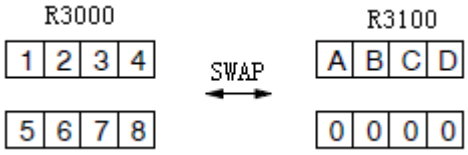
第 2 步：将第一个数据表的起始地址读入累加器中，这个数必须是十六进制数，可使用 LDR 指令将八进制地址值转换为十六进制数。

第 3 步：写入 SWAP 指令，指定第二个数据表的起始地址。

小技巧：数据交换只能执行一个扫描周期，如果指令在多个连续的扫描周期内执行，就会很难知道数据表中的数据。因此，要注意 SWAP 指令只能执行一个扫描周期。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

右边的例子中，将以 R3000 开始的 2 字数据表同 R3100 开始的 2 字数据表进行数据交换，下面是程序。



下面的程序使用了一个 PD 触点（I0 由 OFF 到 ON 时接通一个扫描周期）。首先，将数据表的长度值（2 字）读入累加器，然后将第一个数据表的起始地址读入累加器，使用 LDR 指令，将八进制地址值转换为十六进制数据。注意谁是第一个数据表没有关系，因为 SWAP 指令执行结果是一样的。

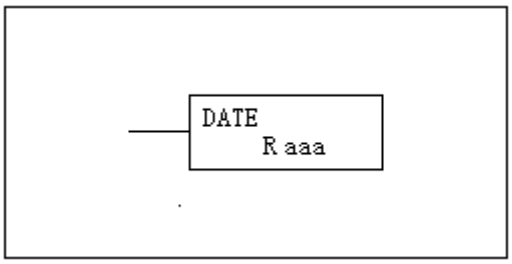
语句视图

```
LDPD I0
LDS K2
LDR O3000
SWAP R3100
```

第 13 章 时钟/日历指令

13.1 日期设定指令（DATE）

DATE 指令用来设定 CPU 的日期，指令需要两个连续的 R 寄存器地址来设置日期。如果指定地址中的值无效，则日期将不进行设定。当前日期可从 4 个连续的 R 寄存器地址（R7771-R7774）中读出。

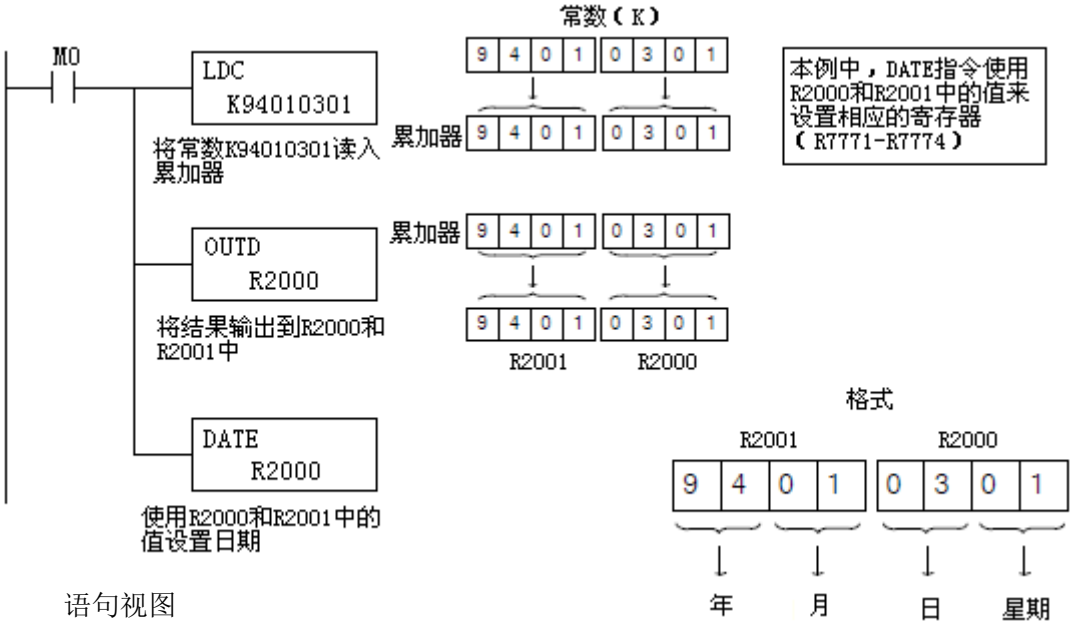


日期	范围	R 寄存器地址（BCD）（只读）
年	0-99	R7774
月	1-12	R7773
日	1-31	R7772
星期	0-06	R7771

星期：0=星期日，1=星期一，2=星期二，3=星期三，4=星期四，5=星期 5，6=星期六

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

下面的例子中，当 M0 为 ON 时，常数 K94010301 被读入累加器中，M0 应是一个一次输出(PD)指令产生的触点。将累加器中的数据写入到 R2000 和 R2001 中。DATE 指令使用 R2000 中的数据来设置日期。

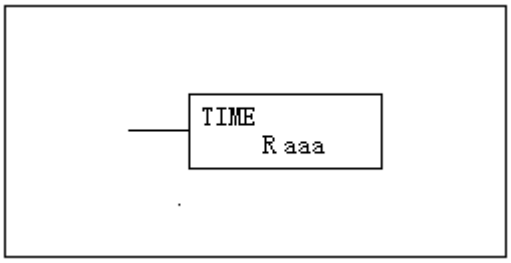


语句视图

```
LD M0
LDC K94010301
OUTD R2000
DATE R2000
```

13.2 时钟设定指令（TIME）

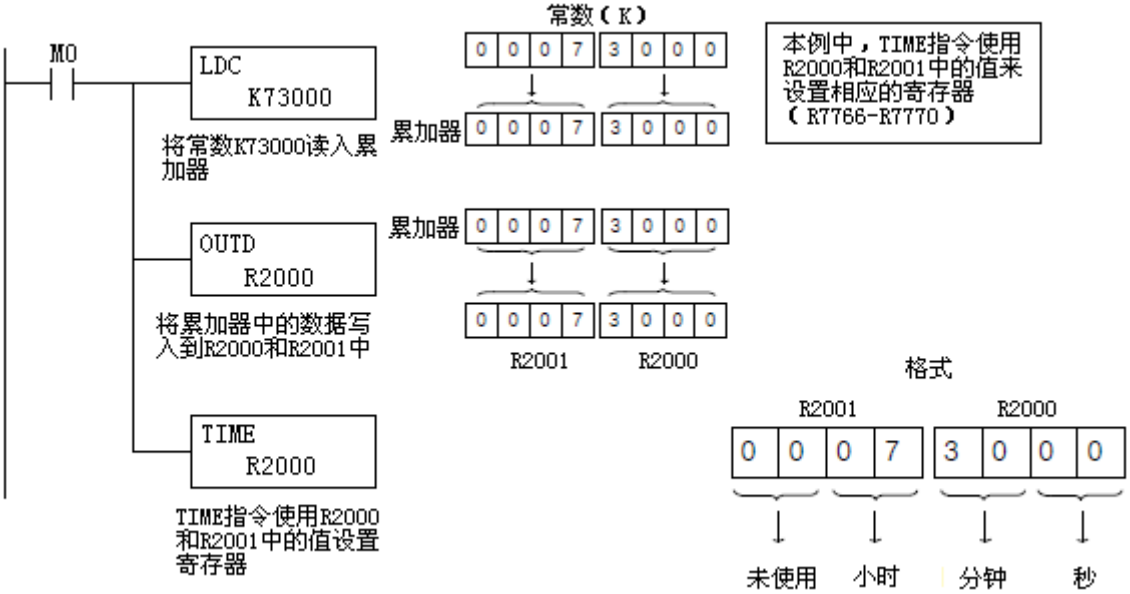
TIME 指令用来设置时间（24 小时时钟）。指令需要两个连续的寄存器地址来设置时间。如果指定地址中的值无效，则时间将不进行设定。当前时间可从 R7747 和 R7766-R7770 中读出。



日期	范围	R 寄存器地址（BCD）（只读）
1/100s（10ms）	0-99	R7747
秒	0-59	R7766
分	0-59	R7767
小时	0-23	R7770

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

下面的例子中，当 M0 为 ON 时，常数 K73000 被读入累加器，OUTD 指令将累加器中的数据写入 R2000 和 R2001 中。TIME 指令使用 R2000 和 R2001 中的数据来设置时间。



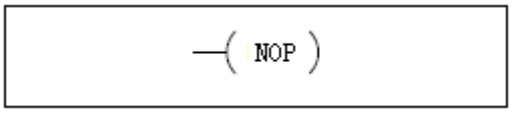
语句视图

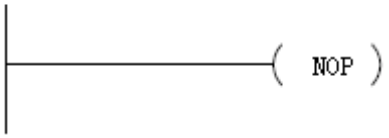
```
LD M0
LDC K73000
OUTD R2000
TIME R2000
```

第 14 章 CPU 控制指令

14.1 空操作指令（NOP）

NOP 指令是一个空操作指令。用该指令可在程序中设置空间，对以后追加程序很方便。

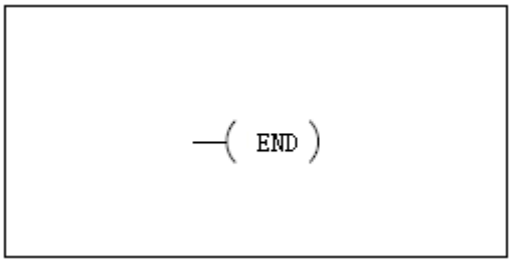




语句视图
NOP

14.2 程序结束指令（END）

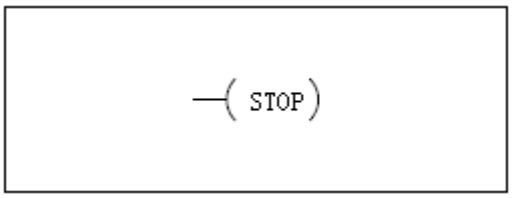
END 指令是主程序的扫描终点标志。END 指令要放在主程序的最后。如果主程序后没有 END 指令，将会产生错误，CPU 将不会进入运行模式。数据标记、子程序和中断程序要写在 END 指令之后。END 指令是无条件指令，前面不能添加触点。



语句视图
END

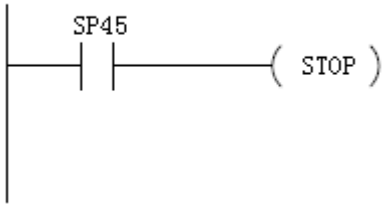
14.3 扫描停止指令（STOP）

STOP 指令将 CPU 强制从运行模式转换为编程(停止)模式。该指令通常通过一个停机条件（比如 I/O 模块错误），停止 PLC 运行。

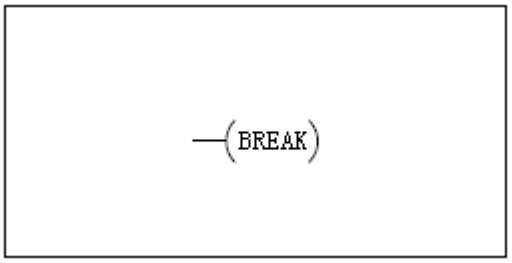


下面的例子中，当 SP45 被置为 ON，表明有一个 I/O 模块错误，CPU 将停止运行，转换为编程模式。

语句视图
LD SP45
STOP

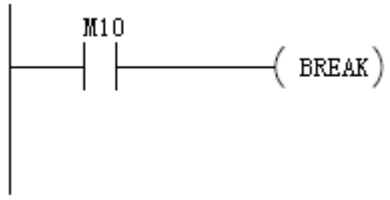


14.4 程序暂停指令（BREAK）



BREAK 指令将 CPU 的操作模式从运行转换为编程测试模式。该指令通常用于辅助应用程序调试。BREAK 指令允许 R 寄存器和映像寄存器中的数据保持不变，这些数据在 STOP 指令执行或是模式由运行转换为编程时通常被清空。

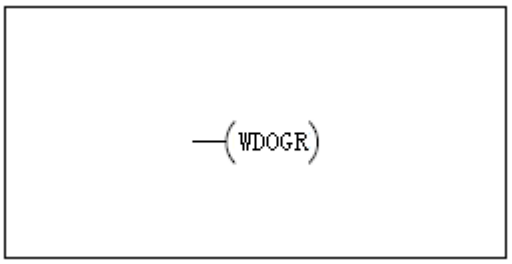
下面的例子中，当 M10 为 ON 时，CPU 将停止运行，运行模式开关将被置于编程测试模式。



语句视图

```
LD M10
BREAK
```

14.5 监视定时器复位指令（WDOGR）



WDOGR 指令复位监视定时器的时间。监视定时器的缺省设置是 200ms，一般的扫描时间都不会超过 200ms，但不是不可能超过。如果程序中有 For/next 回路、子程序、中断程序和表指令等，扫描时间有可能超过 200ms。

如果扫描时间超过监视定时器中设定的时间，将产生一个软件时间溢出错误（E003）并且 CPU 将进入编程模式。如果扫描时间很长，在程序中加入 WDOGR 指令是很重要的，在扫描时间超过监视定时器设置的时间之前，必须执行 WDOGR 指令。

如果扫描时间总是超过监视定时器设定的时间，则需要加长监视定时器的设定值。KPPsoft 编程软件中，“PLC”菜单中可直接改变设置。

下面的例子中，当 WDOGR 指令执行时，CPU 扫描时间被复位。

语句视图

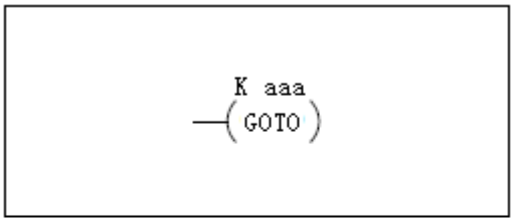
```
WDOGR
```

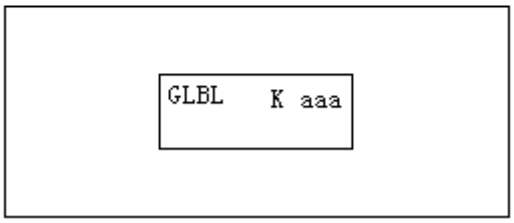
第 15 章 程序控制指令

15.1 跳转指令（GOTO）

15.2 跳转目标指令（GLBL）

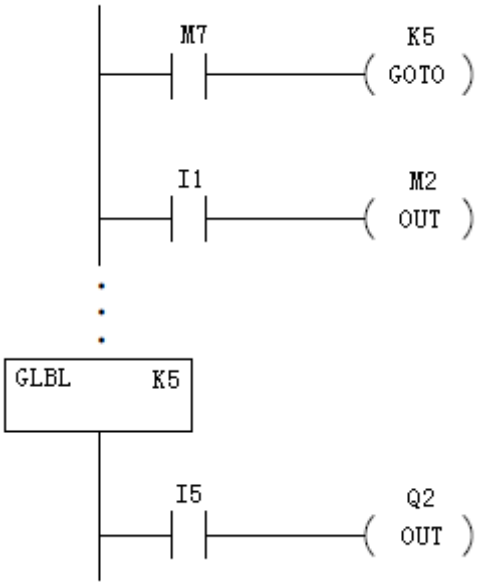
GOTO/GLBL 指令将这两个指令之间的所有指令跳过。GOTO 指令和 GLBL 指令的标号必须相同。GOTO 指令执行时，GOTO 和 GLBL 之间的指令不执行。





操作数类型	D4-454 范围
A	aaa
常数 K	0-FFFF

下面的例子中，当 M7 为 ON 时，GOTO 和 GLBL（有相同的标号）之间所有的指令将被跳过，CPU 不执行被跳过的指令。

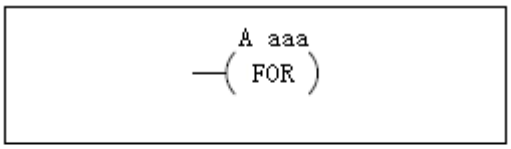


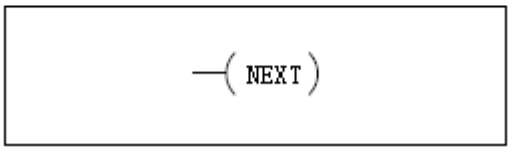
语句视图

```
LD M7
GOTO K5
LD I1
OUT M2
…
GLBL K5
LD I5
OUT Q2
```

15.3 循环执行开始指令（FOR）

15.4 循环体最后指令（NEXT）



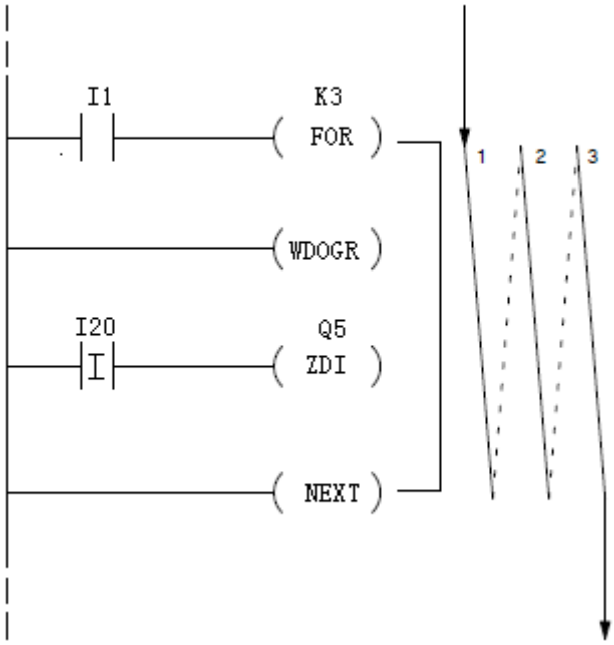


FOR 和 NEXT 指令用于按指定次数执行两个指令之间的程序。FOR 指令的条件成立时，FOR 和 NEXT 两个指令之间的程序按指定次数执行；如果 FOR 指令的条件不成立，则两个指令之间的程序不执行。

FOR/NEXT 指令不能被嵌套。程序中最多可写入 64 个 FOR/NEXT 回路，如果超出，将产生一个错误 E413。FOR/NEXT 回路执行时，正常的 I/O 更新和 CPU 的存储器管理任务被暂停。除了直接 I/O 指令以外，I/O 指令都不会被更新直到程序执行完成。如果程序执行时间很长，需要加入 WDOGR 指令将监视定时器复位。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-9999

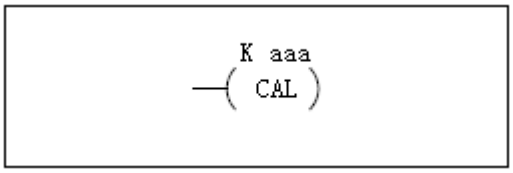
下面的例子中，当 I1 为 ON 时，FOR/NEXT 回路将被循环执行三次。如果 I1 为 OFF，则回路不执行。根据应用需要使用直接指令和 WDOGR 指令。



语句视图

```
LD I1
FOR K3
WDOGR
LDDI I20
ZDI Q5
NEXT
```

15.5 子程序调用指令（CAL）

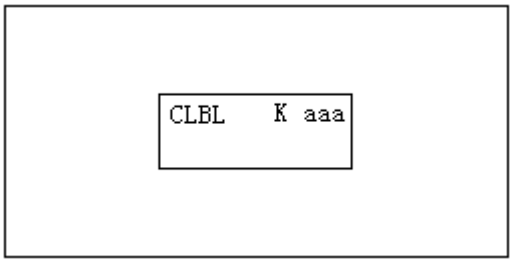


CAL 指令允许在主程序之后写入一段程序，仅在需要时调用。程序中最多可使用 256 个 CAL 指令。

子程序执行完成后，返回到 CAL 指令后的主程序中继续执行。子程序可以有 8 级嵌套，如果超过 8 级，将会产生一个错误 E412。子程序通常用于一个程序块不需要每次扫描都执行的场合。

操作数类型		D4-454 范围
	A	aaa
R 寄存器	R	所有（附录 1）
常数	K	1-FFFF

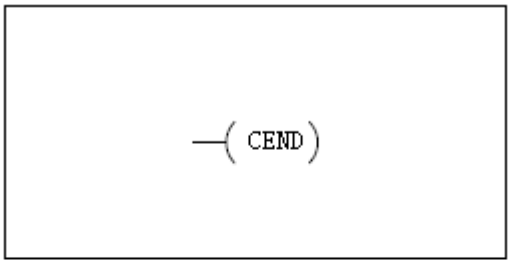
15.6 子程序开始标记（CLBL）



CLBL 指令写在主程序 END 指令之后。当 CAL 指令执行时，CPU 将执行同 CAL 有相同标号的子程序。

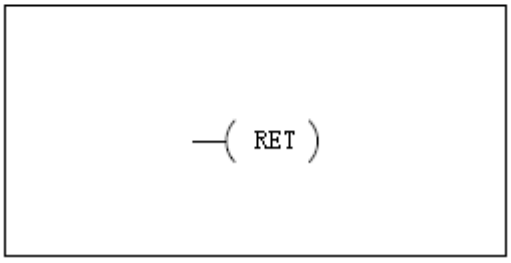
操作数类型		D4-454 范围
	A	aaa
常数	K	1-FFFF

15.7 无条件复归指令（CEND）



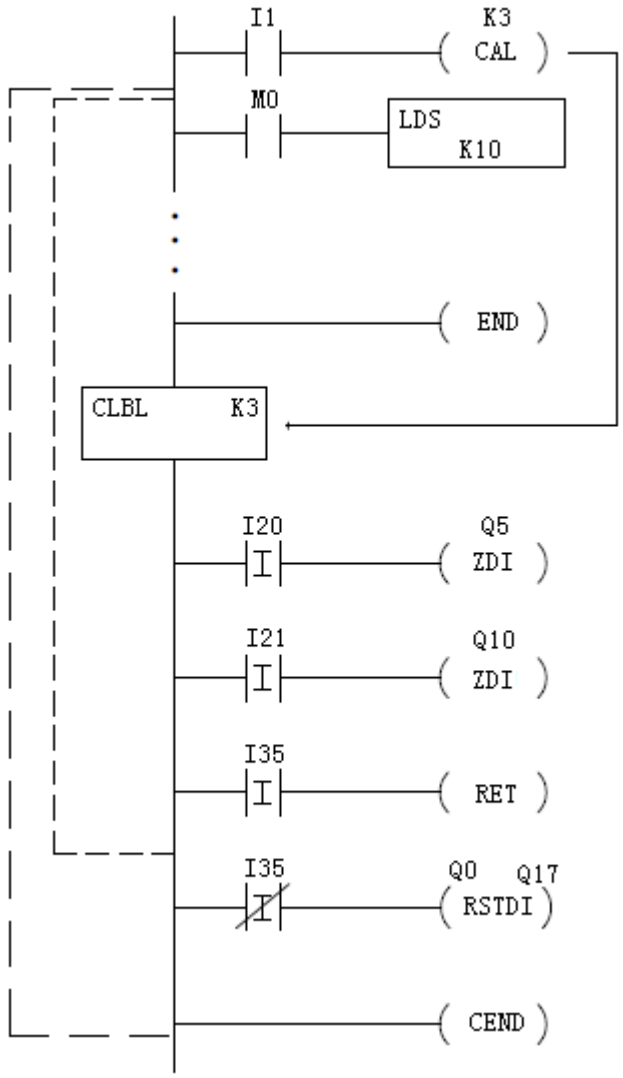
子程序中的 CEND 指令执行时，CPU 将返回到主程序中子程序被调用的地方。CEND 指令用作子程序的结束指令，必须是子程序的最后一个指令，指令前不可添加条件触点。

15.8 条件复归指令（RET）



RET 指令是一个可选用指令，同一个条件触点配合使用来完成从子程序有条件返回。使用 RET 指令时，仍然需要结束子程序的 CEND 指令。

下面的例子中，当 I1 为 ON 时，子程序 K3 将被调用。CPU 将跳去执行标记号为 K3 的子程序。如果 I35 为 ON，CPU 将返回主程序中。如果 I35 为 OFF，则 Q0-Q17 将被复位，然后 CPU 返回到主程序中。

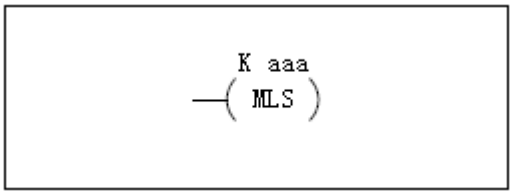


语句视图

```
LD I1
CAL K3
LD M0
LDS K10
…
END
CLBL K3
LDDI I20
ZDI Q5
LDDI I21
ZDI Q10
LDDI I35
RET
LDNDI I35
RSTDI Q0 Q17
CEND
```

15.9 新母线开始指令（MLS）

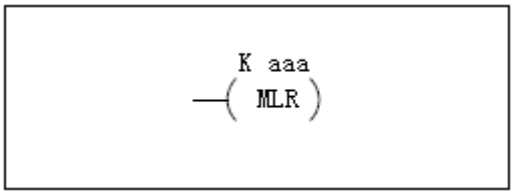
MLS 指令是定义 LD 指令(或 LD 开始的指令群）条件新母线的指令。主母线编号一直是 0，MLR 指令依次指定 1-7 的母线编号。



操作数类型		D4-454 范围
	A	aaa
常数	K	1-7

15.10 母线复归指令（MLR）

MLR 指令根据母线号的指定，返回到前面的母线上。MLR 指令的标记号比相应的 MLS 指令的标记号小 1。



操作数类型		D4-454 范围
	A	aaa
常数	K	0-6

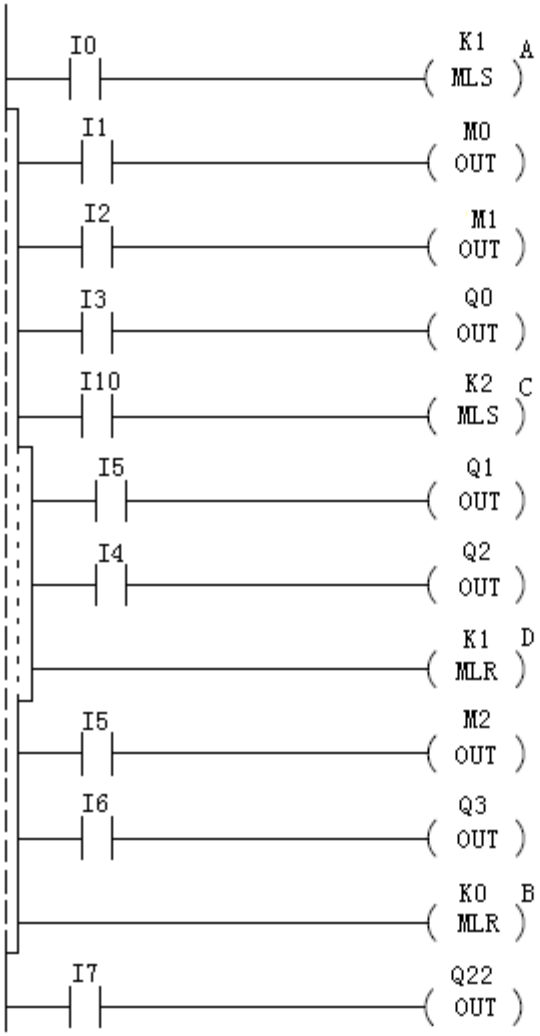
MLS 和 MLR 指令允许用户快速控制程序段的通断，使程序控制更加灵活，下面的例子演示了如何使用 MLS 和 MLR 指令。

下面的例子中，当 I0 为 ON 时，第 1 个 MLS K1（A）和 MLR K0（B）之间的逻辑程序才能接通。（注意，如果 I0 为 OFF，程序仍被扫描，但是没有接通。）当 I10 和 I0 为 ON 时，MLS K2（C）和 MLR K1（D）之间的逻辑程序将接通。最后一行程序不受任何 MLS 线圈控制。

注意，MLS/MLR 指令控制程序段是否接通，但是不控制指令是否执行。指令始终是执行的，但由于没有接通，程序不能将输出线圈置为 ON，在例子中，考虑下面的情况：

1. I0 为 OFF，意味着第二行的程序没有接通。
2. 使用编程设备将 M0 置为 ON。

因为第二行程序（LD I1，OUT M0）没有接通，如果希望 M0 被接通，使用编程设备将 M0 置为 ON。但是，MLS 指令并不意味着控制区域的指令没有执行，实际上，它们被执行了，但是没有接通。所以这种情况下，当这行程序被执行时，M0 将被置为 OFF。这是因为 CPU 检测到此行程序没有接通。当 CPU 执行这行程序时，就将 M0 置为 OFF。

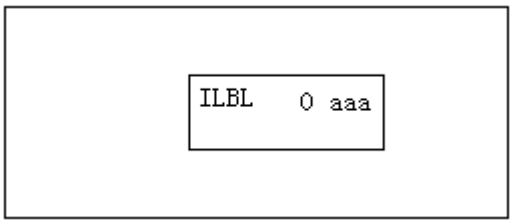


语句表视图

```
LD I0
MLS K1
LD I1
OUT M0
LD I2
OUT M1
LD I3
OUT Q0
LD I10
MLS K2
LD I5
OUT Q1
LD I4
OUT Q2
MLR K1
LD I5
OUT M2
LD I6
OUT Q3
MLR K0
LD I7
OUT Q22
```

第 16 章 中断指令

16.1 中断子程序指令（ILBL）



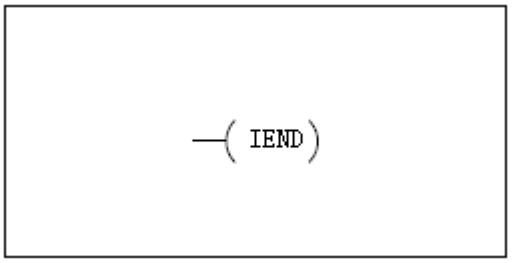
ILBL 指令允许在主程序外写入一段中断程序，需要的时候执行。中断可在程序中调用。

通常情况下，中断程序用于需要对一个输入快速响应的场合或者需要执行的程序段要比普通 CPU 扫描快的场合。中断标记以及相关中断程序必须放在 END 指令之后。当中断子程序通过软件中断调用时，CPU 将结束目前处理的指令，转而执行相应的中断子程序。一旦中断程序执行完成，程序执行将从中断发生前的地方继续执行。

软件中断为 ILBL16 与 ILBL17，可在 R736 与 R737 中设置中断时间。有效范围是 3-999ms。数据必须是 BCD 格式。如果数据超出有效范围，中断将不执行。

操作数类型	D4-454 范围
A	aaa
常数 K	16-17

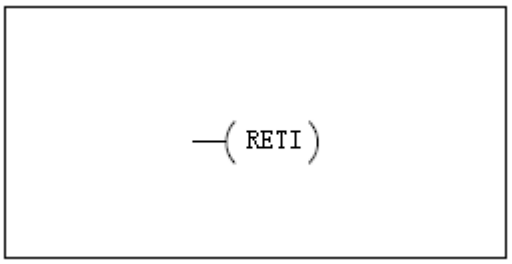
16.2 中断子程序无条件复归指令（IEND）



IEND 指令执行时，CPU 结束中断程序，返回到主程序原先中断的地方继续执行。IEND 指令是中断子程序的最后一个指令，前面不可添加条件触点。

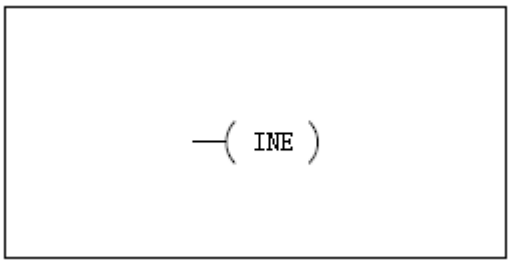
16.3 中断子程序条件复归指令（RETI）

RETI 指令是一个可选用指令，同一个条件触点配合使用，完成从中断子程序有条件返回。使用 RETI 指令时，仍然需要结束中断子程序的 IEND 指令。



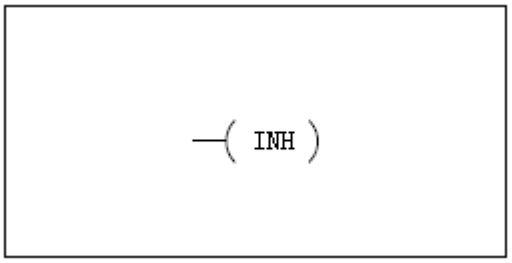
16.4 中断许可指令（INE）

INE 指令是写在主程序中，END 指令之前。INE 指令执行时，硬件或软件中断被许可，直到中断禁止指令 INH 执行时，中断禁止。



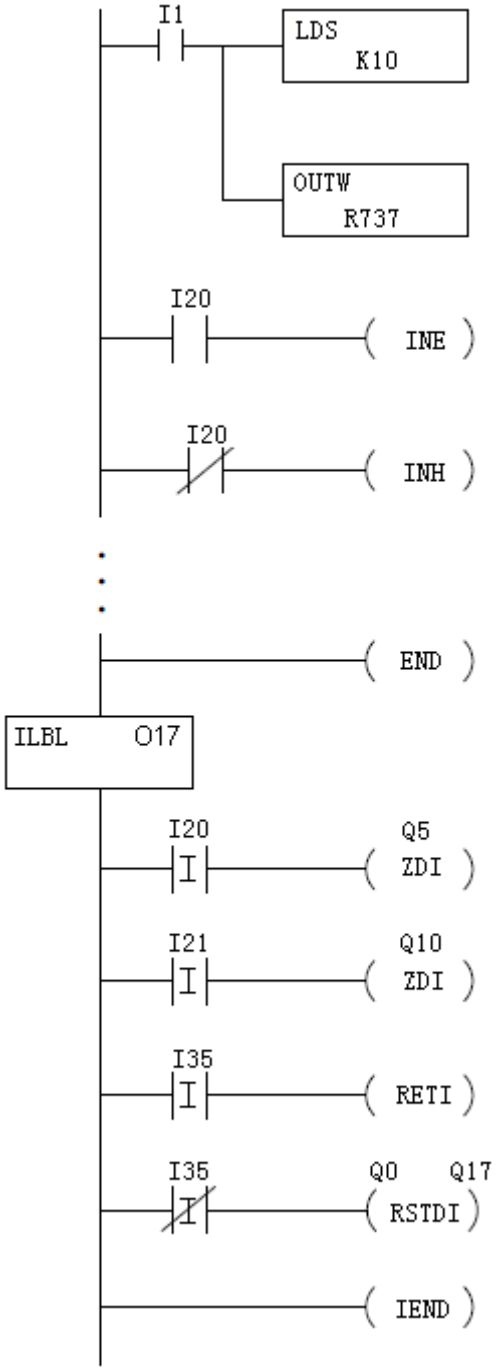
16.5 中断禁止指令（INH）

INH 指令是写在主程序中，END 指令之前。INH 指令执行时，硬件或软件中断被禁止，直到中断许可指令 INE 执行时，中断被许可。



16.6 软件中断举例

下面的例子中，当 I1 为 ON 时，常数 K10 被读入累加器，然后写入到 R737 中。此值将软件中断时间设置为 10ms。当 I20 为 ON 时，中断被允许，当 I20 为 OFF 时，中断被禁止。每隔 10ms，CPU 将跳去执行中断程序 ILBL O17。当 I35 为 ON 时，CPU 将返回到主程序中。当 I35 为 OFF 时，Q0-Q17 被复位。软件中断的中断时间范围是 3-999ms。输入 0、1 或 2，中断时间都是 3ms。



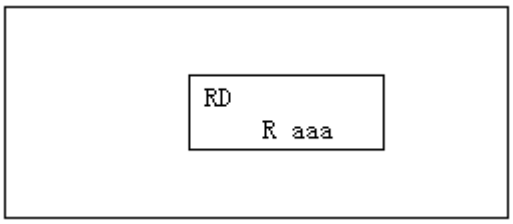
语句视图

```
LD I1
LDS K10
OUTW R737
LD I20
INE
LDN I20
INH
…
END
ILBL O17
LDDI I20
ZDI Q5
LDDI I21
ZDI Q10
LDDI I35
RETI
LDNDI I35
RSTDI Q0 Q17
IEND
```

第 17 章 智能模块指令

17.1 智能模块读出指令（RD）

RD 指令将智能模块的数据（范围是 1-128 字节）读出到 CPU 的 R 寄存器中。指令执行前，需将功能参数读入数据堆栈的第 1、第 2 级和累加器中。



下面是实现本指令功能的步骤：

第 1 步：将框架号（0-3）和插槽号（0-7）读入堆栈的第 2 级中。

第 2 步：将要读出的字节数（最多 128 个字节）读入堆栈的第 1 级中。

第 3 步：将要读出数据的起始地址读入累加器中，这个数必须是十六进制数。

第 4 步：写入 RD 指令，指定存放读出数据的起始寄存器地址。

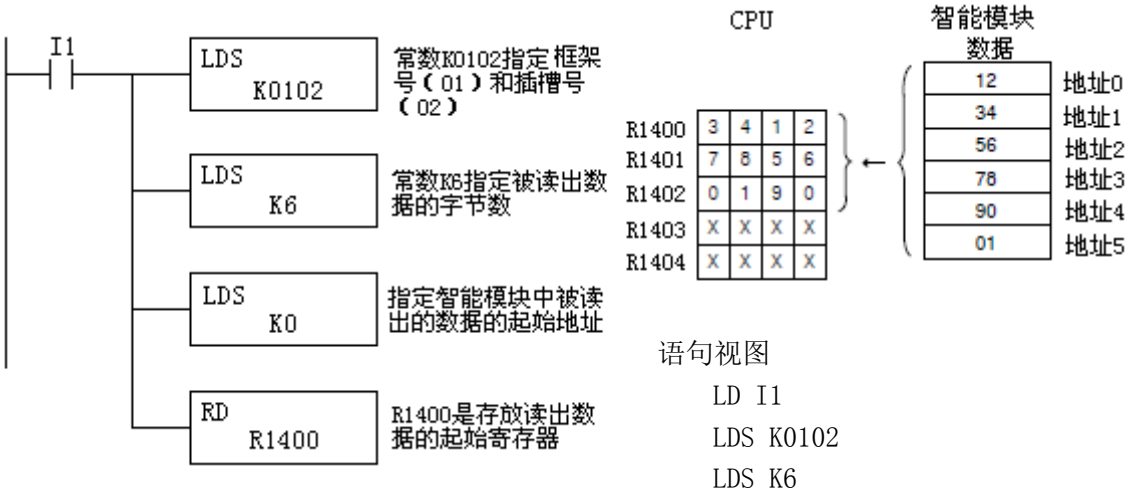
小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

受影响的标志线圈	描述
SP54	当 RX、WX、RD、WT 指令参数错误时 ON

注意：当另一条影响标志位指令执行之前，这些状态标志一直有效。

下面的例子中，当 I1 为 ON 时，RD 指令从框架 1，插槽 2 中的智能模块中读取 6 个字节的数据。智能模块中被读出的数据起始地址为 0，读出后数据被写入 R1400-R1402 中。

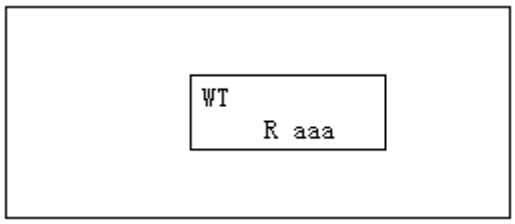


语句视图

```
LD I1
LDS K0102
LDS K6
LDS K0
RD R1400
```

17.2 智能模块写入指令（WT）

WT 指令将 CPU R 寄存器中的数据块（范围是 1-128 个字节）写入到智能模块中。指令执行前，需将功能参数分别读入数据堆栈的第 1、第 2 级和累加器中。



下面是实现本指令功能的步骤：

第 1 步：将框架号（0-3）和插槽号（0-7）读入堆栈的第 2 级中。

第 2 步：将要写入的字节数（最多 128 个字节）读入堆栈的第 1 级中。

第 3 步：将智能模块要接收数据的起始地址读入累加器中，这个数必须是十六进制数。

第 4 步：写入 WT 指令，指定 CPU 中存放要写入数据的起始寄存器地址。

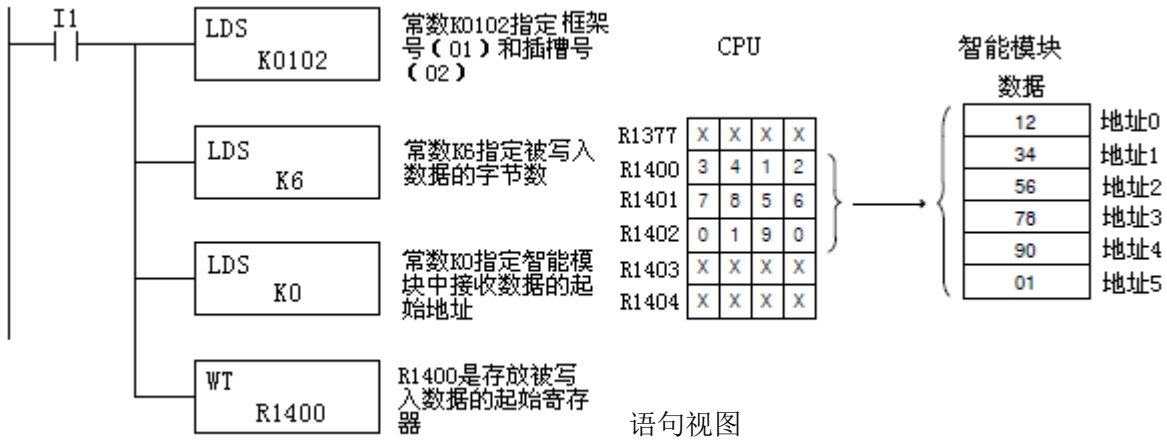
小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）

受影响的标志线圈	描述
SP54	当 RX、WX、RD、WT 指令参数错误时 ON

注意：当另一条影响标志位指令执行之前，这些状态标志一直有效。

下面的例子中，当 I1 为 ON 时，WT 指令将 CPU 中 R1400-R1402 中的 6 个字节的数据写入到框架 1，插槽 2 中的智能模块中。智能模块中接收数据的起始地址为 0。



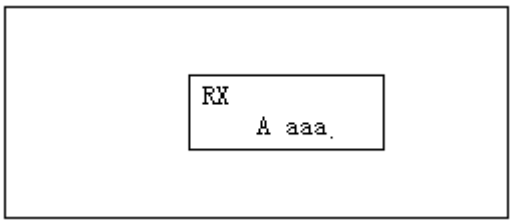
语句视图

```
LD I1
LDS K0102
LDS K6
LDS K0
WT R1400
```

第 18 章 通讯指令

18.1 读出指令（RX）

RX 指令用于网络中主局 CPU 读取子局 CPU 中的数据块。指令执行前，需要将功能参数分别读入数据堆栈的第 1、第 2 级和累加器中。



下面是实现该指令功能的步骤：

第 1 步：将子局地址(0-90 BCD 数)放入低字节，PLC 通讯口号或主局 DCM、ECOM 模块槽号放入高字节，读入堆栈的第 2 级中。

第 2 步：将要被读出的字节数（1-128 BCD）读入堆栈的第 1 级中。

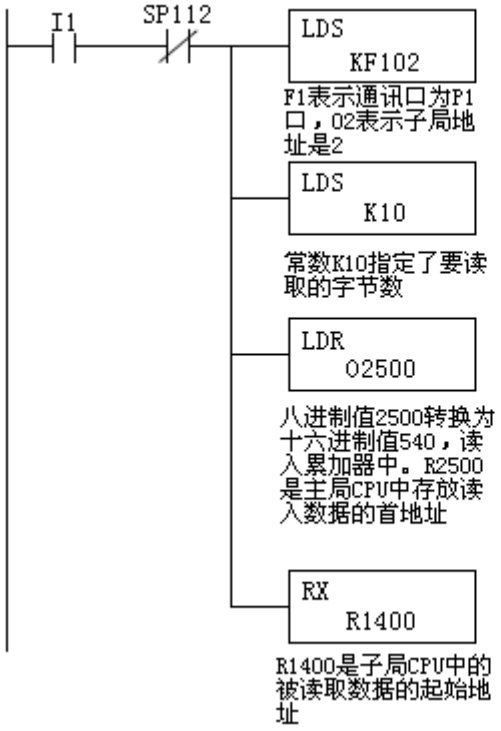
第 3 步：将主局中用来存放读出数据的起始地址写入累加器中，这个地址必须是十六进制数。

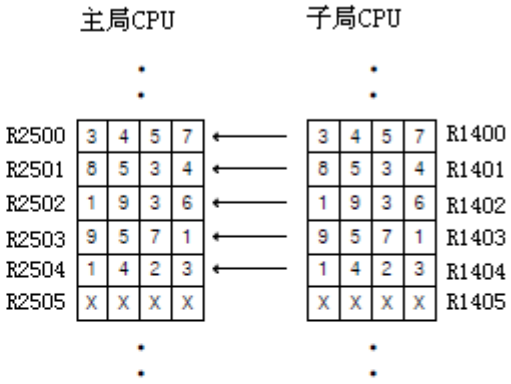
第 4 步：写入 RX 指令，指定子局 CPU 中存放被读出的数据的起始地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型		D4-454 范围
	A	aaa
寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
特殊继电器	SP	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的例子中，当 I1 为 ON 并且 SP112（0：P1 口不在通讯中；1：P1 口在通讯中）为 OFF 时，RX 指令将从子局中读取连续的 10 个字节的数据（R1400-R1404），并将数据写入到主局 CPU 的 R2500-R2504 中。主局通讯口为 P1，子局地址为 2。



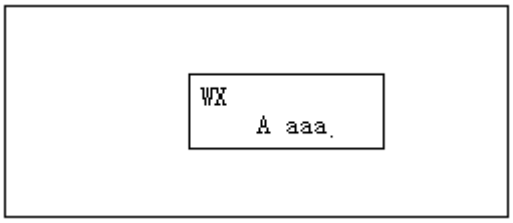


语句视图

```
LD I1
LDN SP112
LDS KF102
LDS K10
LDR 02500
RX R1400
```

18.2 写入指令（WX）

WX 指令用于网络中主局 CPU 向子局 CPU 中写入一个数据块。指令执行前，需要将功能参数分别读入数据堆栈的第 1、第 2 级和累加器中。



下面是实现该指令功能的步骤：

第 1 步：将子局地址(0-90 BCD 数)放入低字节，PLC 通讯口号或主局 DCM、ECOM 模块槽号放入高字节，读入堆栈的第 2 级中。

第 2 步：将要被写入的字节数（0-128 BCD）读入堆栈的第 1 级中。

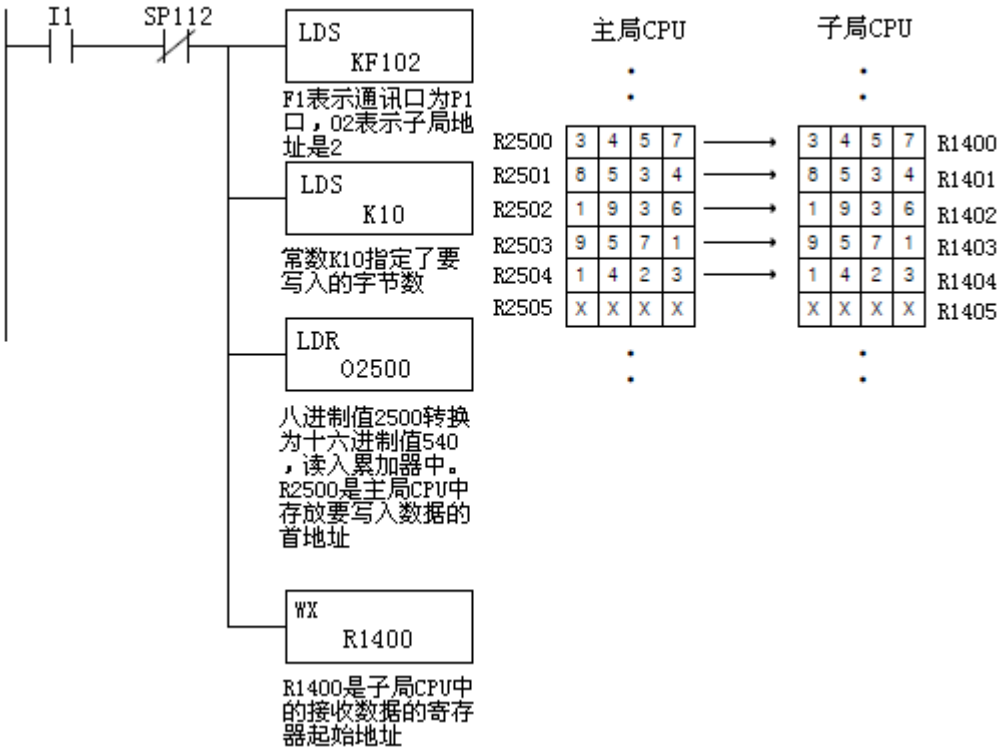
第 3 步：将主局中存放被写出数据的起始地址写入累加器中，这个地址必须是十六进制数。

第 4 步：写入 WX 指令，指定子局 CPU 中要接收数据的起始地址。

小技巧：对于需要十六进制地址数据的参数，LDR 指令可用来将八进制地址值转换为十六进制数据，并将其读入累加器。

操作数类型		D4-454 范围
	A	aaa
寄存器	R	所有（附录 1）
指针	P	所有（附录 1）
输入	I	0-1777
输出	Q	0-1777
中间继电器	M	0-3777
级	S	0-1777
定时器	T	0-377
计数器	C	0-377
特殊继电器	SP	0-777
通讯输入	GI	0-3777
通讯输出	GQ	0-3777

下面的例子中，当 I1 为 ON 并且 SP112（0：P1 口不在通讯中；1：P1 口在通讯中）为 OFF 时，WX 指令将主局中的连续 10 字节的数据（R2500-R2504）写入到子局 CPU 的中 R1400-R1404 中。主局通讯口为 P1，子局地址为 2。



语句视图

```
LD I1
LDN SP112
LDS KF102
LDS K10
LDR O2500
WX R1400
```

第 19 章 信息指令

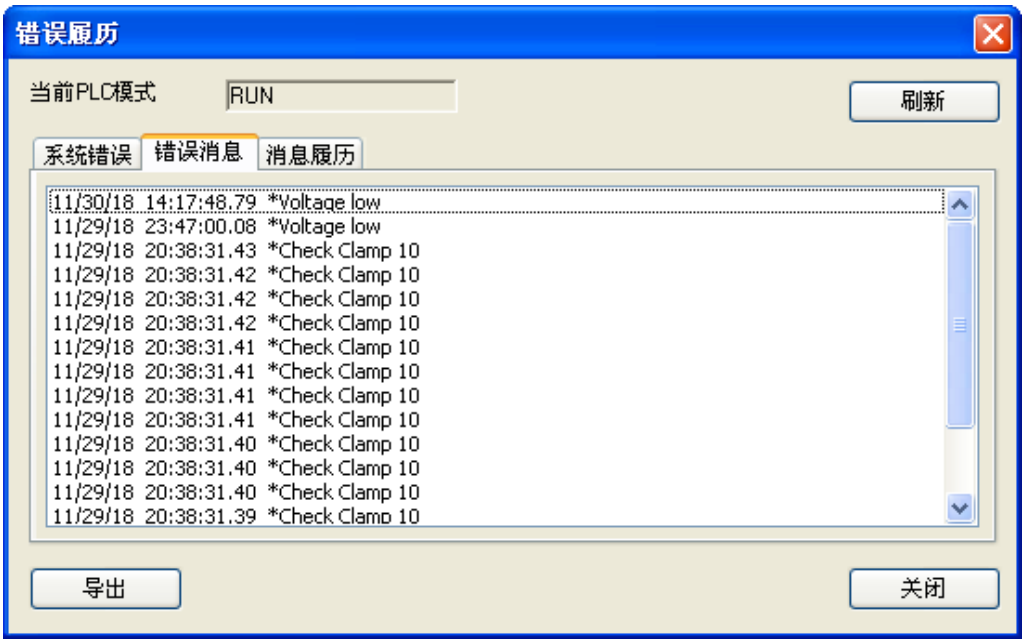
19.1 系统错误和故障信息

D4-454 CPU 提供了错误信息记录功能。有一些预定义的系统错误信息和代码，也可以使用 FALT 指令创建自己特定的信息。CPU 将记录错误及错误发生的日期、时间。有两个独立的表格来存放这些信息。

系统错误表：D4-454 可在一个错误表中显示 32 个错误。当有错误发生时，错误被装入第一个可用的位置。因此，最新的错误可能不会出现在表格的第一行。如果表格已经排满，有新的错误发生时，最老的错误被清除。

故障信息表：D4-454 也允许用户创建自己的错误代码和信息，这些被叫做故障信息。可以创建错误代码，或者创建最多 16 条信息，每条可包含 23 个字母数字字符。两种方式下，都可以在信息表中显示最多 16 条信息或代码。如果表格已经排满，有新的错误发生时，最老的错误被清除。

下图显示的 KPP 中的故障信息表。



有一些指令可配合使用，用来创建这些错误代码和信息。

FALT：外部诊断代码显示指令

DLBL：数据区标号指令

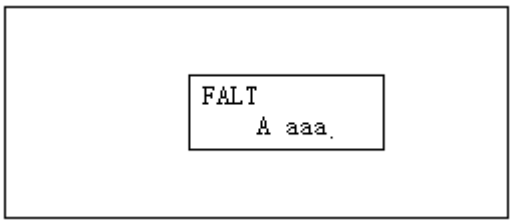
ACON：ASCII 数据登记指令

NCON：数值数据登记指令

下面有这些指令的详细介绍。在本节最后，还有两个例子演示了这些指令一起使用的用法。

19.2 外部诊断代码显示指令（FALT）

FALT 指令用于在 KPP 中显示一条信息或数字错误代码。信息最多可包含 23 个字符，这些字符可以是 R 寄存器数据、数字常数或是 ASCII 字符。



显示 R 寄存器中的数据，在指令中指定寄存器地址。

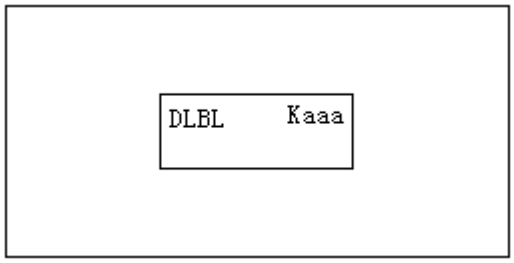
显示 ASCII 或是数字常数，必须配合使用 DLBL 指令和 ACON 指令或者 NCON 指令。这种情况下，需要在 FALT 指令中指定常数(K)的值，指定包含 ACON 和 NCON 指令的数据区标号。

操作数类型	D4-454 范围
A	aaa
R 寄存器 R	所有（附录 1）
常数 K	1-FFFF

受影响的标志线圈	描述
SP50	当 FALT 指令执行时 ON

19.3 数据区标号指令（DLBL）

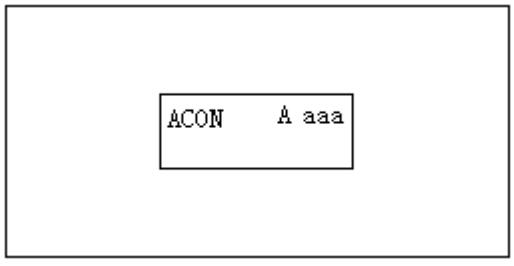
DLBL 指令标记 ASCII/数字数据区域的开始，通常同 ACON 和 NCON 指令一起使用。DLBL 指令写在 END 指令的后面。程序中最多可写入 64 个 DLBL 指令。DLBL 区域中可使用多个 NCON 和 ACON 指令。



操作数类型	D4-454 范围
A	aaa
常数 K	1-FFFF

19.4 ASCII 数据登记指令（ACON）

ACON 指令同 DLBL 指令一起使用，用来存放 ASCII 文本。该指令在手持编程器中的用法同在 KPP 编程软件中的用法不同。



手持编程器中的用法：使用手持编程器，ACON 指令中可存放两个 ASCII 字符。如果只有一个字符，在故障信息表中字符前会有一个空格。

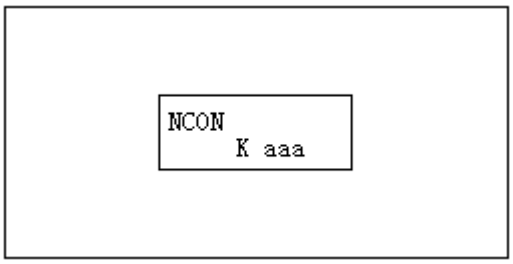
KPP 编程软件的用法：使用 KPP，ACON 指令中可存放 40 个字符，并且支持的字符更多（见附录 2）。

操作数类型		D4-454 范围
	A	aaa
ASCII	A	0-9 A-Z

你可能会觉得奇怪，为什么指令在两种编程工具中的用法不同。事实上，指令的用法没有什么不同。在 KPP 中，当程序下载到 CPU 中时，40 个字符被分进了多个 ACON 指令中，每个指令包含 2 个字符。因此，如果使用 KPP 编程软件创建程序，用手持编程器或 KPP 软件查看程序，可以看到多个 ACON 指令，每个指令包含 2 个字符。

19.5 数值数据登记指令（NCON）

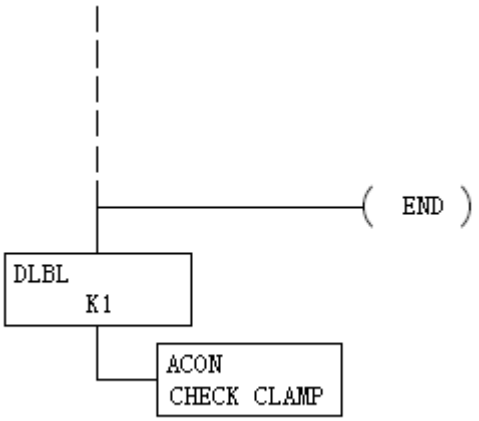
NCON 指令同 DLBL 指令一起使用，用来存放十六进制的 ASCII 等效数字。指令中可存放两个数字。



操作数类型		D4-454 范围
	A	aaa
常数	K	0-FFFF

创建信息

FALT 指令将信息写入相应的错误信息表中。了解 DLBL、ACON 和 NCON 指令如何配合使用来创建信息很重要。DLBL 指令写在 END 指令之后，ACON 和 NCON 指令写在 DLBL 区域中，见右图的例子。



19.6 历史事件记录指令（HISTRY）

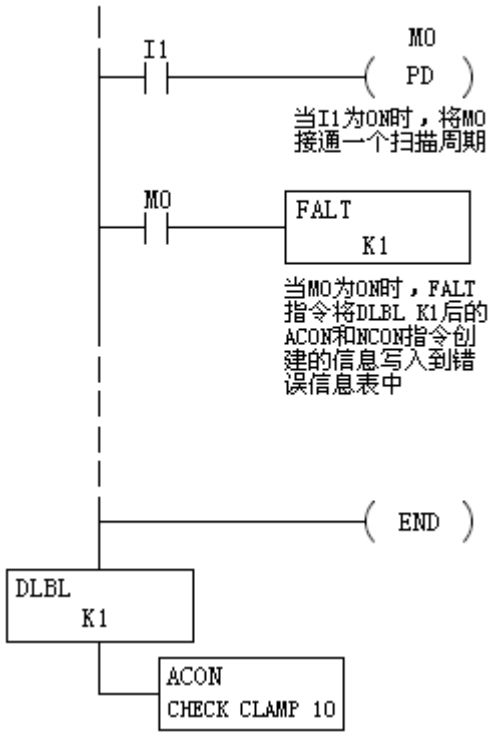
HISTRY 指令用于将事件历史信息存入 PLC 存储器中。

19.7 FALT 指令执行的重要信息

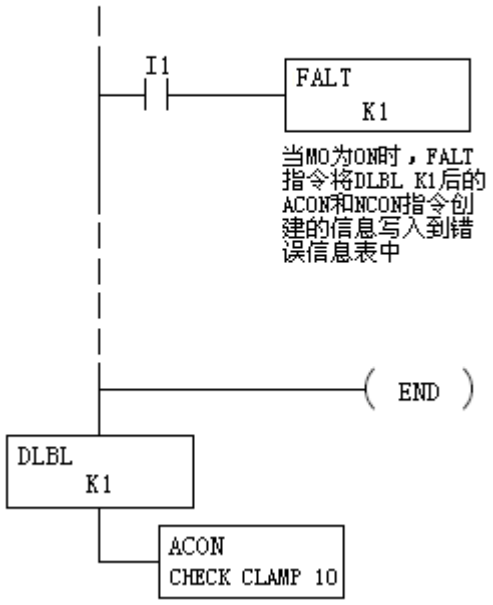
了解 FALT 指令执行时错误代码信息表是如何运行的很重要。该指令每次执行时，代码和信息被插入到错误代码信息表中。由于 CPU 扫描非常快，因此很容易一条信息填满了整张表格。在很多场合这样是不可取的，因为需要记录历史错误信息。（这就是为什么错误代码表中有 32 个位置，错误信息表中有 16 个位置，就是为了保持错误代码或信息。）

例如，如果要检查一个限位开关（I1）。当开关闭合时，需要在错误信息表中登记一条错误信息（CHECK CLAMP 10）。实际情况是，限位开关可能闭合数秒、数分钟、数小时甚至更长的时间。在此期间，CPU 可执行 FALT 指令很多次，因此，错误信息表中将被这一条错误信息填满。

要解决这个问题，只需用限位开关来触发一个用做一次输出线圈（PD）的内部继电器，然后用这个内部继电器作为 FALT 指令的执行条件即可。由于 FALT 指令由 PD 指令触发，而 PD 指令只接通一个扫描周期，因此，错误信息仅向错误信息表中写入一次，这样就能保持历史信息的完好。



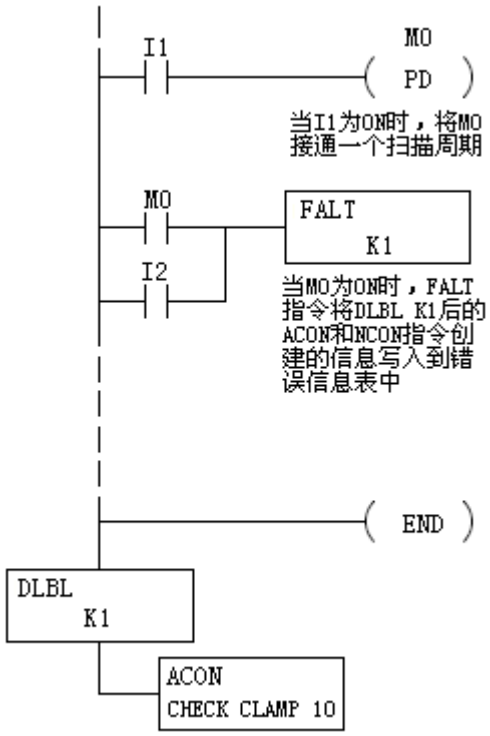
注意：这种方法用手持编程器无法实现，信息将不会显示。必须使用输入触点直接触发 FALT 指令，请看右边的例子程序。



当然，可以将 PD 指令同其它指令组合使用，以使程序同时适用于 KPP 和手持编程器。例如，可以使用两个输入触点来触发 FALT 指令。

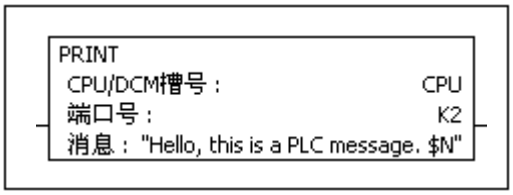
右边的例子中，当 I1 为 ON 时，信息仅被写入信息表一次，可使用 KPP 来查看信息，但是不可以自动显示在手持编程器上。

当 I2 为 ON 时，信息将填满整张信息表，并且，手持编程器上可自动显示信息。



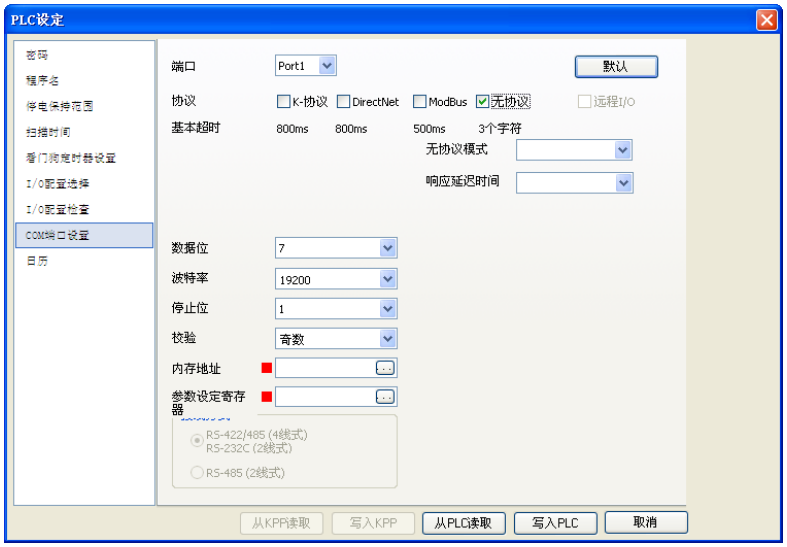
19.8 打印信息指令（PRINT）

PRINT 指令将包含文字或文字/数据变量的信息(最大 128 字符)通过指定的通讯端口（D4-454 CPU 的 1、2 或 3 端口）打印出来，通讯端口必须设置。



操作数类型	D4-454 范围
A	aaa
常数 K	1, 2, 3

D4-454 CPU 支持几种协议。通讯口 1、2 和 3 支持无协议。在 KPP 中设置通讯口，选择“PLC”菜单，然后选择“PLC 设定→COM 端口设置”，下图将会出现。



根据打印机的要求，选择各参数。完成后，点击“写入 PLC”。

DL450 的端口 1 和 2 是标准的 RS232 电平，可与多数串行输入打印机相连。端口 3 是 RS422 电平，不可与多数打印机连接。可将端口 3 用作长距离通讯，然后在另一端将信号转换为 RS232 来驱动打印机。

消息：指令此部分创建要送出通讯口的字符串。字符串是双引号中的字符。美元符号后面的两个十六进制数是一个 8 位 ASCII 码，美元符号后面的字符描述见下表。

#	字符代码	描述
1	$$	美元符号（$）
2	$”	双引号（”）
3	$L 或$l	换行（LF）
4	$N 或$n	回车换行符（CRLF）
5	$P 或$p	换页
6	$R 或$r	回车（CR）
7	$T 或$t	Tab

下面的例子演示了各种语法规则以及输出到打印机的长度。

举例：

“” 长度 0，没有字符

“A” 长度 1，有一个字符 A

“ ” 长度 1，空格

“$” ” 长度 1，双引号字符

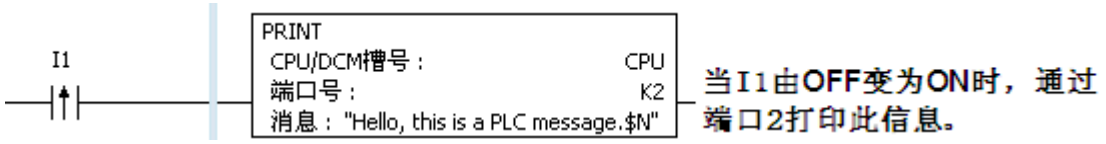
“RL” 长度 2，有一个 CR 和一个 LF

“$0D$0A” 长度 2，两个 8 位 ASCII 字符

“$$” 长度 1，一个$字符

打印一行文字的时候，需要在要打印的文字上加双引号。如果有无效文本或缺少双引号，会产生错误码 E499。在开发应用程序时，测试 PRINT 指令数据很重要。

下面的例子中，给端口 2 发送打印数据，使用一个 PD 触点，这样 PRINT 指令仅接通一个扫描周期。注意文字后面的$N 字符，它在打印机上产生一个回车/换行，这样使打印机从左边开始打印下一行。



R 寄存器打印：用来将 R 寄存器内容打印成整数格式或浮点数形式。使用寄存器地址或寄存器地址加“：”与数据类型。数据类型见下表，字母必须是大写。

注意：寄存器地址之前与之后必须加空格，使其与打印文字串区分，否则会产生错误码 E499。

#	字符代码	描述
1	无	16 位二进制（十进制数）
2	：B	4 位 BCD
3	：D	32 位二进制（十进制数）
4	：DB	8 位 BCD
5	：R	浮点数（实数）
6	：E	浮点数（指数形式的实数）

举例：

R2000 将 R2000 中的二进制数据打印成十进制数

R2000：B 打印 R2000 中的 BCD 数

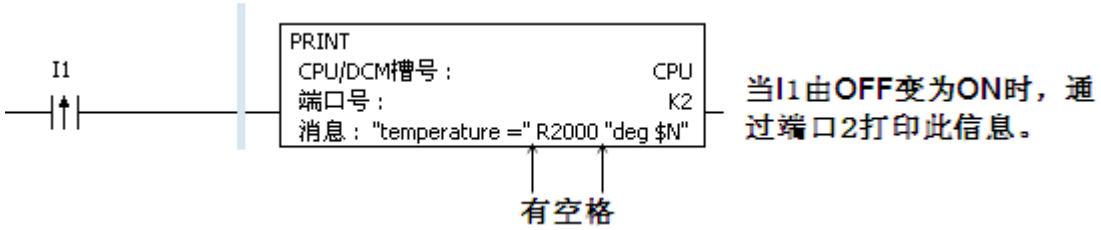
R2000：D 将 R2000 和 R2001 中的二进制数打印成十进制数

R2000：DB 打印 R2000 和 R2001 中的 BCD 数

R2000：R 将 R2000/R2001 中的浮点数打印成实数

R2000：E 将 R2000/R2001 中的浮点数打印成指数形式的实数

下面的例子中，打印一条消息，包括文本和变量。”temperature”是 R2000 中数据的标签。如果 R2000 中的数是 BCD 格式，可在 R2000 后添加字符“：B”，后面的字符是单位符号，$N 是回车换行。在文本和变量之间必须加空格。



R 寄存器文本打印：用来打印存放在寄存器中的文本，在寄存器号后添加符号“%”及字符数来表示文本。如果“%”后的数字是 0，那么打印功能将从首个 R 寄存器中读取要打印的字符数，然后从第二个 R 寄存器开始打印。

举例：

R2000 % 16 打印 R2000-R2007 中的 16 个字符

R2000 % 0 打印 R2001-Rxxxx（取决于 R2000 中的字符数）中的字符

位打印：用于打印 R 寄存器中的指定位或一个继电器位的状态，要打印的 R 寄存器位可通过在寄存器后加“.”及位数来实现，打印的状态类型见下表。

#	字符代码	描述
1	无	ON 状态时打印 1，OFF 状态时打印 0
2	：BOOL	ON 状态时打印“TRUE”,OFF 状态时打印“FALSE”
3	：ONOFF	ON 状态时打印“ON”，OFF 状态时打印“OFF”

举例：

R2000.15　　打印 R2000 中位 15 的状态，1/0 格式

M100　　打印 M100 的状态，1/0 格式

M100：BOOL　　打印 M100 的状态，TRUE/FALSE 格式

M100：ONOFF　　打印 M100 的状态，ON/OFF 格式

R2000.15：BOOL　　打印 R2000 中位 15 的状态，TRUE/FALSE 格式

最多能打印 128 个字符，各种形式可打印的字符数见下表：

类型	最大字符数
文本，1 字符	1
16 位二进制	6
32 位二进制	11
4 位 BCD	4
8 位 BCD	8
浮点数（实数）	12
浮点数（指数实数）	12
R 寄存器/文本	2
位（1/0 格式）	1
位（TRUE/FALSE 格式）	5
位（ON/OFF 格式）	3

特殊继电器 SP112-SP117 显示 DL450 CPU 的通讯口状态（忙，或是通讯错误）。

注意：在 PRINT 指令执行前，必须有相应的特殊继电器作为条件，以确保程序不会在端口正在处理上前一个 PRINT 指令或是 WX 和 RX 指令时执行 PRINT 指令。

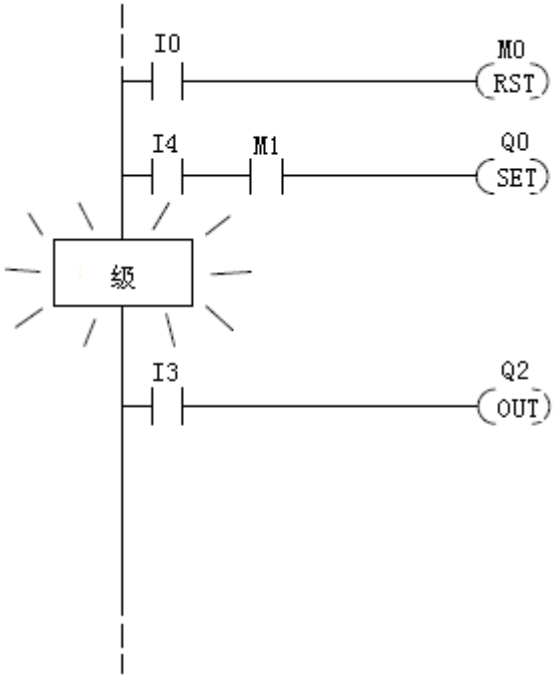
第 20 章 级式编程方法

20.1 级式编程介绍

级式编程提供了比单纯的梯形图编程相对容易的方法来组织和编写复杂的应用程序。级式语言编程不能替代或否定传统的梯形图编程方法，这也是为什么级式编程也叫 RLL^{plus}。级式语言只是允许将一个梯形图程序以级为单位进行分解和重新组织，这样编程比传统的梯形图程序更快更直观。

许多用户习惯用传统的梯形图语言进行编程，对于新的编程方法，如级式语言编程，往往持怀疑态度，甚至害怕去尝试。虽然传统的梯形图程序在解决各种逻辑关系方面功能很强大，但是它也有不足之处：

- 由于没什么结构，大型程序很难管理。
- 梯形图中，必须创建很多闭锁。
- 当进程被卡，很难找出错误的程序段。
- 由于不直观，程序后期修正很困难。



级式语言编程弥补了这些不足，因此，学习级式语言编程是很有必要的。本章详细介绍了级式语言的编程方法供用户学习，为了达到最佳学习效果，请按如下的要求做：

- 从头开始学，不要跳过任何内容。
- 通过例子学习级式语言编程概念。

20.2 画状态转移图

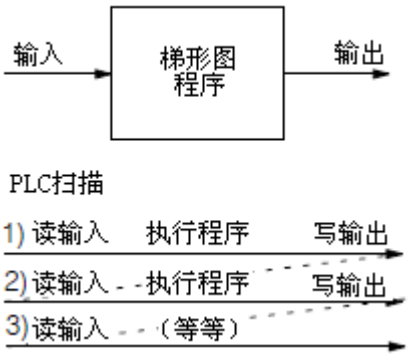
CPU 一遍一遍的重复扫描梯形图程序，扫描的基本步骤是：

1） 读输入

2） 执行梯形图程序

3） 写输出

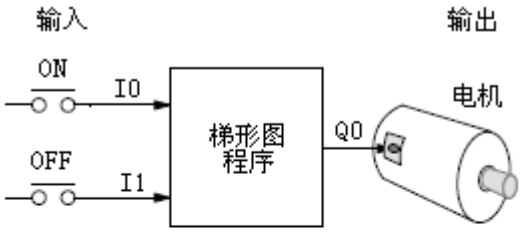
这几毫秒的时间里，输入的变化就能影响输出。



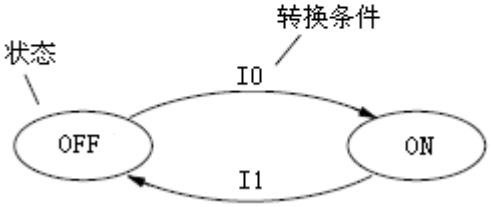
许多制造过程包含一系列的活动或条件，每个活动或条件持续数秒、数分钟甚至数小时，我们将这叫做“进程状态”，它在某个特定的时间有效或无效。梯形图程序面临的一个挑战是，一个特定的输入有可能只持续很短的时间，通常情况下，我们需要创建闭锁来保持这个输入的状态，以在需要的时间段维持一个进程状态。

可将一个梯形图程序分解成多个代表“进程状态”的“级”，然后将它们重新组织。在详细介绍级之前，首先介绍状态转移图。

有时我们需要暂时忽略 PLC 的扫描性质，将思维集中在需要确定的进程状态上。清晰的思维和简要的分析有助于我们编写出高效、无漏洞的程序。状态图是一个帮助我们画出进程的工具，你会发现，如果我们的状态图正确，我们的程序也将是正确的。



右图是一个简单进程的例子，控制一个工业电机。使用一个绿色的瞬时 SPST 按钮来开启电机，一个红色的按钮将其关闭。操作者按相应的按钮 1 秒左右。进程的两个状态是 ON 和 OFF。

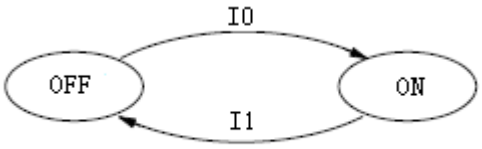


输出等式：Q0=ON

下一步是画一个状态转移图，如右图所示，显示两种状态 OFF 和 ON,用转换线条连接起来。当 I0 为 ON 时，状态转移是 OFF→ON，当 I1 为 ON 时，状态转移是 ON→OFF。

控制器的输出是 Q0，当输出为 ON 即为真，其布尔表达式 Q0=ON。

下面我们首先使用梯形图来执行状态图，然后再用级式编程来执行状态图，这样有助于用户看清两种方法的区别。

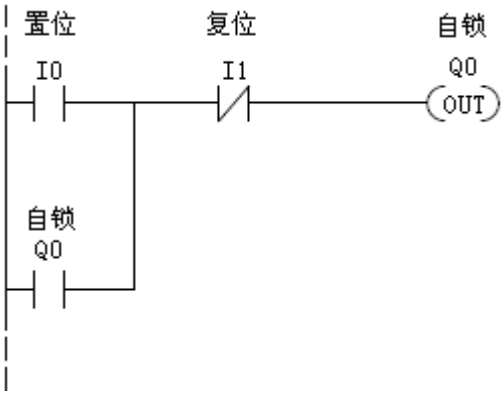


输出等式：Q0=ON

我们需要绘制右边的状态转移图，它的作用是：独立于编程语言描述问题，换句话说，通过绘制状态转化图，我们已经解决了控制问题。

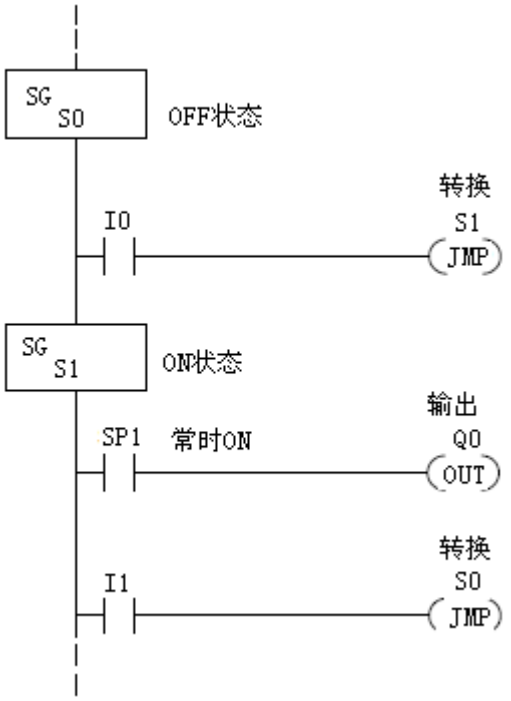
首先，我们使用传统的梯形图程序将上述的状态图编程，然后显示将状态图转换为级式语言编程有多方便。

梯形图程序：右图是梯形图程序，输出 Q0 有双重作用，既控制电机的启动与停止，又作为自锁继电器。当瞬时开关 I0 被按下时，线圈 Q0 被置为 ON，第二行的触点 Q0 也为 ON，进行了自锁，因此，当 I0 被按下后，Q0 就一直为 ON，电机始终处于运行状态。



当关闭按钮 I1 被按下时，常闭触点 I1 断开，将 Q0 的自锁复位，输出 Q0 被置为 OFF。

级式程序：级式程序如右图所示。两个级 S0 和 S1 对应两个状态 OFF 和 ON。级下的程序行属于对应的级，这就意味着 PLC 只需扫描处于激活状态的那些级。



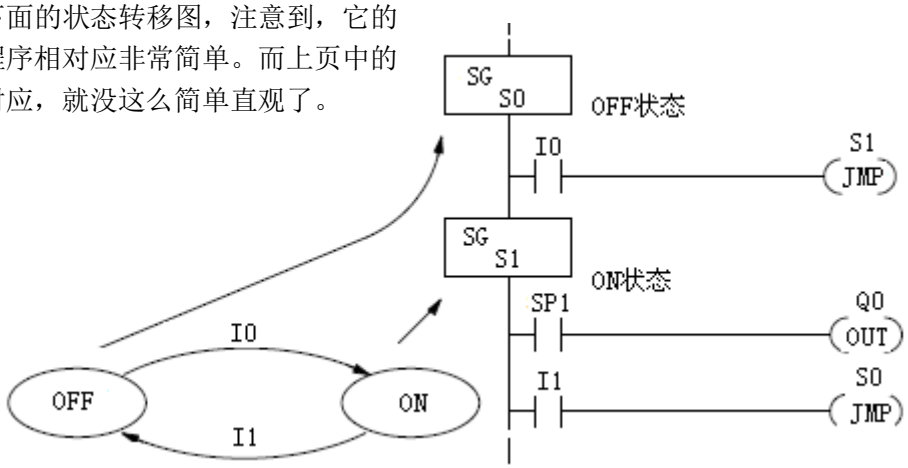
假设从 OFF 状态开始，因此级 S0 处于激活状态。当 I0 被按下时，发生级转移。执行 JMP S1 指令时，将 S0 复位，S1 置位，因此下一个扫描周期，CPU 将不再执行 S0，而是执行 S1。

S1 为 ON 时，希望电机始终处于运行状态。特殊继电器 SP1 一直闭合，Q0 输出使电机运转。

当关闭按钮 I1 被按下时，S1 回到 OFF 状态。JMP S0 指令执行，将 S1 复位，S0 置位。下一个扫描周期，CPU 将不再执行 S1，而是执行 S0。S1 不执行，则 Q0 被置为 OFF。

两种语言做比较，你可能会说：“我没有看出级式语言编程有多大优势，甚至级式语言程序比普通的梯形图程序更长”。其实，随着控制问题变得越来越复杂，级式语言程序在简单和程序大小等方面都优于传统的梯形图程序。

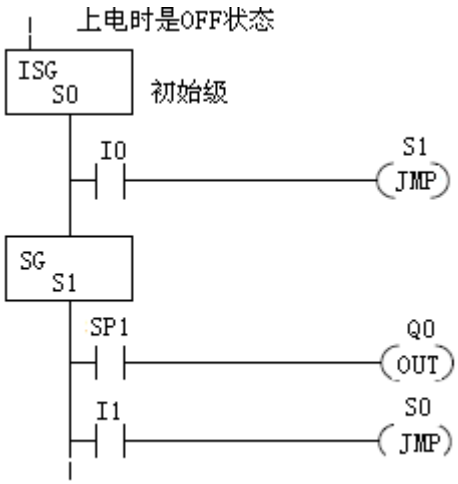
例如，对于下面的状态转移图，注意到，它的状态与级式语言程序相对应非常简单。而上页中的梯形图程序状态对应，就没这么简单直观了。



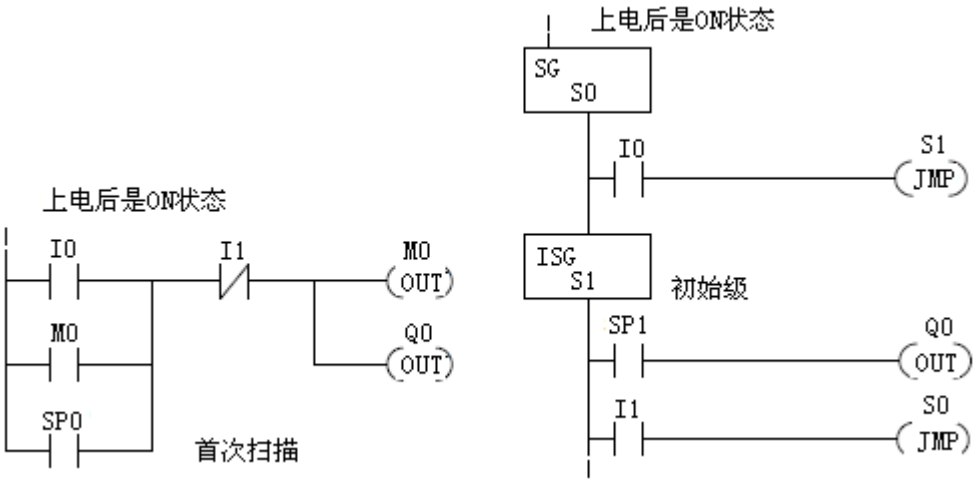
初始级

PLC 在上电以及编程→运行模式的转换后，程序中的所有级（SG）都是 OFF 状态。因此，如果没有初始级的话，级中的程序没办法扫描（因为 PLC 只扫描状态为 ON 的级中的程序）。

假设希望电机上电后的起始状态是 OFF。初始级是 PLC 上电后，定义状态为 ON 的级。右边是修改后的程序，将 SG S0 改为 ISG S0，这就确保了上电时 PLC 将扫描 I0，因为 S0 为 ON。上电以后，初始级就同其它级一样运行。

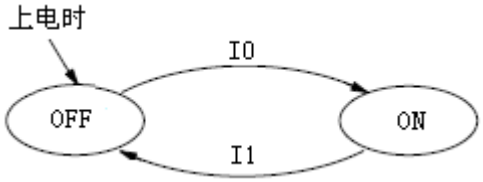


如果希望电机上电后的起始状态时 ON。下面的梯形图程序中，必须添加一个一次扫描线圈 SP0，首次扫描时将 M0 置为 ON。在下面的级式语言程序中，只需将初始级由 S0 改为 S1 即可。



注意：如果初始级被设置了停电保持，那么上电后，初始级在首次扫描时将保持其断电前的状态而不是被置为 ON。

可以在状态转移图上标记上电时的状态，见右图，这样有助于我们在编写级式语言程序时记得初始级是哪些级。程序中允许使用多个初始级。

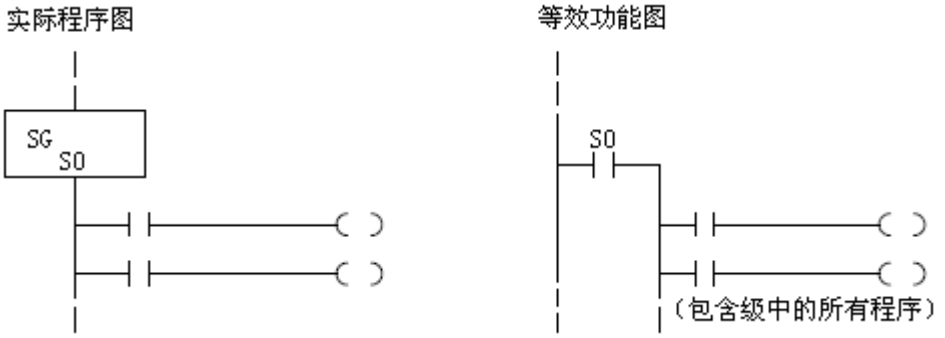


级状态位

一个级就是一个程序段，这个程序段在某个特定的时间，级的状态是 ON 或是 OFF。PLC 存储器中的所有级状态位（S0-S1777）都是独立的状态位，每个级状态值都是开关量 0 或 1。

程序执行总是从上到下，从左到右。下图演示了级状态位的作用，级登记指令下方的程序持续运行，直到到达下一个级登记指令或者直到级 S0 的程序结束。右边是等效功能图。当 S0 为 ON 时，级登记指令下面的所有 S0 中的程序执行。

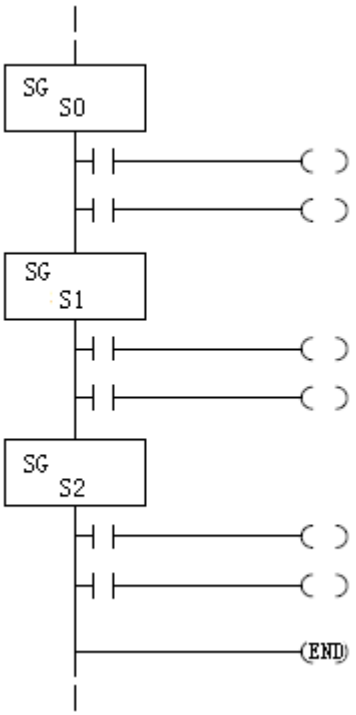
- 当 S0=0 时，S0 的程序不扫描（执行）。
- 当 S0=1 时，S0 的程序扫描（执行）。



级式指令特性

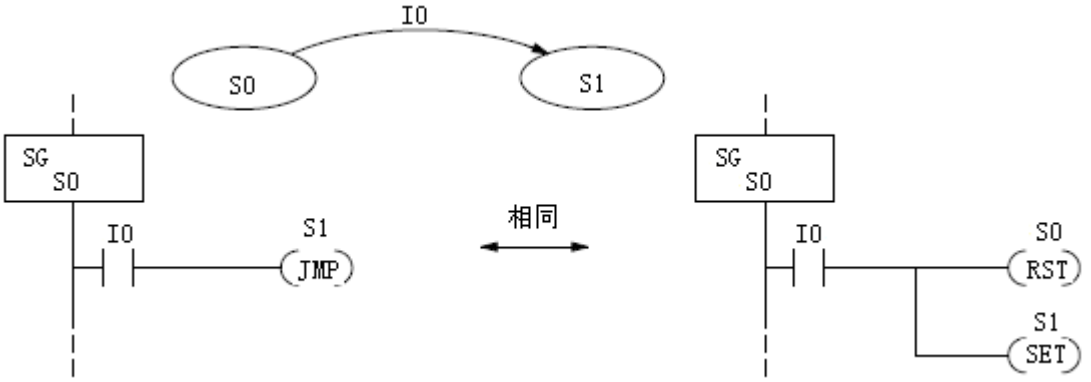
级式指令将程序划分为若干级，下面是一些级的规则：

- 执行：只有状态为 ON 的级中的程序每次扫描被执行。
- 级转移：级转移指令涉及的级，下面扫描到就执行。
- 级编号:级号是八进制,和 I/O 一样,因此“S8”是无效级号。
- 级数：D4-454 总共有 1024 个级(S0-S1777)。
- 级号不能重复：级号是唯一的，不能重复使用。
- 级号顺序：级号可任意分配，可跳过任意数字，也可以以任意顺序排列级号。
- 最后一级：最后一个级包含的程序是级登记指令到 END 线圈之间所有的程序。



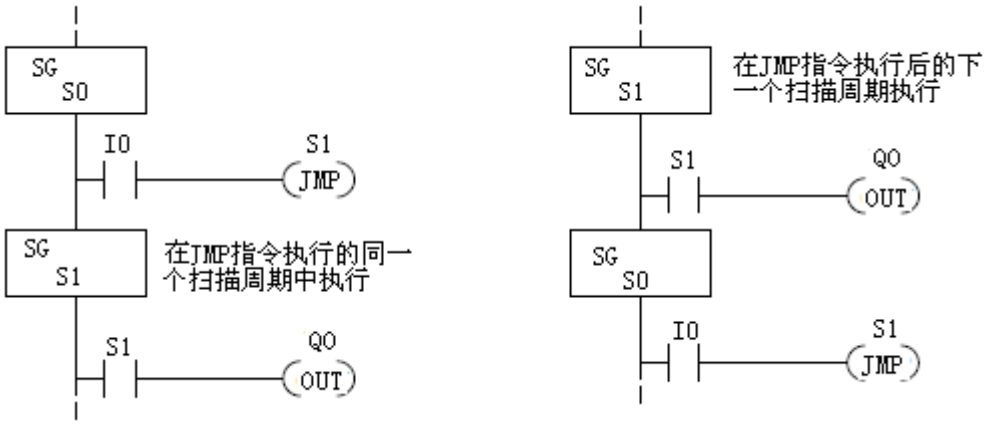
20.3 级转移指令的使用方法

级转移指令 JMP 将其所属级复位,将 JMP 指令中的指定的级置位。见下面的状态转移图。当 I0 为 ON 时，状态 S0→S1 转换发生。下面的两段程序功能是相同的。



JMP 指令很容易出现理解偏差，它不像 GOTO 和 CAL 指令那样当执行时立即跳转，它是这样工作的：

- JMP 指令执行时将其所属级复位。当前扫描周期中,级中的程序行都要执行，即使 JMP 指令后面的程序也一样。
- 复位在下一个扫描周期起作用，先前执行 JMP 指令的级被复位，PLC 执行程序时将其跳过。
- JMP 指令执行时，JMP 指令指定的级将被立即置位，因此，当程序执行到这个指定的级时，这个级就被执行。下面左边的程序例子中，S0 中的 JMP 指令执行后，S1 中程序将在同一个扫描周期被执行。下面右边的程序例子中，S0 中的 JMP 指令执行后，S1 中的程序要到下一个扫描周期才被执行，因为级 S1 在 S0 的上方。

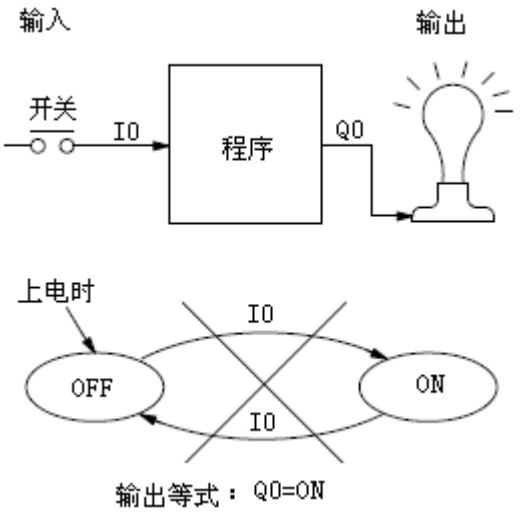


注意：上面两个例子中假定 S0 状态为 ON，S1 状态为 OFF。

20.4 级式语言编程举例：开/关灯控制

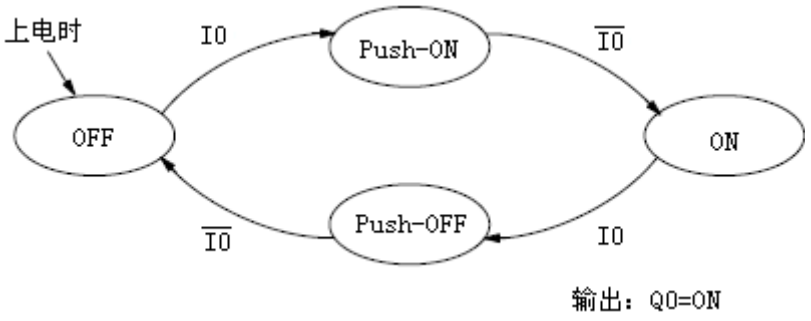
右图的例子中，使用一个普通的瞬时按钮来控制一个灯泡的开关。程序将开关输入闭锁，因此我们将开关按下然后释放可将灯打开，再次按下然后释放将灯关闭。

下面画出状态转移图。如右图所示，两个状态都使用 I0，但是，这样画是不正确的。



注意本例与上面电机例子不同，因为我们只有一个按钮。当按下这个按钮时，两个转换条件都得到满足，将以最高速度在两个状态间转换。如果使用上面的状态转移图，用级式语言编程实现，结果是在每个扫描周期灯 ON/OFF 闪烁，这显然是不行的。

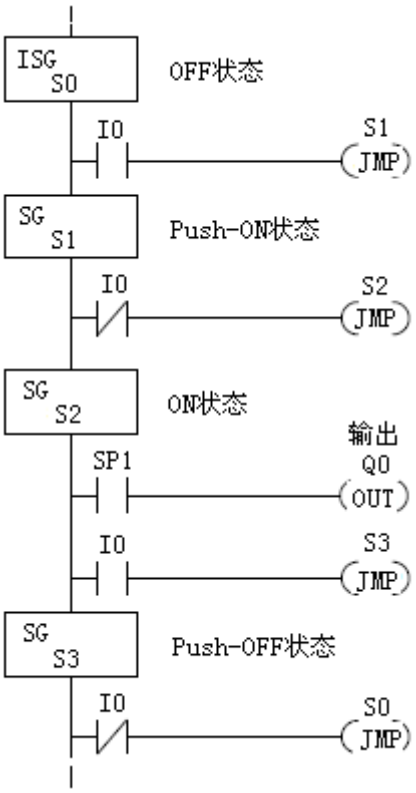
解决方法是将按下和释放按钮作为两个独立的事件。见下面新的状态转移图。上电时是 OFF 状态。当开关 I0 被按下，进入 Push-ON 状态。释放后，进入 ON 状态。注意，I0 上面有一横杠表示 I0 不导通。



ON 状态时，按下再释放回到 OFF 状态。现在，当按钮释放后，我们有了两个独立的状态（OFF 和 ON），这正是我们解决问题所需要的。

级式语言编程实现见右图。假定上电时状态是 OFF，将 S0 作为初始级。在 ON 状态，我们添加了常 ON 特殊继电器 SP1。

注意，即使程序变得很复杂，用级式编程实现状态转移图功能也是很容易的。



20.5 级式语言四步编程

你可能已经注意到，用级式语言编程来解决控制问题的步骤都是一样的。如果你熟悉了本章的所有例子，等到实际应用时，自然而然会按照这些步骤来做。下面是这些步骤：

第 1 步：写一个书面进程描述

书面描述进程的所有功能。按顺序列出每个功能。如果进程比较复杂，可将其划分为多个进程。注意，可以将各个进程的之间的联系描述出来。

第 2 步：画出框图

输入是进程处理所需要的全部信息，输出连接到进程要控制设备。

- 列出进程所需的输入输出清单。
- 将 I/O 点分配给物理设备。

第 3 步：画出状态转移图

状态转移图描述了框图的核心功能，读取输入，生成输出。

- 标示和命名进程中的状态。
- 标示状态之间转换所需的事件。
- 确保进程能够重新启动，或者说是循环的。
- 为进程选择上电时的状态。
- 写出输出等式。

第 4 步：用级式指令编程

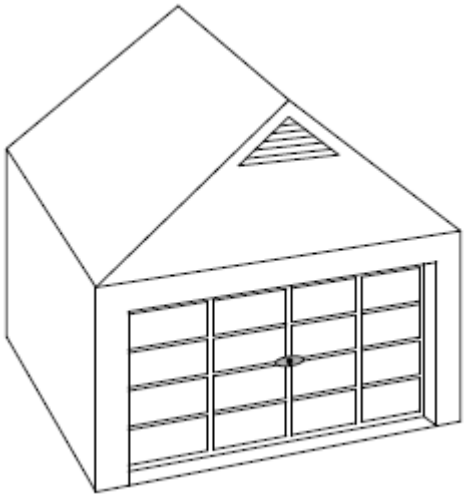
用级式指令编程实现状态转移图的功能。

- 每个状态作为一个级。注意级编号是八进制的。D4-454 CPU 最多能使用 1024 个级，即 S0-S1777。

- 每个级中编写转换程序。
- 上电时需要为 ON 的级要使用初始级登记指令（ISG）。
- 将输出或动作放在相应的级中。

注意到，步骤 1-3 是步骤 4 的准备工作。下面是级式编程的完整的例子。

20.6 级式编程举例：车库门开关控制器



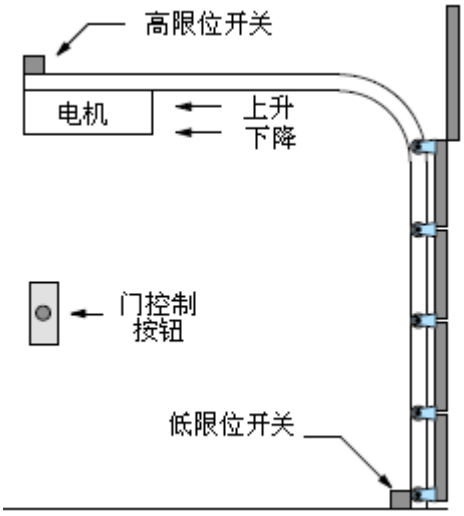
下面的例子中，我们创建一个车库门开启控制器。

功能描述

第一步必须描述门开启控制器是如何工作的。首先完成基本功能，然后在里面添加特别的功能（级式程序很容易修改）。

车库门开启控制器有一个电机，在得到命令时将门升起或下降。将一个瞬时按钮按下然后释放将门开启。门开启后，再将按钮按下然后释放可将门关闭。

为了在系统中标示输入和输出，有时需要画出系统的主要组成部分，见右图。门有一个高、低限位开关。当门开启或关闭到位后限位开关闭合。门在开启或关闭的过程中，限位开关都是断开的。

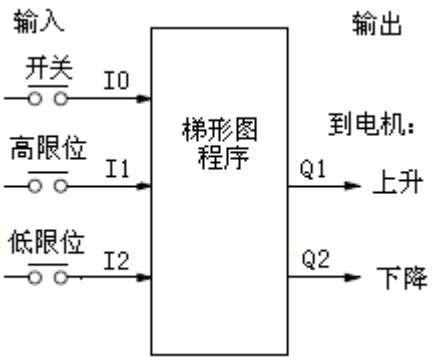


电机有两个命令输入：开启和关闭。没有命令时，电机是停止状态。门的命令是一个简单的按钮。无论是右图中的安装在墙壁上的按钮，还是遥控式的按钮，其内部都是一对或逻辑触点。

画出框图

控制器的框图见右图。输入 I0 来自门控制按钮，输入 I1 是高限位开关信号，输入 I2 是低限位开关信号。当门不是完全打开或关闭时，I1 与 I2 都是断开状态。

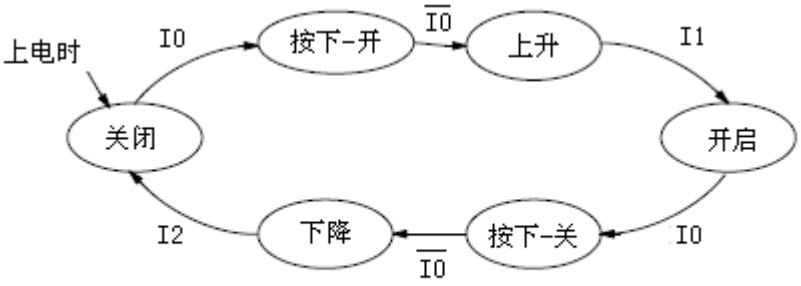
控制器有两个输出来驱动电机。Q1 是开（上升）命令，Q2 是关（下降）命令。



画出状态转移图

现在画出状态转移图，同上面的灯开关控制一样，本例也是只有一个开关。

- 当门是关闭状态时，按下然后释放按钮（I0）后，转换到上升状态，此时 Q1 被置为 ON，电机将门升起（开启）。
- 当限位开关 I1 闭合后，状态转移为开启状态，此时电机关闭。
- 再次按下并释放按钮（I0）后，转换到下降状态，此时 Q2 被置为 ON，电机将门下降（关闭）。当限位开关 I2 闭合后，状态又回到关闭状态。



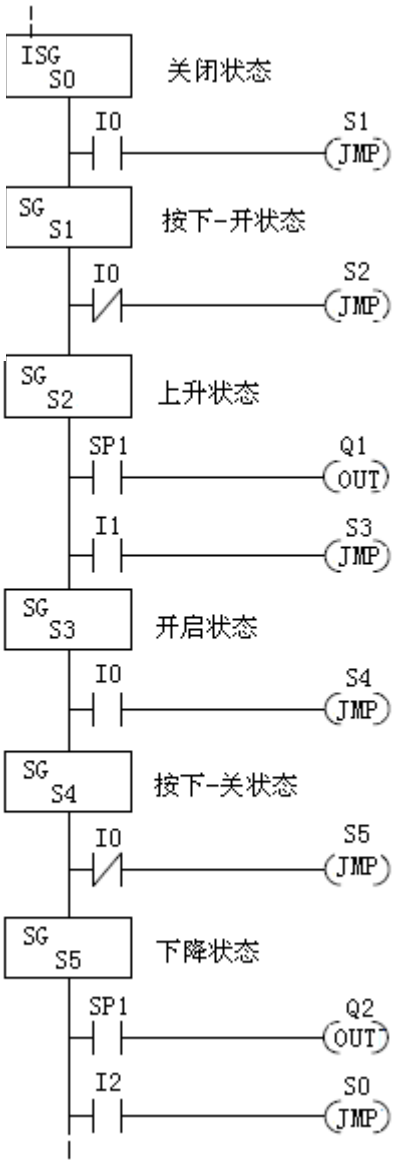
输出等式：Q1=上升　Q2=下降

级式语言编程实现

见右图。假定上电时是关闭状态，将 S0 作为初始级。级 S0 保持 ON 状态，直到按下开关按钮，状态转移（JMP）到按下-开状态，级 S1。

门开关按钮按下再释放将状态由 S1 转移到上升状态 S2。S2 中，使用常 ON 触点 SP1 来触发电机上升命令 Q1。当门到达高限位开关，限位输入 I1 闭合，将状态转移到级 S3，开启状态，在这里等待下一个门开关命令。

在级 S3 中，门开关按钮按下再释放将状态转移到下降状态 S5。S5 中，使用常 ON 触点 SP1 来触发电机下降命令 Q2。当门到达低限位开关，限位输入 I2 闭合，回到关闭状态 S0。



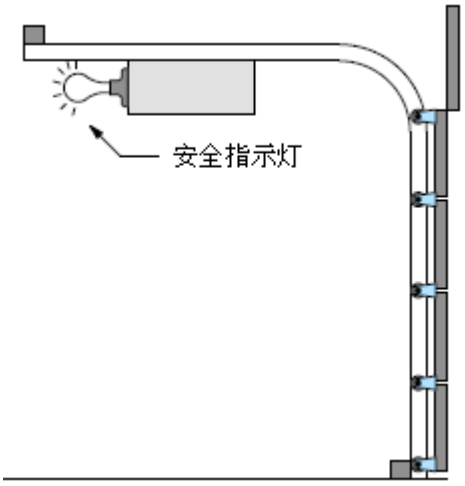
注意：上电时初始级（ISG）自动为 ON，其后，和其它级一样运行。

加入安全指示灯功能

下面给系统添加安全指示灯功能。最好的做法是首先完成主要功能的编程，正如我们前面所做的，然后再添加次要功能。

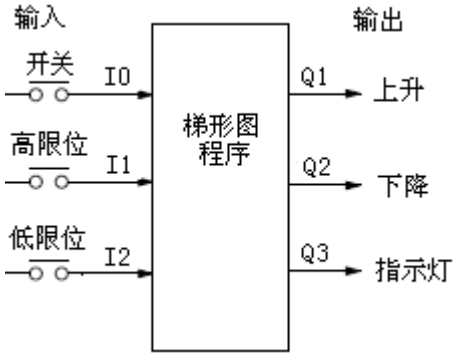
许多市售的车库门都安装有安全指示灯，见右图，安装在电机外壳上，当门有动作时，安全指示灯被点亮，然后持续约 3 分钟。

这部分牵涉到并行进程的用法，不使用 JMP 指令，而使用 SET 和 RST 指令。

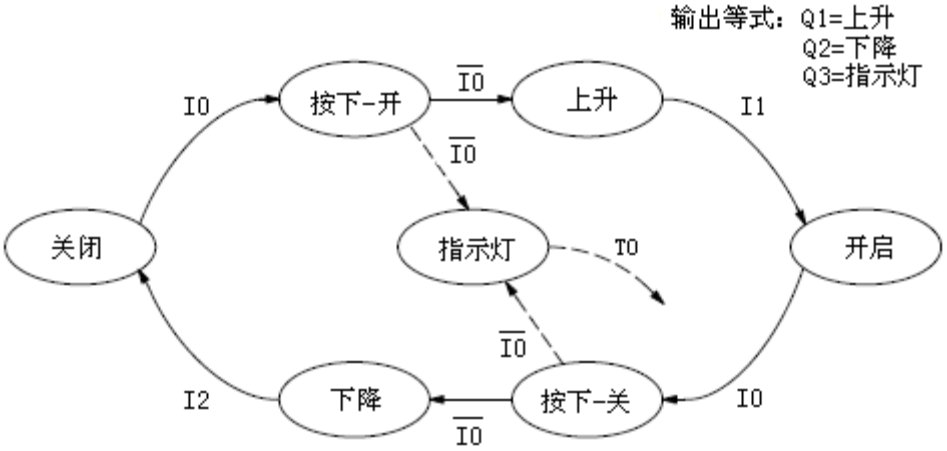


修改框图和状态转移图

为了控制安全指示灯，我们在框图中添加了一个输出 Q3，见右图。下面的状态转移图中，我们添加了辅助状态“指示灯”，只要按下再释放门开关按钮，上升或下降状态被激活的同时，指示灯状态也被激活。画到状态“指示灯”的线是虚线，因为这不是一个主要路径。



可认为指示灯状态是一个与上升和下降状态并行的进程。到指示灯的路径不是一个状态转移（JMP），而是一个状态置位（SET）。在指示灯的级中，我们将使用一个 3 分钟定时器，定时时间到了以后，定时器位 T0 被置为 ON，指示灯的级被复位。指示灯状态不转移到任何地方，只是指示灯级被置为 OFF，指示灯熄灭。



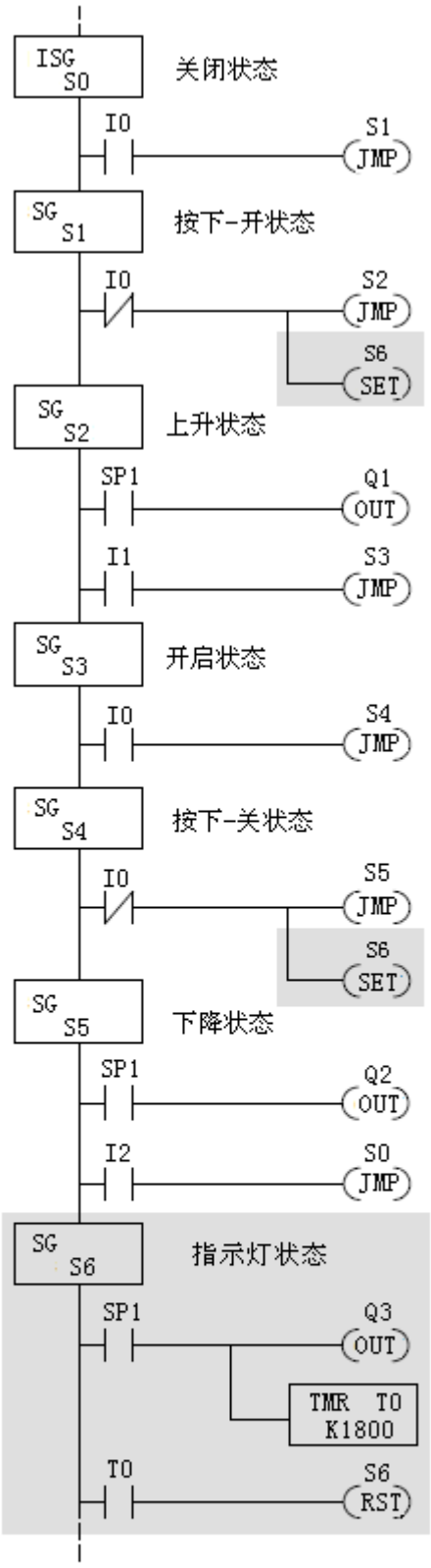
在级中使用定时器

修改后的程序见右边，阴影部分是新添加的程序。

在级 S1 中，添加 SET 指令将级 S6 置位。当触点 I0 断开，状态由 S1 转移到状态 S2 和 S6。在级 S4 中，我们做了同样的改动。因此，任何时候按下门开关按钮，指示灯都会亮。

你可能会关心在哪里添加指示灯级，并且如何给其编号，其实这些都不重要。

- 只需选择一个不使用的级号用于新添加的级即可。
- 放在程序中的位置也不是关键，本例中将其放在最后。



你可能会认为，级应该按转移顺序排列，这是很好的做法，但没必要一定这样做，有时也做不到，例如，本例中，有两处转换到级 S6 的指令。

级 S6 中，通过激活 Q3 将安全指示灯点亮。SP1 是常 ON 线圈。定时器 T0 以 0.1 秒增计时，要完成 3 分钟计时，K 的计算算式为：

$$K=\frac{3分*60秒/分}{0.1秒}$$

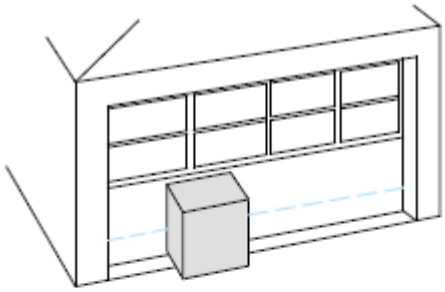
K=1800

当 S6 为 ON 时，定时器就增计时。当定时时间达到设定值，相应的定时器触点被置为 ON。因此，3 分钟后，T0=1，指令 RST 将 S6 复位。

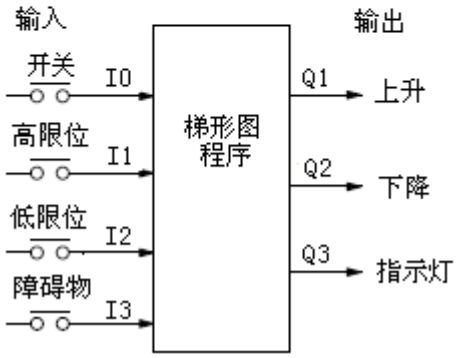
当 S6 为 ON 时，指示灯亮。主路径中的状态转移照常进行，不受 S6 的影响。因此，门可上升、下降，期间指示灯要持续点亮 3 分钟。

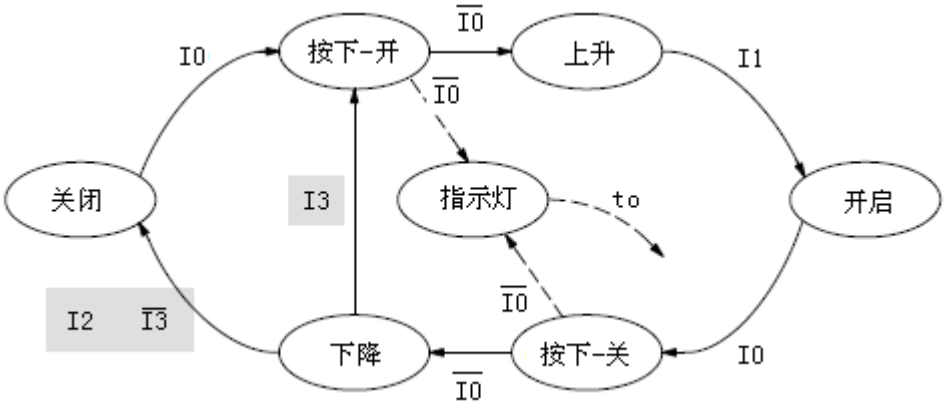
添加紧急停止功能

现在有的车库门控制器有紧急停止功能，当检测到门下有物体时，就暂停门的下降动作。一般情况下用光电管（“电子眼”）实现，下降中的门将停止下降并上升。本例中，我们将添加这样的功能。在框图中添加一个光电管的输入 I3，见右图。如果检测到物体在门的运动路径中，I3 将闭合。



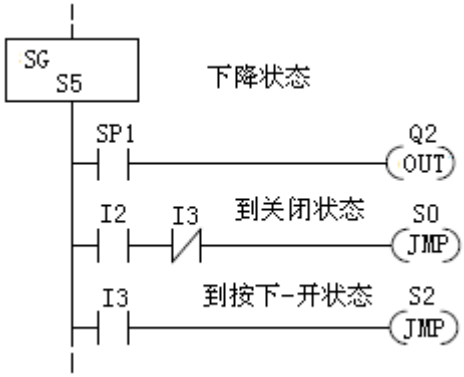
下面，在状态转移图中做一个简单的添加，下面的状态转移图中，阴影部分就是添加的部分。注意下降状态，门正在下降时，当检测到有障碍物时(I3 闭合),下降状态将转移到按下-开状态。我们没有直接将状态转移到上升状态，这样在上升输出 Q1 闭合前，给了下降输出 Q2 一个扫描周期的时间来断开。





理论上，低限位开关输入（I2）和障碍物输入（I3）同时接通是有可能的。这种情况下，状态将同时转换到按下-开状态和关闭状态，这样做是没什么意义的。

通过将到关闭状态的转移条件改为 I2 和 $\overline{I3}$ 串联，给障碍物事件较高的优先级。级 S5 中的程序修改见右图，第一行的程序不变，第二、第三行的程序按照需要进行修改。注意，程序中使用了 I3 的常开和常闭触点,这就保证了同一时间只能执行一个 JMP 指令。

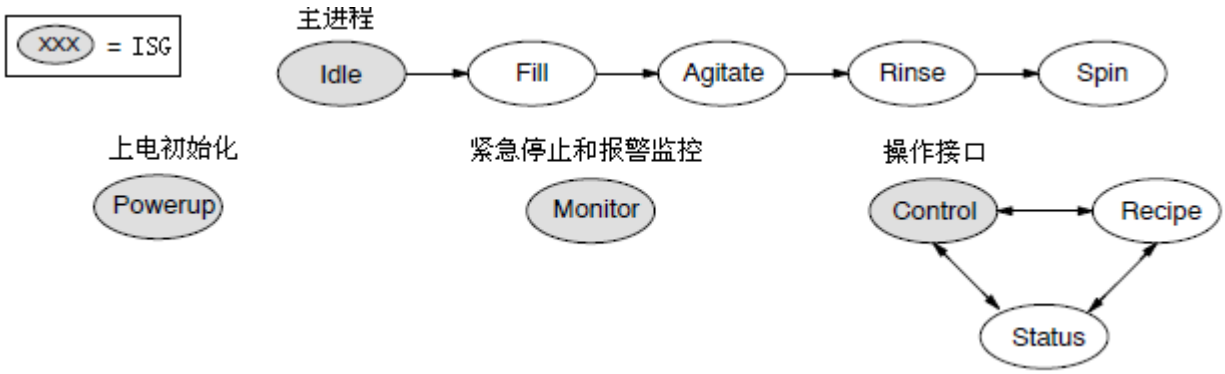


20.7 级式程序设计注意事项

级式程序组织

到目前为止，本章的例子中主进程都是使用一个独立的状态转移图。但是，一个程序中可以有多个进程。初学者可能会错误的认为每次只能一个级为 ON，在一次扫描中使级为 ON 后又使其为 OFF。对于需要每次扫描都执行的程序段，只需将它们放在有一个常 ON 的级中即可。

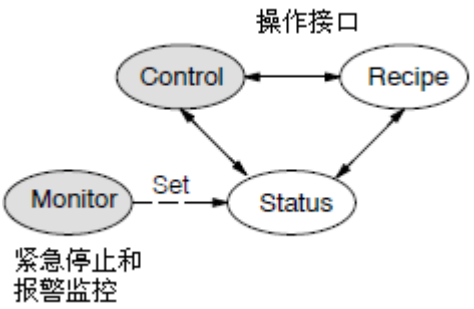
下面的图示是一个典型的应用。操作期间，主进程、上电初始化、紧急停止和报警监控、操作界面都要运行。上电时，下面的四个初始级开始运行。



上面的典型应用中，各独立级的操作顺序如下：

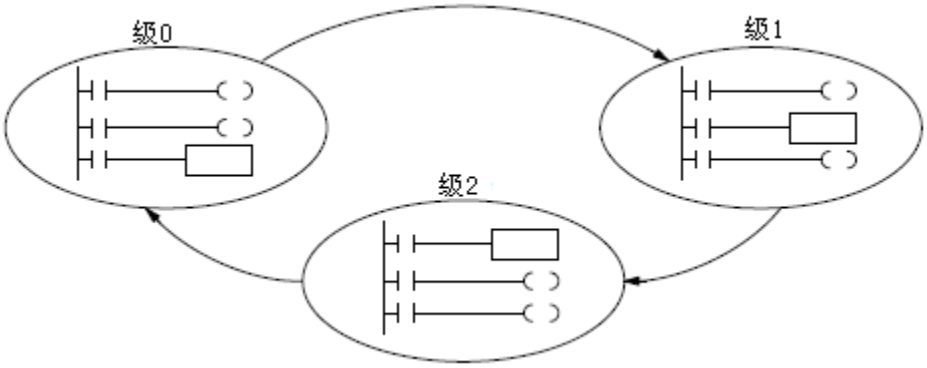
- **上电初始化**：级中的程序仅上电时扫描一次。级中的最后一行程序将级复位，因此级仅激活一个扫描周期。
- **主进程**：级中的程序控制进程或机器的核心部分。主进程的程序全部运行一次代表机器的一次循环，或进程的一次处理。
- **紧急停止和报警监控**：这个级常 ON，因为它要检测需要报警或紧急停车的错误。此级经常将主进程或其它地方的级复位，是为了在出错之后将它们初始化。
- **操作接口**：这是另一个要始终为 ON 的级，因为它要随时准备应对一个操作。它允许操作员界面来改变模式等，独立于当前主进程。

虽然进程是独立的，但是它们之间也有联系。例如，在一个错误的条件下，状态级希望操作接口级自动转换为状态模式，来显示错误信息，见右图。监控级（Monitor）可将状态级（Status）置位，将控制级（Control）和配方级（Recipe）复位。



级中指令的执行

可认为状态或者级将梯形图程序简单的划分成下图中的样子。每个级中包含相应的状态所需要的程序，将级转移的逻辑也包含在这个级中。使用初始级（ISG）可以简单的将级设置为上电时为 ON。



几乎所有的指令在级中和在标准的梯形图中的运行是一样的，可以将级看做状态为 ON 或 OFF 的小型的梯形图程序。

输出线圈：在状态为 ON 的级中，输出线圈将相应的输出置为 ON 或 OFF。但是，请注意以下几点：

- 输出正常运行，每个输出（比如“Q3”）仅在一个级中使用。
- 当级状态为 OFF 时，输出线圈自动转换为 OFF，但是 SET 和 RST 指令后的输出不会改变状态。
- 一个输出可用于多个级，但是要求同一时间只能一个级 ON。
- 如果多个级同时控制同一个输出，那么每次扫描中，程序最下方的一个状态为 ON 的级决定了输出的状态。因此，如果想使用多个级对同一个输出进行逻辑或控制，可使用 OUT 指令替代 ZOUT 指令。

一次输出线圈（PD）：如果要在一个级中使用一个上升沿线圈，要注意，这个线圈的输入必须经历一个 0→1 的转换。如果级变为 ON 时的第一个扫描周期中线圈已经接通，那么 PD 线圈将不工作，这是因为 0→1 的转换没有发生。

PD 线圈替代：如果想让一个任务只执行一次（在一次扫描周期中执行），可将这个任务放在一个级中，这个级在同一个扫描周期中要转移到下一级。

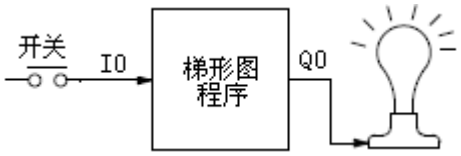
计数器：如果在级中使用了计数器，那么在计数器输入发生一个 0→1 的转换前，级必须被激活了至少一个扫描周期。否则计数器将不计数。普通计数器指令在级中使用时有个限制：它的计数值不能被其它级中的 RST 指令复位，使用级计数器指令可以解决这个问题。

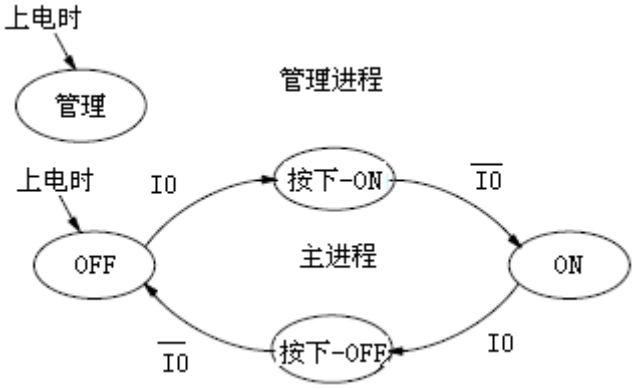
级计数器：级计数器有个优点，那就是它的计数值可被其它级中的 RSTTC 指令复位。它有一个计数输入，没有复位输入，这是它同普通计数器的唯一区别。

凸轮控制：凸轮控制有它自己的进程，同级式编程的进程不一样。如果需要在级中使用凸轮指令，要确保将其放在一直处于激活状态的初始级中。

将级用作管理级

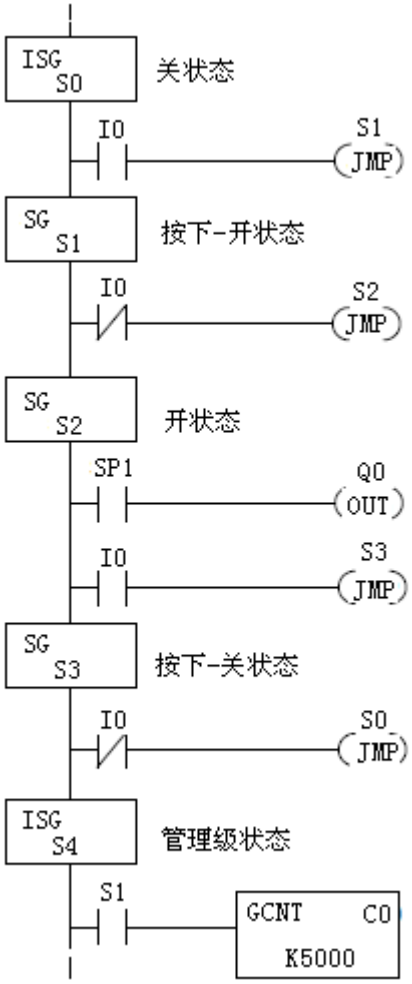
上面的灯开关控制例子中，假设我们要通过将灯 ON-OFF 次数计数来监控灯进程的“生产力”。这个应用需要添加一个简单的计数器，但是问题的关键是在哪里添加计数器。





初学者通常会尝试将计数器放在他们想要监控的级中，这样做的问题是级只激活一段时间。要想计数器计数，计数输入必须在其所在的级被激活至少一个扫描周期后发生 0→1 的转换才可以。满足这个要求可能有点难。

这种情况下，只需添加一个管理级就可以，见上面的状态转移图。管理级中的计数器将主进程的 S1 级状态位作为计数输入。在这个管理进程中，级状态位被用作一个触点来监控一个进程。



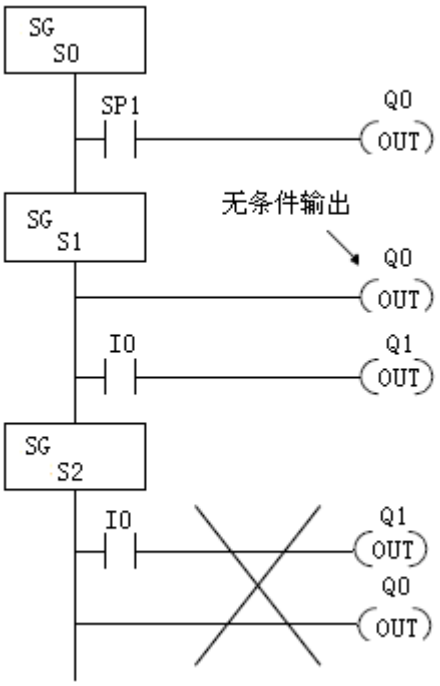
注意：管理级和关状态级都是初始级，管理级始终是激活状态。

级计数器

上例中的计数器是一个级计数器。注意到它没有复位输入（不带复位端）。当执行一个复位指令 RSTTC 时，本例中 RSTTC 指令的操作数是 C0，计数复位。级计数器的优点是它的计数值可被其它级中的 RSTTC 指令复位。普通计数器没有这个功能。

无条件的输出

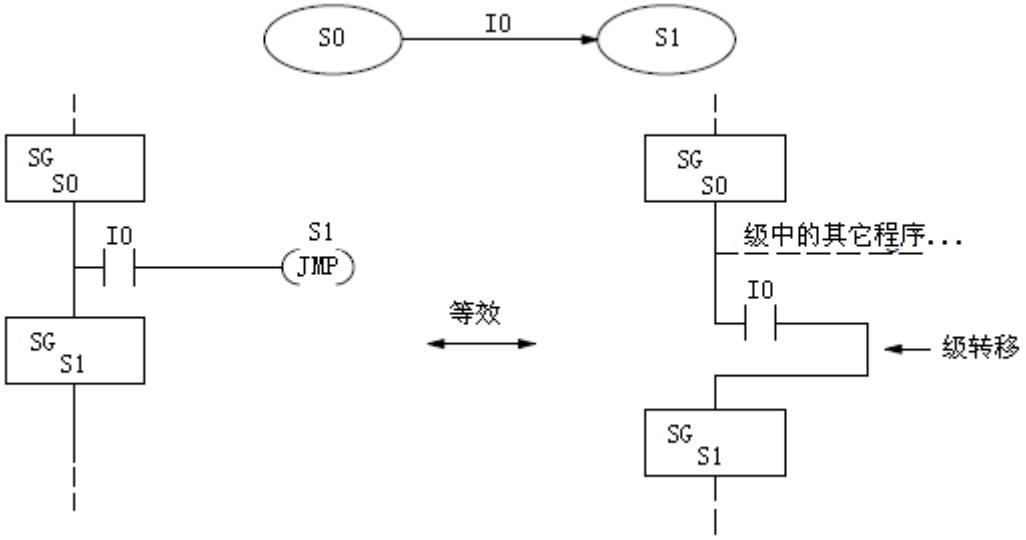
到目前为止本章的大部分例子都如右图的 S0 一样，在输出前添加一个常 ON 触点。这个触点可被省略掉，只需将这个无条件输出放在级的第一行程序中即可，见级 S1。



警告：无条件输出如果放在级的其它地方，那么级被激活时，输出不一定为 ON。在右图的级 S2 中，Q0 是一个无条件输出，由于它接通与否取决于上面一行程序，因此，Q0 状态同 Q1 状态一致（这样做是不正确的）。

级转移技术

在“级转移指令的使用方法”一节中已经介绍了级的转移方法。在 KPP 中，可以使用下面的方法来替代级转移方法。主要的要求是当前级要放在下一级（要转到的级）的上方。见下图：



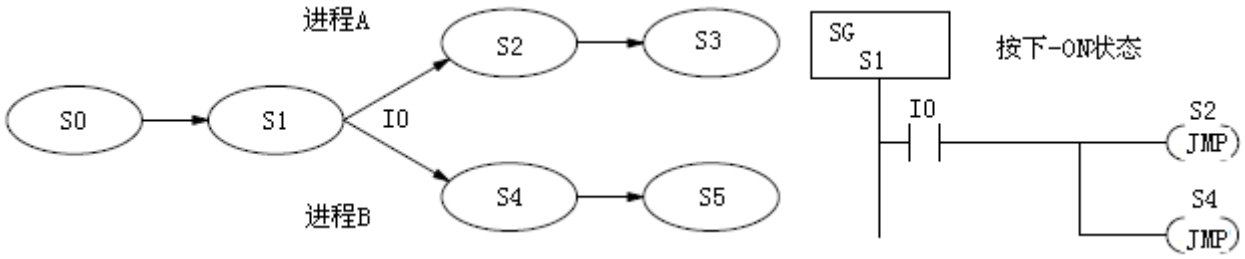
上面讲过，JMP 指令可在当前级的任何地方发生，结果都是一样的。但是，上图中的级转移只能发生在级的最后一行程序中，级中的其它程序都要在它之前。

级转移方法用起来没有 JMP 指令简单，因此，还是建议使用 JMP 指令。

20.8 并行进程的概念

进程并行

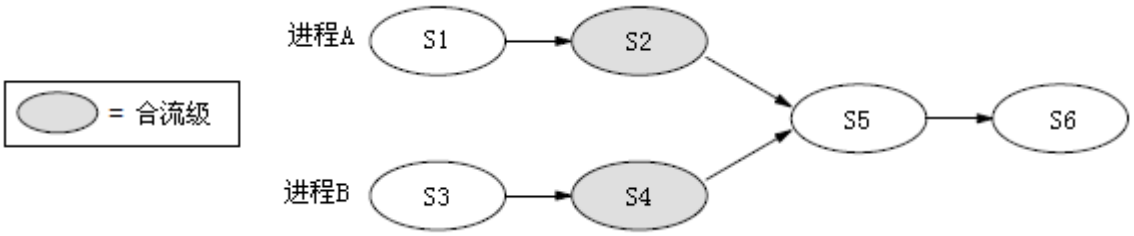
前面介绍的都是一个状态转移到另一个状态，但是有些情况下，需要同时转移到两个或两个以上的进程中，见下图。可以全部使用 JMP 指令实现转移，见下图。也可以使用一个 JMP 指令和一个 SET 指令，但是至少要使用一个 JMP 指令，因为这样可以离开其所属级。注意，级中的所有指令都会执行，甚至当发生级转移后也会执行。



注意，如果想让 S2 和 S4 在同一个扫描周期被激活，这两个级必须同时放在 S1 的上面或下面。总的来说，并行进程处理很简单。

进程合流

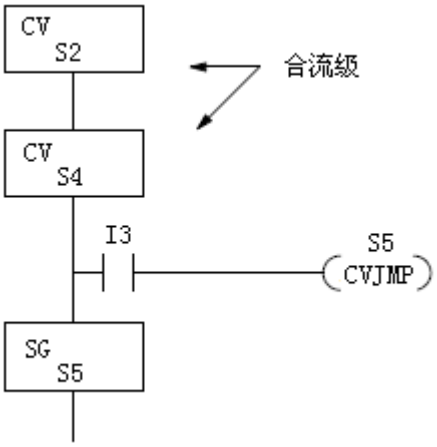
现在考虑相反的情况，就是进程合流。这其实就是停下多个进程，同时开始另一个进程。见下图，当 S2 和 S4 在某个时间点同时转移到 S5 中时，进程 A 和进程 B 合流。因此，S2 和 S4 是合流级。



合流级登记指令（CV）

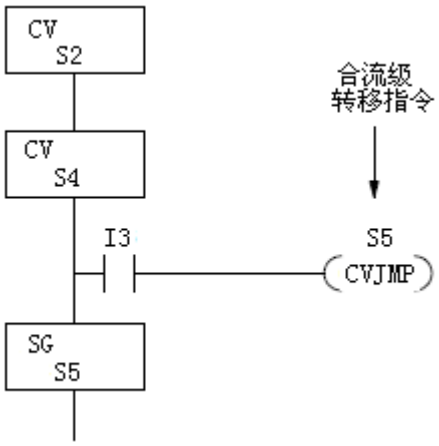
虽然合流原理很简单，但是它也带来新的问题：多个并行进程基本上不会同时完成，我们怎么会知道哪个并行的级（本例中是 S2 和 S4）是最后一个完成的呢。这一点很重要，因为我们要确定怎么转移到级 S5 中。

解决的方法是将合流级作成合流移行条件，我们使用级合流指令（CV）来实现。见右边的例子，合流级 S2 和 S4 被放在一起组成一个合流级群，两个合流级之间不能添加任何指令。转换条件（本例中为 I3）必须放在最后一个合流级下方。当合流级群中所有的级被激活后（为 ON），转换条件才能接通。



合流级转移指令（CVJMP）

当级群中所有的级被激活后，转换条件才能接通。合流级转移指令仅同合流级登记指令配合使用。右图中，当转换条件 I3 闭合后，将转移到级 S5，而且 CVJMP 指令将级组中所有级自动复位。因此，CVJMP 指令是一个强大的指令。



下面是级合流指令使用的一些要求和技巧：

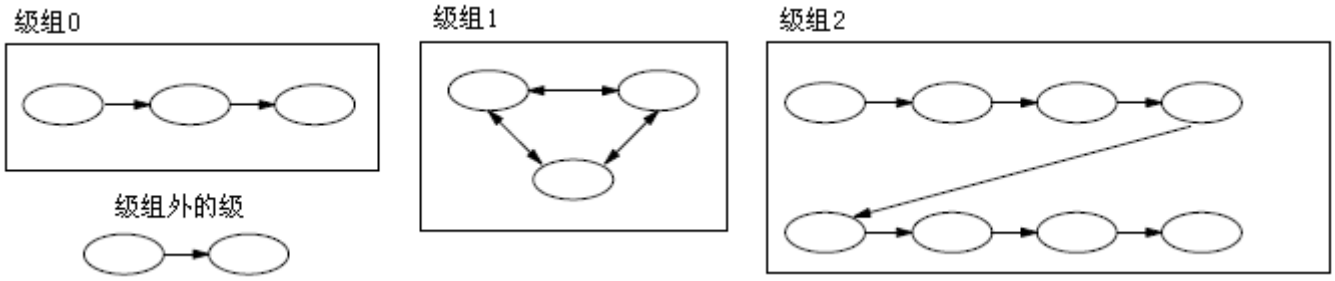
- 合流级是一个或多个并行进程的最后一级。合流级转移意味着一个特定进程的结束，也代表一个等待点，等待其它所有平行进程全部结束。
- 合流级群中最多可有 16 个级，也就是说，最多可有 16 个级合流到一个级。
- 合流级群中的级在程序中必须放在一起，合流级之间不能添加任何指令。
- 在一个合流级群中，级的激活可以是任何顺序，不用管哪个级是最后激活，因为级群中的所有级都要被激活之后，最后一个合流级才能接通。
- 级群中的最后一个级中可以有逻辑指令，但是，只有级群中所有的级被激活后，这个逻辑才能执行。
- 合流级转移指令（CVJMP）同合流级登记指令配合使用，CVJMP 指令将级群中的所有级复位，将指定的级激活。
- CVJMP 指令只能同合流级登记指令配合使用，同普通级和初始级式指令使用时是无效的。
- CV 和 CVJMP 指令不能用于子程序和中断程序中。

20.9 大型程序的管理

一个级中可以有很多行程序，也可以只有一两行程序。良好的程序设计应确保每个级的平均程序行数比较少。但是，大型的应用程序还是会创建大量的级。这里介绍级组的概念，将相关的级组成一个组，叫级组。

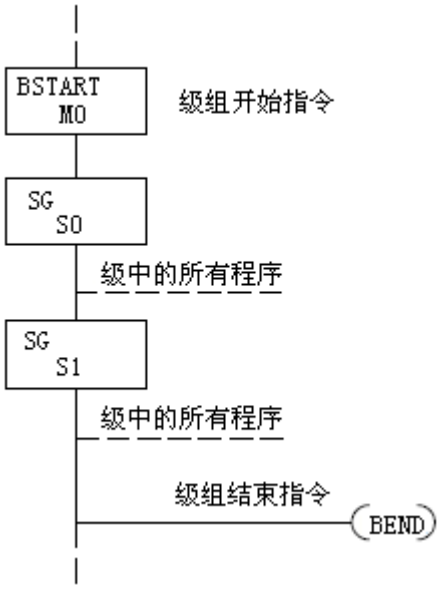
级组指令（BSTART、BEND）

级组是包含级的程序的一部分。下图中，每个级组有一个编号。同级一样，级组状态可为 ON 或 OFF。级组中级如何从一个级转移到另一个级是没有限制的。注意，使用级组不需要程序中的每个级都在级组中，见下图“级组外的级”。



一般认为一个有 20 个级以上的程序已经足够大，可以使用级组了，但是级组的使用不是强制的。使用级组时，级组的数量应为两个以上，因为使用一个级组的优点基本上可以忽略不计。

使用特殊的开始和结束指令将级组与其它程序分隔开。右图中，BSTART 指令标志着级组的开始，BEND 指令标志着级组的结束。级组中的级（右图中是 S0 和 S1）及级中的程序组成了级组。



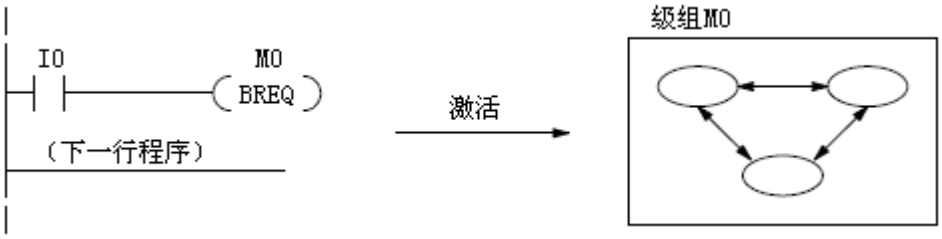
注意，级组指令的操作数（本例中是 M0），级组指令使用一个内部线圈作为操作数，以使程序的其它部分可以控制一个级组。级组指令使用的任何一个内部线圈在程序中都不可再用作别的输出。

注意，级组中的级必须是普通的级（SG）或合流级（CV），不可使用初始级（ISG）。级组中的级编号可以是任何顺序，且完全独立于级组的编号。

级组请求指令（BREQ）

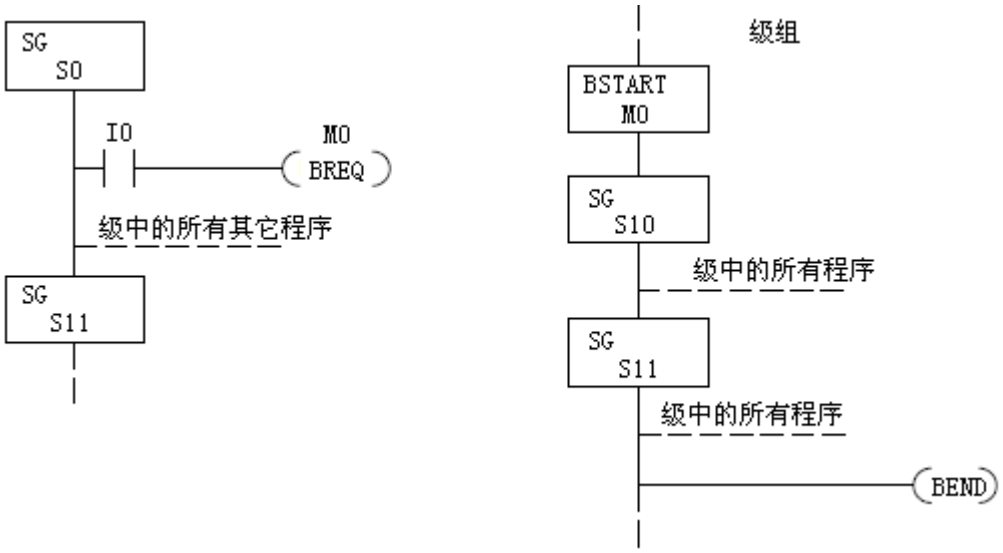
BREQ 指令的作用是激活一个级组。在上电时或是在编程→运行模式转换时，所有的级组及其里面的级都没有激活。BREQ 指令是一种输出线圈，见下图。当 I0 闭合时，BREQ 指令将操作数为 M0 的级组激活。当 BREQ 指令变为 OFF 时，相应的级组及其里面的级均变为 OFF。

不要将级组请求指令的操作同子程序调用指令的操作混淆。BREQ 线圈执行后，程序继续执行下一行程序。只要程序执行到 BREQ 指令指定的级组，级组中的程序就会执行，因为级组是 ON 状态。同样的，不要将 BREQ 指令同级转移指令（JMP）混淆。



当级组状态变为 ON 后，级组中的第一个级的状态在同一个扫描周期中自动变为 ON。级组的第一个级是程序中 BSTART 指令下方紧邻 BSTART 指令的一个级，因此这个级同初始级有类似作用。

BREQ 指令可在程序中多次使用。很明显，BREQ 指令的第一次执行必须在一个级组之外，因为级组的初始状态是 OFF。然后，BREQ 指令可在一行普通的程序中执行，或者它可以在一个激活的级中执行，见下图。注意，不论是将 BREQ 指令状态转换为 OFF 或是将 BREQ 指令所在的级状态转换为 OFF，都会将相应的级组状态转换为 OFF。也可以控制一个 BREQ 指令在另一级组中的级组。



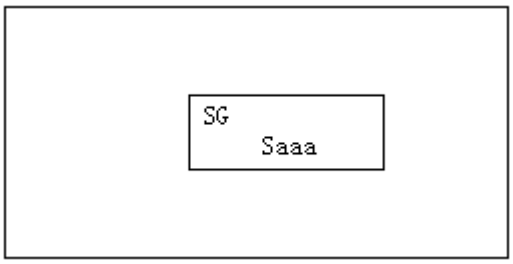
注意：级组可放在 BREQ 指令的之前或之后。

BREQ 指令有多种用法，因此很难说哪种用法最好。只需记住级组的目的是将相关联的级做成一个级组，以帮助用户组织应用程序。注意，初始级必须放在级组的外面。

第 21 章 级式指令

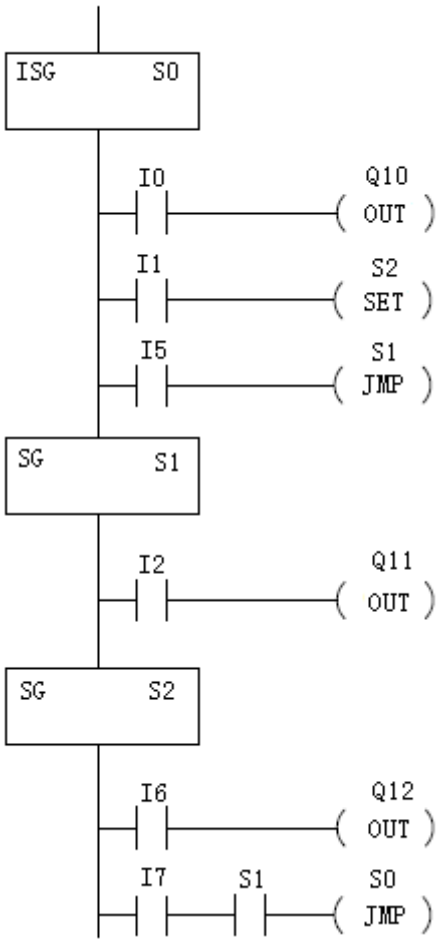
21.1 级登记指令（SG）

SG 指令用于创建级式程序。级是可以通过转移逻辑（JMP 或 SET）激活的程序段。转移逻辑（JMP 或 RST）执行后的下一个扫描周期，级状态变为 OFF。



操作数类型	D4-454 范围
A	aaa
级 S	0-1777

下面的例子是一个简单的级式程序例子。例中使用了初始级、级和转移指令。

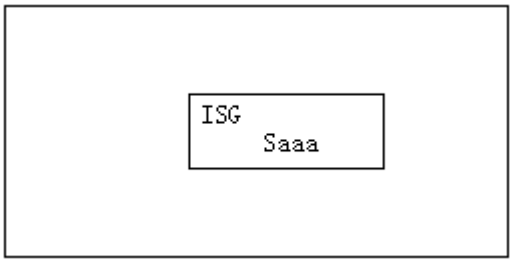


语句视图

```
ISG S0
LD I0
OUT Q10
LD I1
SET S2
LD I5
JMP S1
SG S1
LD I2
OUT Q11
SG S2
LD I6
OUT Q12
LD I7
AND S1
JMP S0
```

21.2 初始级登记指令（ISG）

初始级登记指令通常用于级式程序的开头。在 CPU 进入运行模式后，初始级就被激活。初始级也可被状态为 ON 的级中的转移逻辑（JMP 和 SET）激活。只要级号不重复，程序中可以使用多个 ISG 指令。

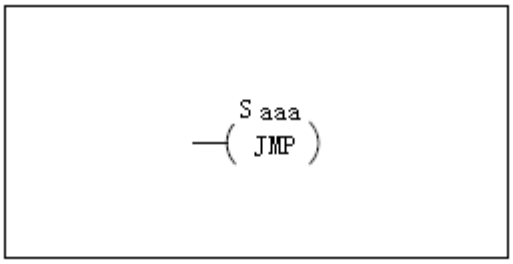


操作数类型		D4-454 范围
	A	aaa
级	S	0-1777

注意：如果初始级被设置了停电保持，那么上电后，初始级在首次扫描时将保持其断电前的状态而不是被置为 ON。

21.3 级转移指令（JMP）

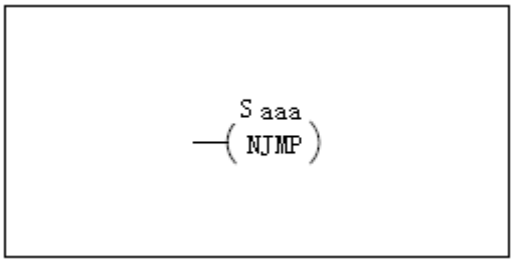
级转移指令是允许程序在所属级的 ON 状态且转移条件成立时，向指定的级转移的指令。当转移的条件成立时，级转移发生。JMP 指令执行后的下一个扫描周期，其所属级状态被置为 OFF。



操作数类型		D4-454 范围
	A	aaa
级	S	0-1777

21.4 条件不成立时级转移指令（NJMP）

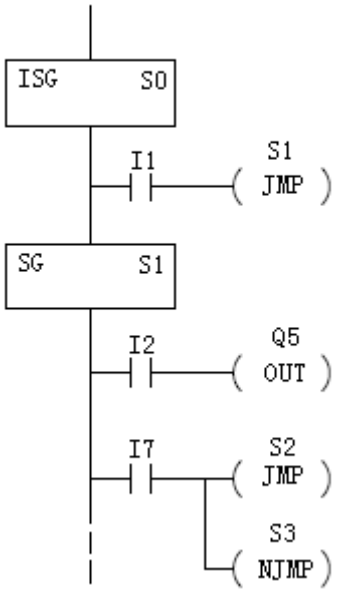
NJMP 指令是允许程序在所属级的 ON 状态且转移条件不成立时，向指定的级转移的指令。当转移的条件不成立时，级转移发生。NJMP 指令执行后的下一个扫描周期，其所属级状态被置为 OFF。



操作数类型		D4-454 范围
	A	aaa
级	S	0-1777

在下面的例子中，CPU 开始执行程序时，仅 ISG S0 为 ON。当 I1 为 ON 时，程序执行从初始级 S0 转移到 S1。在级 S1 中，当 I2 为 ON 时，输出 Q5 被置为 ON。如果 I7 为 ON，程序执行将从 S1 转移到 S2；如果 I7 为 OFF，程序执行将从 S1 转移到 S3。

语句视图




```
ISG S0
LD I1
JMP S1
SG S1
LD I2
OUT Q5
LD I7
JMP S2
NJMP S3
```

21.5 合流级登记指令（CV）

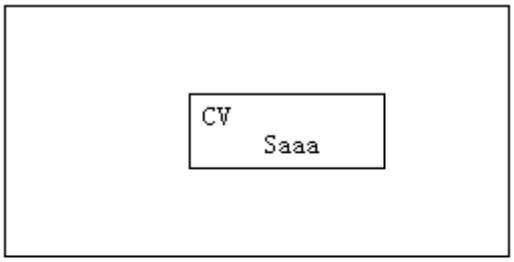
21.6 合流级转移指令（CVJMP）

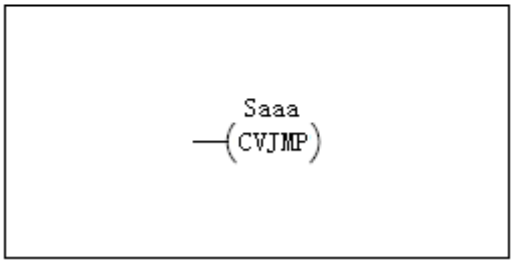
合流级登记指令用于将几个级定义为合流级，由 CV 指令登记的级群称为合流级群。

当合流级群中的所有合流级为 ON 时，CVJMP 指令及最后一个合流级后的逻辑将被执行。最后一个合流级中的程序执行前，前面所有合流级的状态必须为 ON。CVJMP 指令执行后的下一个扫描周期，所有合流级状态被置为 OFF。

只能在最后一个合流级登记指令和 CVJMP 指令之间添加逻辑指令。可以使用多个 CVJMP 指令。

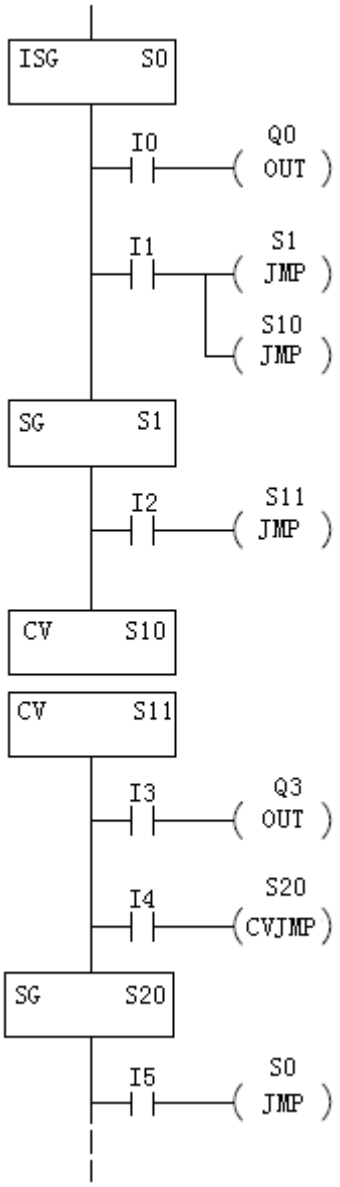
合流级只能用于主程序中，不可用于子程序或中断程序中。





操作数类型		D4-454 范围
	A	aaa
级	S	0-1777

下面的例子中，当合流级 S10 和 S11 都为 ON，并且 I4 为 ON 时，CVJMP 指令执行。CVJMP 指令将 S10 和 S11 置为 OFF，将 S20 置为 ON。然后，如果 I5 为 ON，程序执行将返回到初始级 S0。

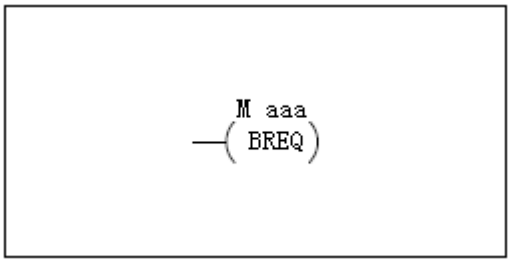


语句视图

```
ISG S0
LD I0
OUT Q0
LD I1
JMP S1
JMP S10
SG S1
LD I2
JMP S11
CV S10
CV S11
LD I3
OUT Q3
LD I4
CVJMP S20
SG 20
LD I5
JMP S0
```

21.7 级组请求指令（BREQ）

BREQ 指令用于激活一个级组。BREQ、BSTART、BEND 指令必须一起使用。关于 BREQ 指令需要知道以下几点：

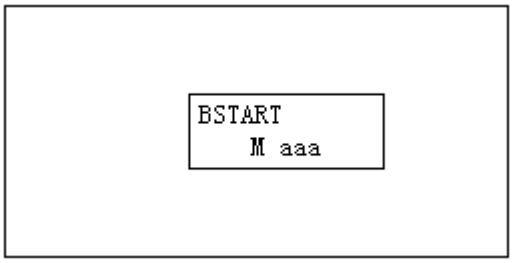


- 使用内部线圈作为操作数，不能将级作为操作数。被使用的内部线圈在程序中不能再用作别的输出。
- 必须保持 ON 状态：BREQ 指令实际上控制 BSTART 与 BEND 指令之间的所有程序，即使在级组程序已经开始执行之后，BREQ 指令也必须保持 ON 状态，否则级组中所有的级状态自动变为 OFF。如果 BREQ 指令或是 BREQ 指令所在的级的状态变为 OFF，那么所定义的级组中的所有级状态将变为 OFF。
- 激活级组中第一个级：当 BREQ 指令执行时，自动激活 BSTART 指令后的第一个级。

操作数类型		D4-454 范围
	A	aaa
中间继电器	M	0-3777

21.8 级组开始指令（BSTART）

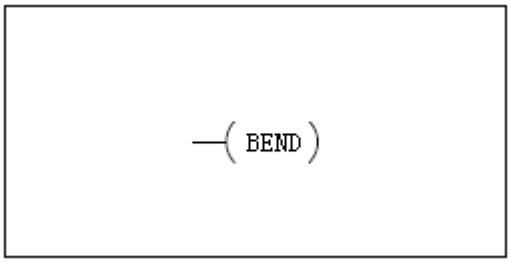
BSTART 指令是一个级组开始的标志，级组中的级状态可被成组置为 ON 或 OFF。BSTART 指令的下一个指令必须是 SG 指令。级组中不能使用初始级。BSTART 指令中使用的内部线圈在程序中不能再用作别的输出。



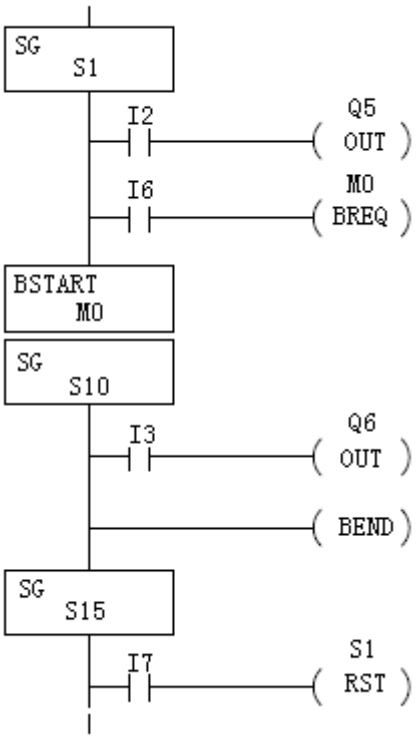
操作数类型		D4-454 范围
	A	aaa
中间继电器	M	0-3777

21.9 级组结束指令（BEND）

BEND 指令是级组结束的标志。这个指令没有操作数。



下面的例子中，当级 S1 为 ON 并且 I6 为 ON 时，BREQ 指令自动激活级 S10。如果 BREQ 指令变为 OFF，或者其所在的级状态变为 OFF，那么 BSTART 与 BEND 指令之间的所有级自动变为 OFF。级 S15 中，I7 可将 S1 复位，因此可将 BREQ 指令置为 OFF，也因此可将级组中的所有级复位。



语句视图

```
SG S1
LD I2
OUT Q5
LD I6
BREQ M0
BSTART M0
SG S10
LD I3
OUT Q6
BEND
SG S15
LD I7
RST S1
```

21.10 级式编程常见问题

下面列出了级式编程中经常会问到的问题。

问：级式编程提供了什么通用梯形图编程不具备的功能？
答：级可以在编程之前识别过程的所有状态。这种方法将梯形图程序分成不同块，更有组织性。这些程序块就是级，只在过程需要的时候激活。许多过程都可以分成连续的级，以事件作为转换条件。

问：级与软件的子程序是否相同？
答：不同。子程序由主程序在需要的时候调用，只执行一次，然后返回调用点继续执行主程序。而级是主程序的一部分。一个级代表过程的一个状态。当级激活时，每个 CPU 扫描周期都执行，直到级不处于激活状态。

问：什么是级状态位？
答：级状态位为 CPU 映像寄存器的一个位，代表级的实时状态是处于激活还是未激活。例如，级 0 的状态位为 S0。如果 S0=0，则级 0 下的梯形图指令在每次 CPU 扫描时被略过；如果 S0=1，则级 0 下的梯形图指令在每次 CPU 扫描时执行。通过对级状态位状态的控制，在某一部分程序监控另一部分程序。

问：级如何激活？
答：有三种方法：

- 如果级为初始级(ISG)，则上电后自动激活；
- 在其他级执行 JMP 指令，JMP 指令指定的级激活；
- 在程序行执行 SET 指令(例如 SET S0)。

问：如何使级不激活？
答：有三种方法：

- 标准级(SG)上电后就不处于激活状态；
- 在本级执行 JMP 指令，将本级状态位重置为 0；
- 在程序行执行 RST 指令(例如 RST S0)。

问：级变换的能流技术是什么？
答：连接相邻级（程序中直接级上方或下方）的能流方法实际上与上述级中执行的 JMP 指令指向下一个级相同。能流转换在 KPP 中编辑更困难，故从前面两个问题中独立出来列在此处。

问：我可以使级只激活一个扫描周期吗？
答：可以，但这不是使用级的目的。要实现只激活一个扫描周期，可以在该级的最后放置 JMP 指令，在转换到新的级之前，该级下的梯形图指令会执行一个扫描周期。

问：JMP 指令与软件中的 GOTO 语句不是很像吗？

答：不，有很大的不同。GOTO 指令将立刻执行 GOTO 指令指定地址的程序，而 JMP 指令仅将当前级的状态位复位，并将 JMP 指令指向的级状态位置位。级状态位是 0 还是 1，决定了相应的级是否处于激活状态。JMP 指令执行后有下面的结果：

- 当 JMP 指令执行时，当前级剩余的指令即使在 JMP 指令之后，还是会执行。在下一个扫描周期，由于本级不处于激活状态，本级的指令不会执行；
- JMP 指令指向的级将在接下来扫描到时执行。如果该级位于当前级之后，将会在同一个扫描周期得到执行；如果该级位于当前级之前，将会在下一个扫描周期得到执行。

问：何时使用 JMP 指令，何时使用置位/复位级状态位？

答：根据状态图使用这些指令：

- 在状态转移时使用 JMP 指令，从一个状态转移到另一个状态；
- 在当前级产生新的并行状态或并行级序列，或者管理级开始一个状态序列时，使用 SET 指令置位级状态位；
- 在当前级是过程的最后状态并且已完成任务，或者管理级结束一个状态过程，使用 RST 指令复位级状态位。

问：什么是初始级？什么时候使用初始级？

答：初始级(ISG)在上电时自动激活，之后与其他标准级以相同方式动作。如果需要，可以有多个初始级。可以将需要一直运行的梯形图指令放在初始级，也可以将初始级作为起点。

问：可以将梯形图指令放在级之外，使之一直运行吗？

答：是可能的，但这不是好的编程习惯。可以将需要一直运行的梯形图指令放在初始级，不在该初始级使用 JMP 指令，也不使用 RST 指令将该级复位。可以在适当的时候，用 SET 指令将其他级激活。

问：可以同时有多个级处于激活状态吗？

答：可以，这也是经常会发生的事情。重要的是将应用程序组织为几个独立的过程，每个过程作为一个级。好的过程设计是连续的，每个时刻只有一个级。但无论如何，程序中所有的级都可以同时处于激活状态。

第 22 章 凸轮控制介绍

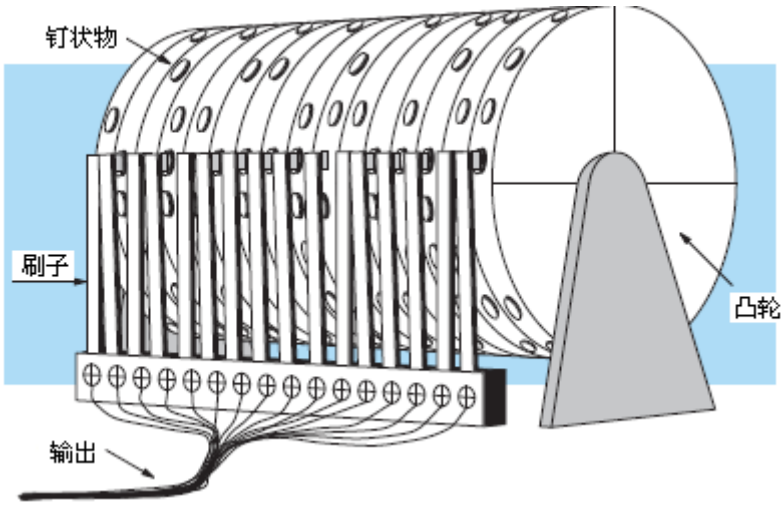
22.1 简介

凸轮控制术语

凸轮控制指令适用于包含一定步骤的重复性过程控制。简单的一条凸轮控制指令，可以完成许多行梯形图程序才成完成的工作。因此，凸轮控制指令可以省去很多编程和调试的时间。

这里通过下图的原始机械凸轮，介绍一下凸轮控制指令相关术语。机械凸轮表面通常以特定的模式分布着一些钉状物，代表着一组期望的机械控制动作。电动机或电磁铁在特定的时间对凸轮旋转一个精确的量。在旋转过程中，固定的刷子感应钉状物的存在（存在时=ON，不存在时=OFF）。钉状物与刷子的碰触使电触点进行开合，产生了电信号输出，这个输出信号可用来控制机器的开关。

通常情况下凸轮一周有一定数量的位置，叫做步。每步代表一些进程步骤。上电时，凸轮复位到一个指定的步。凸轮从一步到下一步是基于一个定时器或是一些外部事件。在一些特定的条件下，机器操作者可以使用凸轮驱动机构的点动控制，手动增加凸轮的步。固定的刷子和钉状物碰触生成的唯一的 ON/OFF 序列，用于机器的特定控制。因为凸轮是环形的，因此每一圈的序列都是自动重复的。凸轮的应用千差万别，有的应用一秒转一圈，有的应用一星期转一圈。



相比机械凸轮，电子凸轮有更多的优点，例如，它有预设功能，而机械凸轮不可能有这个功能。预设功能可使控制从当前步直接跳到任何其它步。

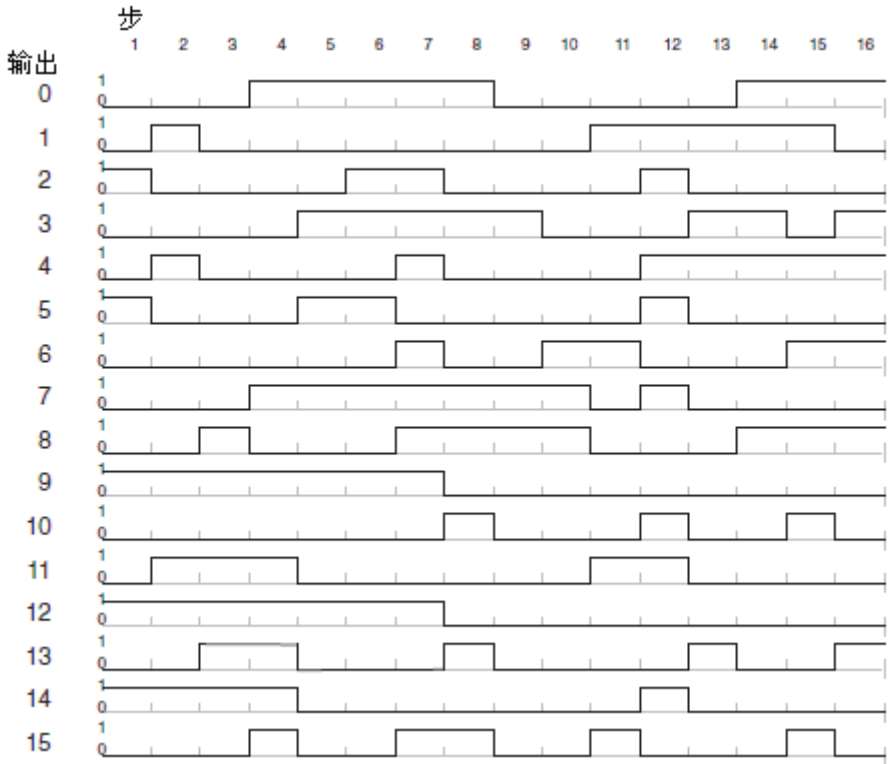
凸轮控制图表

为了编辑的方便，电子凸轮在编程软件和本手册中以图表的形式表现，见下表。每一行代表一步，编号是 1-16。每一列代表一个输出，编号是 0-15。实心点代表刷子碰触到钉状物（ON 状态），空心点代表刷子没有碰触到钉状物（OFF 状态）。

步骤	输出 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	○	●	○	●	○	○	●	○	○	○	●	○	○	●	○	○
2	○	●	○	●	●	○	●	○	○	○	○	●	○	○	●	○
3	○	●	●	●	●	○	●	●	○	○	○	○	○	○	○	○
4	●	●	●	●	●	○	●	○	●	○	○	○	○	○	○	●
5	○	○	○	●	○	○	●	○	●	○	●	○	●	○	○	●
6	○	○	○	●	○	○	●	○	●	○	●	○	●	●	○	●
7	●	●	○	●	○	○	●	●	●	●	○	●	●	●	○	●
8	●	○	●	○	○	●	○	●	●	○	○	○	●	○	○	●
9	○	○	○	○	○	○	○	●	●	○	○	○	●	○	○	○
10	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	○
11	●	○	○	○	●	○	○	○	○	●	○	○	○	○	●	○
12	○	●	○	○	●	●	○	○	●	○	●	●	○	●	●	○
13	○	○	●	○	○	○	○	○	○	○	○	●	●	○	●	○
14	○	○	○	○	○	○	○	●	○	○	○	●	●	○	●	●
15	●	○	○	○	○	●	○	●	○	●	○	●	○	○	●	●
16	○	○	●	○	○	○	○	●	○	●	○	●	●	○	○	●

输出顺序

下图是上述的凸轮控制图表的 ON/OFF 输出顺序表，对比二者，你会发现，两者是完全相等的。



22.2 步转移

凸轮控制指令类型

D4-454 CPU 中有四种类型的凸轮控制指令：

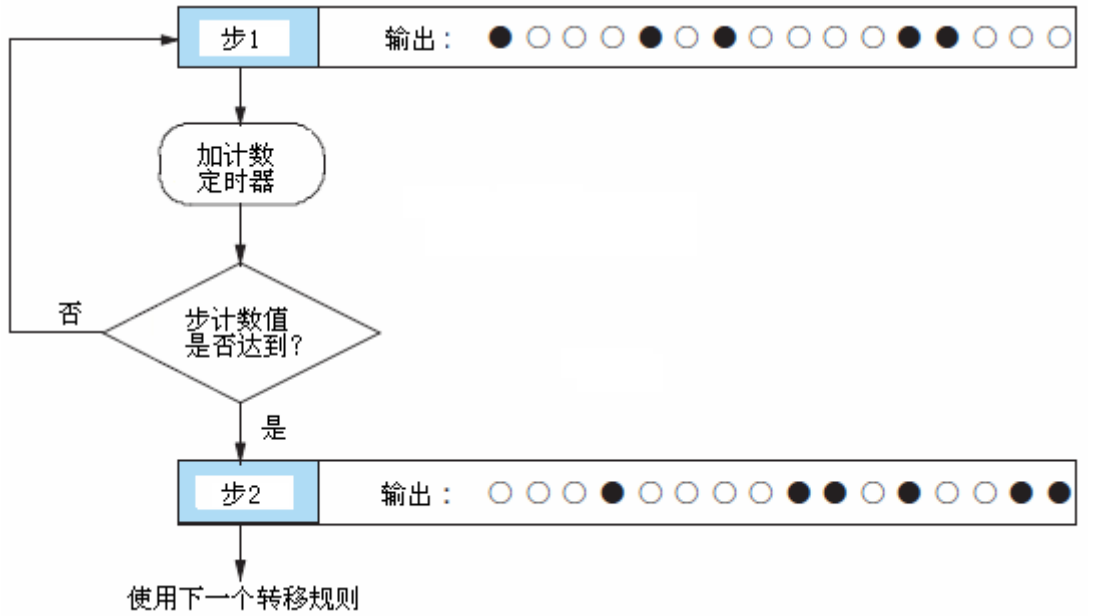
- 时间驱动型凸轮（DRUM）
- 时间/事件型凸轮（EDRUM）
- 带掩码事件驱动型凸轮（离散点输出）（MDRMD）
- 带掩码事件驱动型凸轮（字输出）（MDRMW）

这四个凸轮指令都包含了基于时间的步转移，其中三个还包含了基于事件的步转移，以及其它选项包括将输出定义为字或单个位和输出屏蔽（单个输出禁用/启用）。

每个凸轮有 16 步，每步有 16 个输出，输出可以是 Q 或 M 线圈。

仅基于时间步转移

步转移仅基于时间。每步都有各自的转移条件，转移条件在输入凸轮指令时已经设定完成。下图演示了时间驱动型步转移是如何工作的。



凸轮停留在步 1 一个特定的时间（可以编程设定）。定时器的时基可以编程设定，范围是 0.01-99.99s。这样相当于确立了每个“滴答时钟”或停留时间的分辨率。每步使用相同的时基，但是有各自独立的计数值，可以编程设定。凸轮在每步的停留时间由下面的公式算出：

每步的停留时间=0.01s×时基×每步的计数值

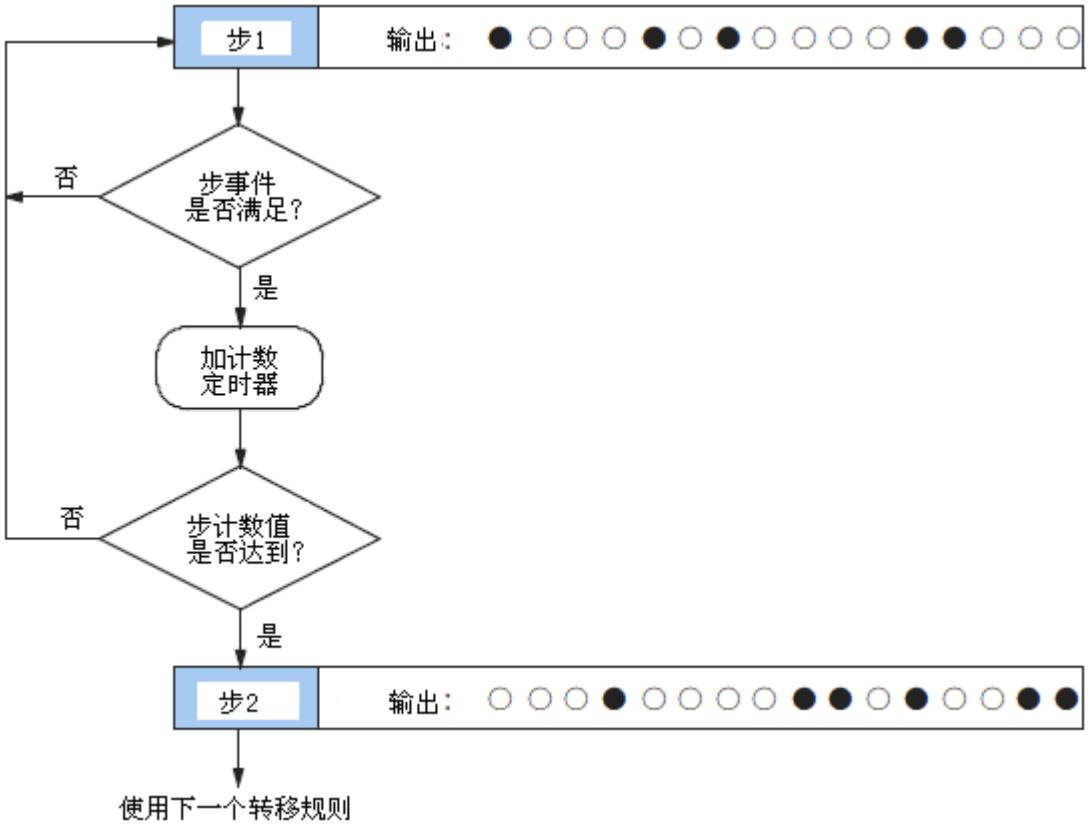
例如，如果步 1 是 5s 的时基和 12 的计数值，则凸轮将在步 1 停留 60s 的时间。每步的最大停留时间由下式算出：

每步最大停留时间=0.01s×9999×9999=999,800 秒=277.7 小时=11.6 天

注意：如果是第一次选择时基，一个好的方法是:使时基=最短步的停留时间×1/10。然后就可以以 10%的增量调整最短步的停留时间了。其它时间较长的步可以以更小的增量优化停留时间（以百分比计算）。还要注意的是，凸轮控制指令每个 CPU 扫描周期执行一次，因此，如果时基比 CPU 扫描时间快很多的话是毫无意义的。

时间/事件型步转移

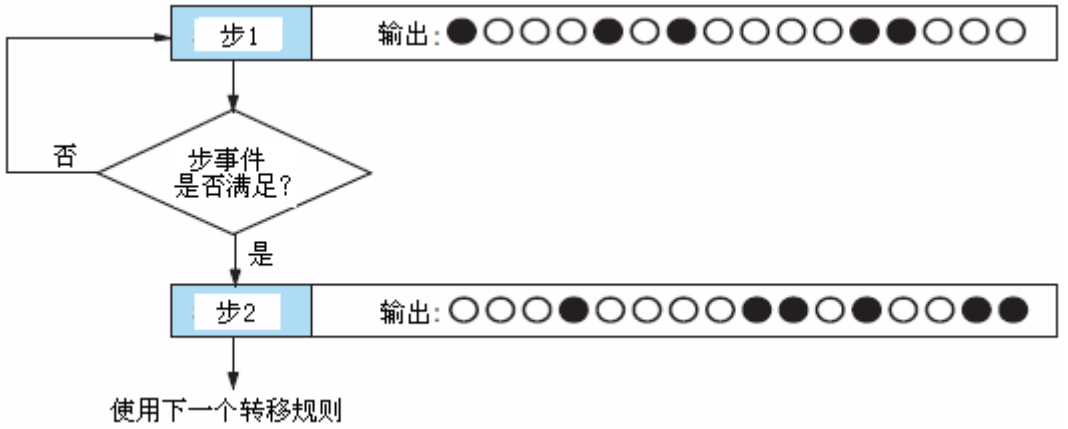
步转移基于时间/事件。下图演示了时间/事件驱动型转移是如何工作的。



当凸轮进入步 1，输出图表如上图所示。然后开始检测此步的步事件是否满足。步事件可以是 I、Q、M 等开关量。假设我们选择 I0 为步 1 的事件。如果 I0 为 OFF，则凸轮停留在步 1。当 I0 为 ON 时，事件条件满足，计数器开始计数。只要 I0 保持为 ON，计数器就计数。当步 1 的计数达到时，凸轮将进入步 2，输出将立即改变为步 2 的输出。

仅基于事件步转移

步转移不一定都需要事件和时间两个条件，可以只使用两个条件的中的其中一个，甚至一个凸轮中的所有步可以混合使用转移规则。例如，步 1 的转移条件可以是一个事件，步 2 的转移条件是时间，步 3 的转移条件是时间和事件。此外，还可以选择使用 16 步中的几步或者 16 个输出中的几个输出。



计数器分配

每个凸轮控制指令要使用四个计数器。对凸轮控制指令进行编程时，选择第一个计数器编号，凸轮指令自动使用接下来的三个计数器。当凸轮完成一个循环时，第一个计数器的计数器位变为 ON；当凸轮被重置时，第一个计数器的计数器位变为 OFF。这些计数器的计数值和计数器位精确地反应了凸轮指令的进程，可以在程序中监控这些数据。

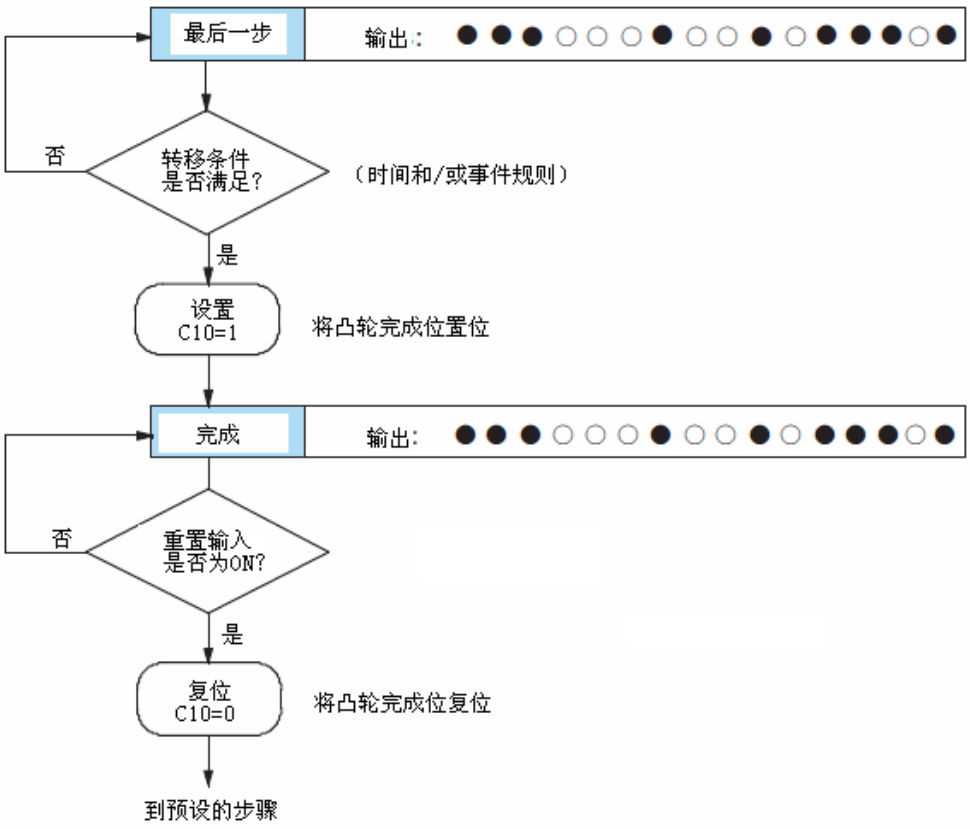
假设编程时使用一个凸轮的 8 个步，选择 C10 为第一个计数器（注意计数器编号是八进制的），计数器的使用见右边的表格。最右边一栏是典型值，下面会进行讲解。

计数器分配			
C10	步的计数	R1010	1528
C11	定时器值	R1011	0200
C12	预设定步	R1012	0001
C13	当前步	R1013	0004

CA10 的值 1528 表明当前步中的计数值是 1528，当前步是步 4，由 CA13 的计数值表示。如果编程时设定步 4 的计数值是 3000，那么此步的计数刚好完成一半。CA11 是计数定时器，单位是 0.01s。因此，数值的最小变化是 0.01s。200 表示计数到 1528 所用的时间是 2s（0.01×200）。CA12 保持编程时在凸轮控制指令中设定的步预设值。当凸轮的重置输入为 ON 时，凸轮回到步 1。仅当程序向其写入数据或凸轮指令被编辑又或者程序重启时，C12 的值才改变。当凸轮的循环结束时，C10 的计数器位变为 ON，当凸轮被重置时，C10 的计数器位变为 OFF。

最后一步完成

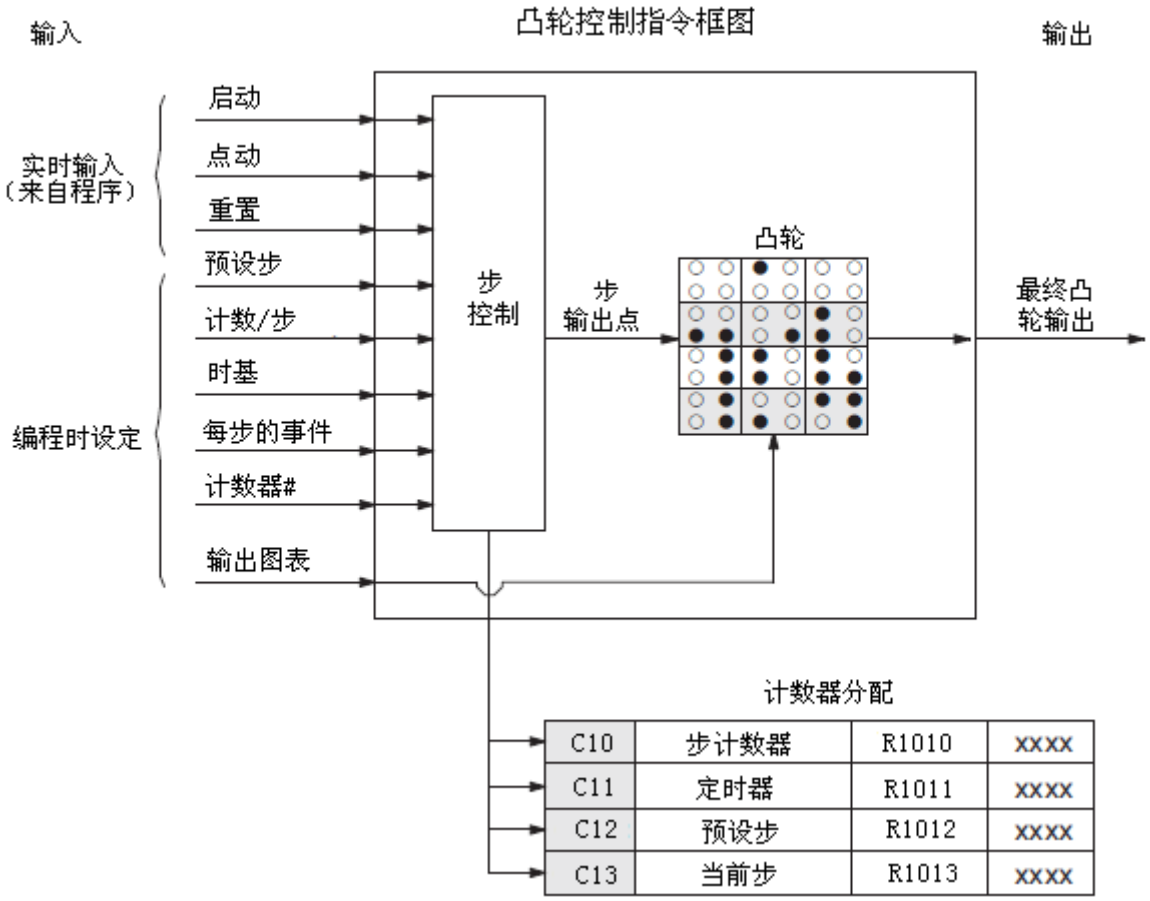
凸轮控制的最后一步可以是任何一步，因为只使用部分凸轮是有效的。见下图，当最后一步的转移条件满足后，凸轮将相应的计数器（本例中为 C10）位置 ON，这个计数器编程时在凸轮控制指令中设定。此后进入到“凸轮完成”的状态，此时的凸轮输出仍保持最后一步的输出。凸轮的一个循环完成时，启动和外部事件输入失效。当重置输入变为 ON 时（或 CPU 完成编程→运行模式的转换时），凸轮离开“凸轮完成”状态。此时，凸轮完成位（本例中为 C10）被复位，然后凸轮直接进入预设的步中。



22.3 凸轮操作概述

凸轮控制指令框图

凸轮操作指令的输入和输出，见下图。



凸轮控制指令接受几个输入用于步控制，即凸轮的主要控制。这些输入及其功能如下：

- 启动：仅当重置输入为 OFF 时，启动输入有效。当启动输入为 ON 时，时间驱动型凸轮的定时器开始计时，事件驱动型的凸轮检测事件输入。当启动输入为 OFF 时，凸轮停止在当前的状态（重置输入必须保持为 OFF），凸轮输出保持当前的 ON/OFF 状态。
- 点动：仅当重置输入为 OFF 时，点动输入有效（启动输入可以是 ON 或是 OFF）。当点动输入发生 OFF→ON 的转换时，凸轮转移到下一步（仅 EDRUM 指令支持这个输入）。
- 重置：重置输入的优先级比启动输入高。当重置输入为 ON 时，凸轮转移到预设步。当重置输入为 OFF 时，启动输入可正常操作。
- 预设步：定义预设的步，范围是 1-16（通常情况下预设步为 1）。当重置输入为 ON 或 CPU 首次进入运行模式时，凸轮转移到预设的步。
- 计数/步：凸轮停留在每步的定时计数值。每步都有自己的计数值，计数/步是可选的。
- 时间值：计数/步定时器的当前值。
- 计数器#：凸轮用于步转移控制的四个连续计数器的首个计数器号。通过监控这些计数器可以监控凸轮的控制过程。
- 每步的事件：I,Q,M,S,T,C 或 SP 开关量输入都可以作为步转移输入。每步都有各自的事件，对于时间/事件驱动型凸轮，事件是可选的。

警告：CPU 在 RUN 模式下，凸轮的输出在任何时间都是启用状态。启动输入不必为 ON，并且重置输入不禁用输出。在进入运行模式后，凸轮根据当前步的输出图表自动将输出置 ON 或 OFF。起始步取决于计数器的设置：是停电保持还是非停电保持。

凸轮寄存器的上电状态

选择上电和编程→运行模式时的启动步是很重要的，请参考下面的图表。如果计数器存储器被设置为非停电保持，凸轮在每次上电和编程→运行模式转换时进入的步都是一样的。但是，如果计数器存储器被设置为停电保持，凸轮会保持它先前的状态。

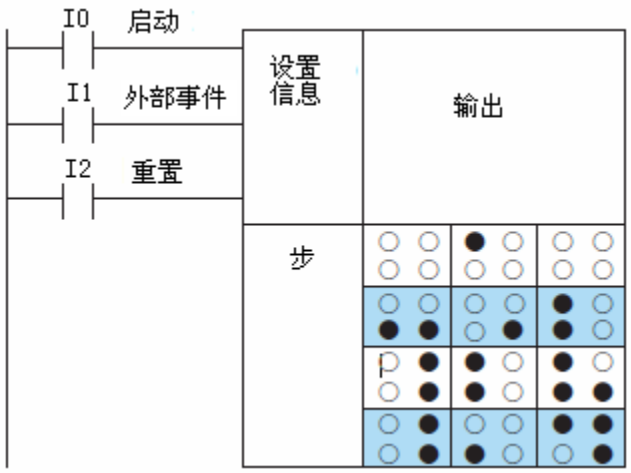
计数器号	功能	上电后的初始状态	
		非停电保持	停电保持
C（n）	步的计数值	初始值=0	先前状态（无改变）
C（n+1）	计数器定时值	初始值=0	先前状态（无改变）
C（n+2）	预设步	初始值=预设步	先前状态（无改变）
C（n+3）	当前步	初始值=预设步	先前状态（无改变）

相对来说凸轮循环时间较短的、上电时需要重置的应用，可选用非停电保持；循环时间较长的、上电时需保持停止时状态的应用，可选用停电保持。缺省设置是停电保持。因此，如果初始化 SPD 寄存器，存储器将是停电保持。

22.4 凸轮控制技术

凸轮控制输入

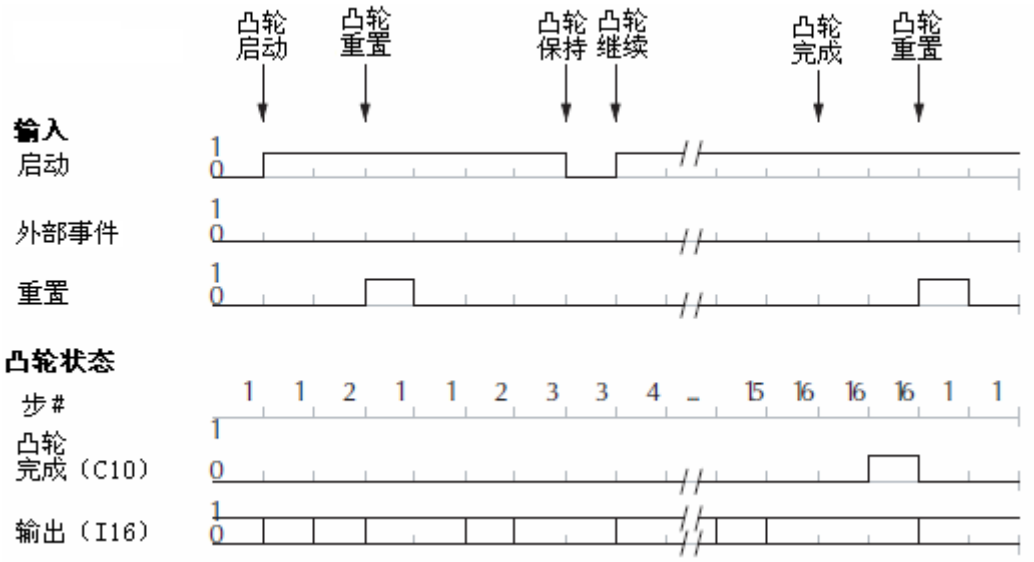
右图是一个简化的通用凸轮控制指令，有三个输入：启动，外部输入和重置输入（只有 EDRUM 指令支持外部事件输入）。第一个计数器的计数器位（例如 C10）为 ON 表明凸轮循环结束。



下面的时序图显示了一个时间驱动型凸轮的输入以及凸轮是如何响应的。当 CPU 进入运行模式，凸轮将步号初始化为预设步号（通常情况下为步 1）。当启动输入变为 ON 时，凸轮开始运行，等待一个事件和/或计数器计时的到来。

凸轮进入步 2 后，重置输入变为 ON，而启动输入仍然为 ON。由于重置信号的优先级高于启动信号的优先级，因此凸轮进入到预设步（步 1）。注意在重置时凸轮保持在预设步，并且那个步没有运行（对时间或定时器的响应）直到重置信号变为 OFF。

凸轮进入步 3 后，启动输入变为 OFF，在启动输入变为 OFF 的瞬间，凸轮的定时器停止计时,直到启动信号重新变为 ON。

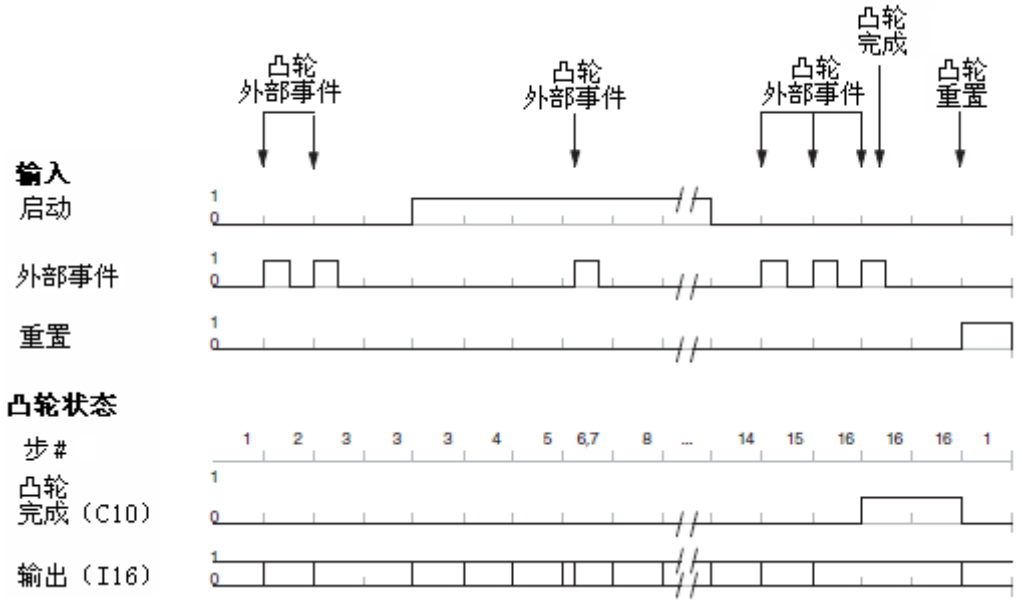


当凸轮完成最后一步后（本例中是步 16），凸轮完成位（C10）变为 ON，此时凸轮仍保持在步 16。当重置输入变为 ON 时，凸轮完成位（C10）变为 OFF，然后凸轮进入预设的步。

注意：上面的时序图中所有步的持续时间都相同,实际应用中,步的持续时间可以相差很大,可以在编程时设定。

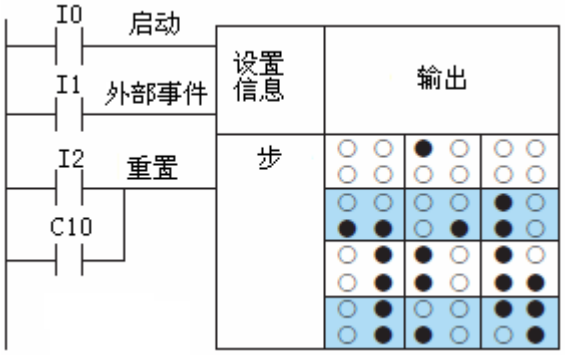
下面的时序图中，把重点放在外部事件输入对事件驱动型凸轮的操控。注意到，当外部事件输入发生 OFF→ON 转换时，凸轮的步就会增加一。启动输入可以是 ON 也可以是 OFF（但是重置输入必须为 OFF）。下图中，两个外部事件输入使凸轮进入了步 3。启动输入变为 ON 后，凸轮开始正常运行。在步 6 中，又一个外部事件输入发生，使凸轮进入了步 7，并将定时器复位到 0。由于启动输入仍然为 ON，凸轮正常运行到下一个步中。

当凸轮进入步 14 后，启动输入变为 OFF。两个或更多的外部事件输入使凸轮进入到步 16。但是注意，需要第三个外部事件输入将凸轮由步 16 转移到“凸轮完成”。最后，重置信号进入，将凸轮转移到预设步并将凸轮完成位置 OFF。



自复位凸轮

实际应用中通常需要凸轮完成一个循环后能自动开始下一个循环，使用凸轮完成位很容易实现这一功能。右图中，凸轮指令使用的计数器是 C10，因此，将 C10 同重置输入逻辑或。当最后一步完成后，凸轮将 C10 置 ON，这样就将凸轮转移到预设的步，同时也将 C10 置 OFF。I2 仍可用作一个手动重置触点。



初始化凸轮输出

只要 CPU 进入运行模式，凸轮输出便被启用。在编程→运行模式的转换后，凸轮马上进入预设步，预设步的输出被激活。如果应用需要上电时所有输出都为 OFF，可将预设步用作“重置步”，将所有输出都置为 OFF。

使用复杂步事件进行步转移

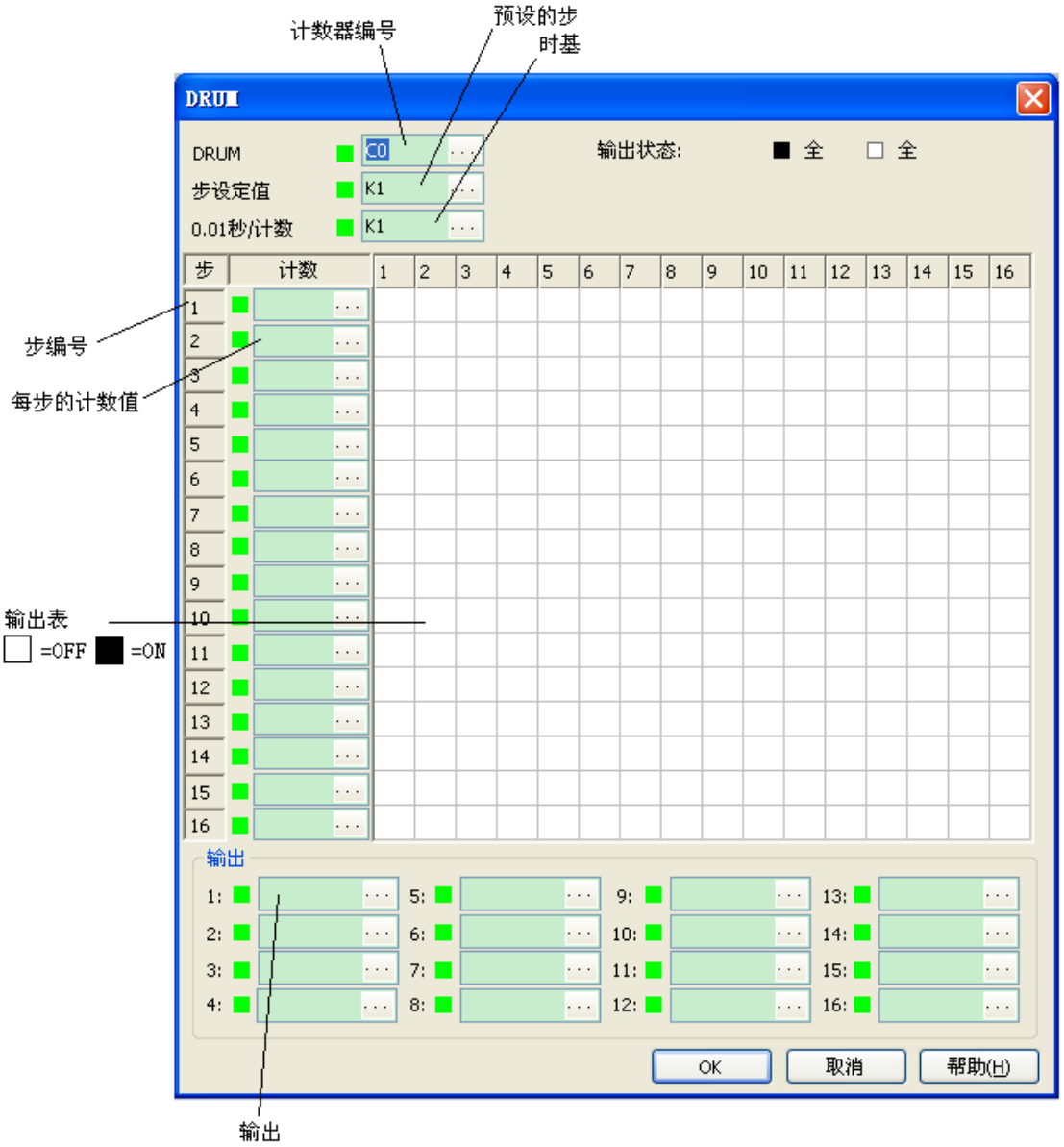
事件型凸轮指令每步只能输入一个事件触点用于步转移，但是这并不意味着用于步转移的事件只能是一个触点。例如，将 M0 触点作为凸轮控制指令其中一步的步转移事件，但是在程序里，可对 M0 进行编程，将其作为一个很多其它“事件”的逻辑输出触点。

第 23 章 凸轮控制指令

所有凸轮控制指令可以使用编程软件进行编程。

23.1 时间驱动型凸轮控制指令（DRUM）

DRUM 指令是最基本的凸轮控制指令。下图是 KPP 软件中 DRUM 的显示。



DRUM 指令有 16 步和 16 个输出，只基于时间进行步转移，每步的时基相同，计数值可单独设定。不使用的步编程时要设定步计数值=0（缺省值）。开关量输出可以是单独的 I、Q 或 M 输出，也可以不使用。输出格式可以通过 KPP 软件进行图形设置。

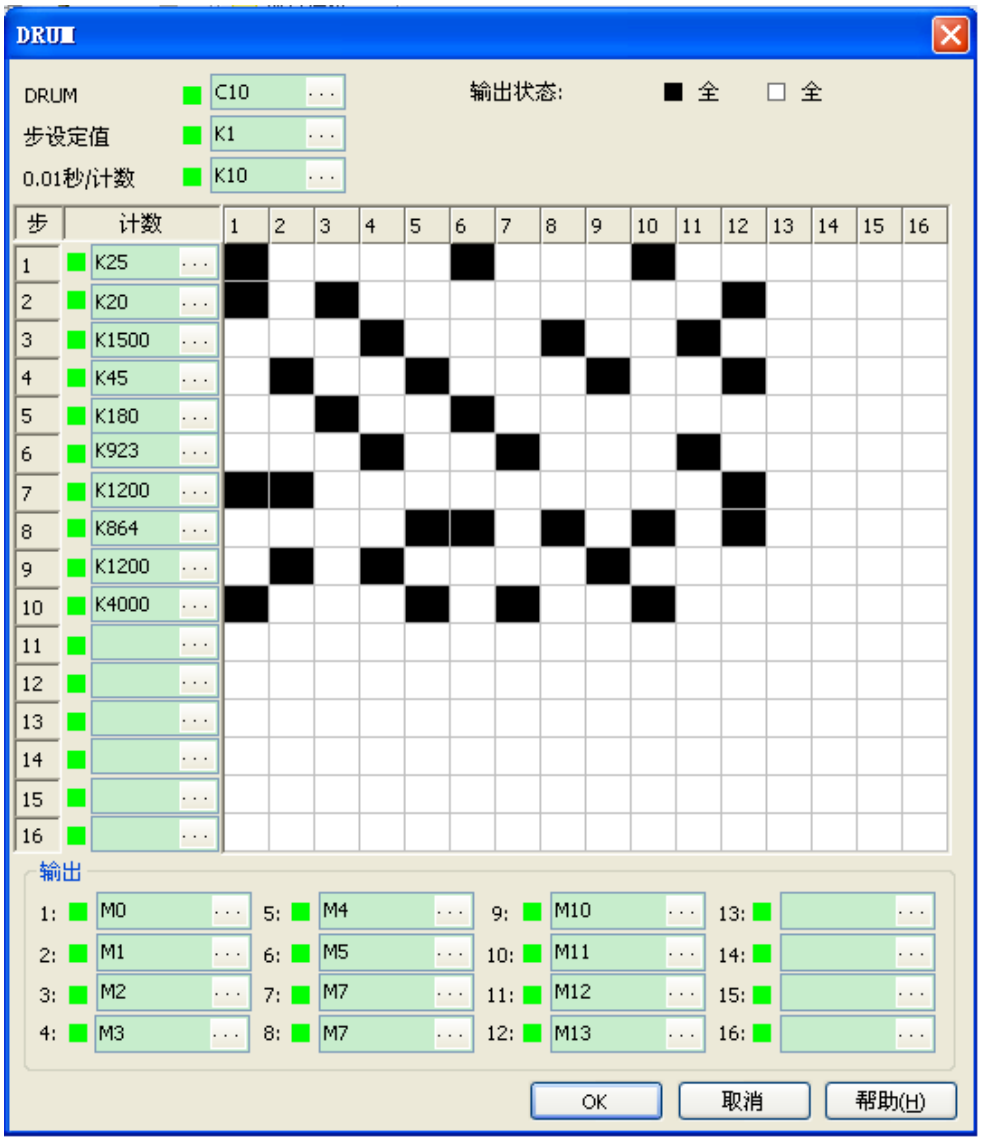
当启动输入为 ON 时，凸轮的定时器开始计时，最后一步完成后或重置输入为 ON 时计时停止。在 CPU 进行编程→运行的模式转换后或重置输入为 ON 时，凸轮进入预设定的步。

凸轮参数	数据类型	范围
计数器编号	C	0-374
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
开关量输出	I,Q,M	所有（附录 1）

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向 CA(n+2)中写入预设定的步。其他三个计数器仅做监控用。

计数器编号	n 的范围	功能	计数器位功能
C(n)	0-374	每步的计数	C(n)=凸轮完成
C(n+1)	1-375	定时器值	C(n+1)未使用
C(n+2)	2-376	预设定的步	C(n+2)未使用
C(n+3)	3-377	当前步	C(n+3)未使用

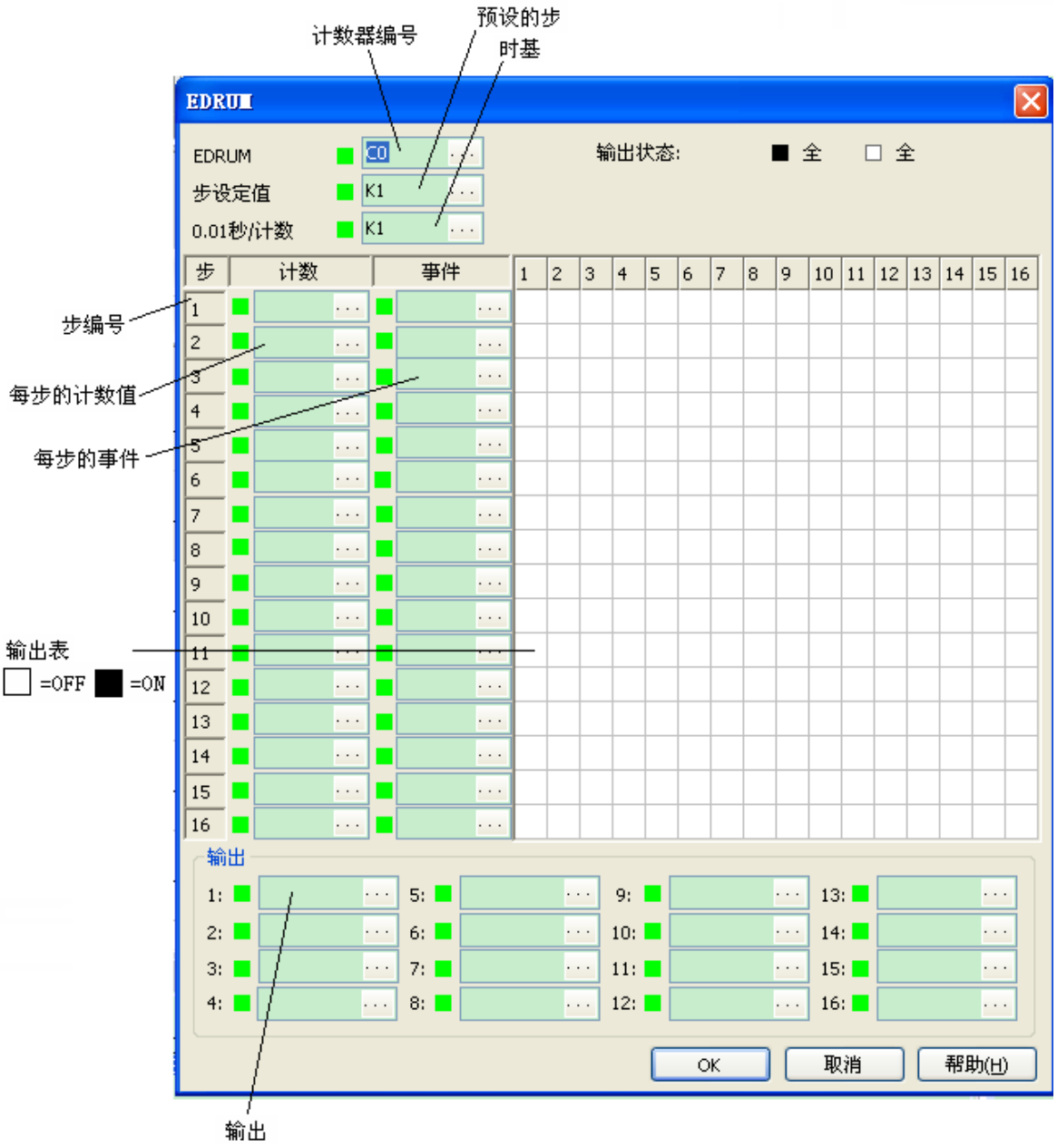
下面的程序例中，使用了步 1-10 和 16 个输出中的 12 个输出。预设定的步是步 1。时基是 K10×0.01=0.1s。因此，步 1 的停留时间是 25×0.1=2.5s。最后一行程序，凸轮完成最后一步时计数器位 C10 将 Q0 置位。凸轮重置输入可将 C10 复位。


DRUM
DRUM
C10
步设定值
K1
0.01秒/计数
K10
输出状态:
全
全
步
计数
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 K25
2 K20
3 K1500
4 K45
5 K180
6 K923
7 K1200
8 K864
9 K1200
10 K4000
11
12
13
14
15
16
输出
1: M0
2: M1
3: M2
4: M3
5: M4
6: M5
7: M7
8: M7
9: M10
10: M11
11: M12
12: M13
13:
14:
15:
16:
OK
取消
帮助(H)

23.2 时间/事件驱动型凸轮控制指令（EDRUM）

EDRUM 指令是基于时间和事件进行步转移的指令。

EDRUM 指令有 16 步和 16 个输出，基于时间和/或事件进行步转移，外部事件输入发生 OFF→ON 的转换时也会发生步转移。步转移的时间在每步的计数值中设定，事件为开关型触点。不使用的步和事件必须空着。开关量输出点可单独设定。



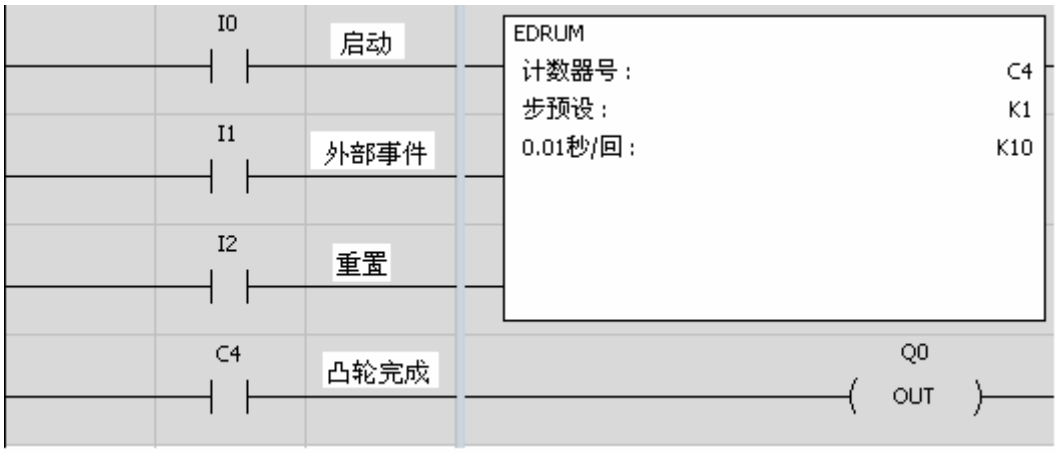
凸轮参数	数据类型	范围
计数器编号	C	0-374
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
事件	I,Q,M,S,T,C,SP	所有附录 1
开关量输出	I,Q,M	

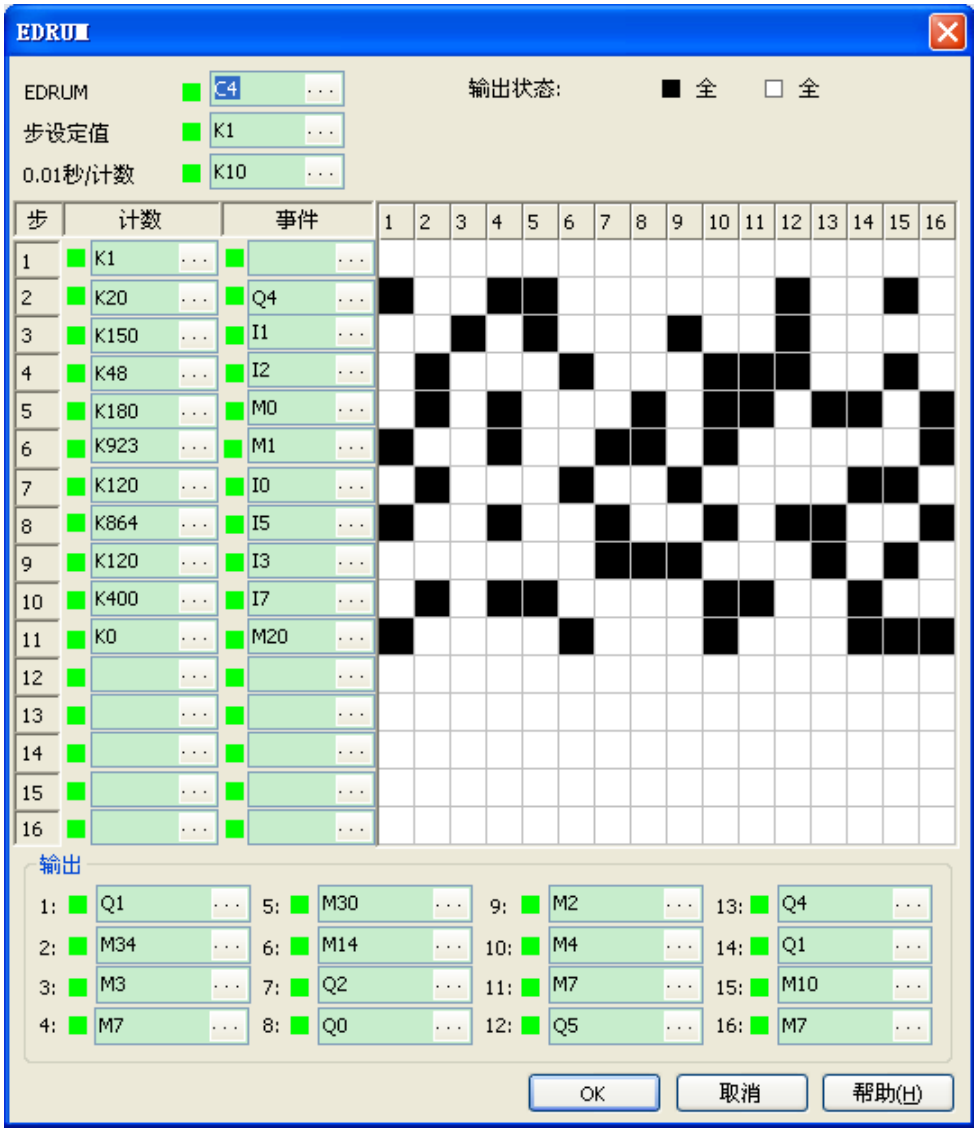
当启动输入为 ON 时，凸轮定时器被启用。只要当前步的事件为 ON，则当前步的定时器开始计时。当步计数值等于步的设定计数值时，凸轮转移到下一步。当最后一步完成或重置输入为 ON 时，此过程停止。CPU 在编程→运行的模式转换后或是重置输入为 ON 时，凸轮进入预设定的步。

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向 CA(n+2)中写入预设定的步。其他三个计数器仅做监控用。

计数器编号	n 的范围	功能	计数器位功能
C(n)	0-374	每步的计数	C(n)=凸轮完成
C(n+1)	1-375	定时器值	C(n+1)未使用
C(n+2)	2-376	预设定的步	C(n+2)未使用
C(n+3)	3-377	当前步	C(n+3)未使用

下面的程序例中，使用了步 1-11 和全部 16 个输出。预设定的步是步 1。时基是 K10×0.01=0.1s。因此，步 1 的停留时间是 1×0.1=0.1s。注意步 1 仅基于时间进行步转移（事件是空着的），并且步 1 的所有输出均为 OFF，这是一个典型的上电时的状态。最后一行程序凸轮完成最后一步时完成位 C4 将 Q0 置位。凸轮重置输入可将 C4 复位。





注意：如果一个事件驱动型凸轮（所有步的计数值均为 0）的所有事件均为 ON，则 PLC 每个扫描周期完成一步；因此，凸轮完成一个循环需要 16 个扫描周期。但是，由于 CPU 处于运行模式时凸轮的输出一直是启用的，因此凸轮的输出将相当于每个扫描周期的脉冲输出。

23.3 带掩码事件驱动型凸轮控制指令（MDRMD）（离散点输出）

MDRMD 指令不但具有基本事件型凸轮的所有功能，还增加了每步的最终输出控制功能。它根据前面讲的凸轮运行基本规则运行。

MDRMD 指令有 16 步和 16 个输出，每步的凸轮输出同一个输出屏蔽字位逻辑与。输出屏蔽字区域指定了 16 个输出屏蔽字的首个屏蔽字地址。基于时间和/或事件进行步转移，外部事件输入发生 OFF→ON 的转换时也会发生步转移。步转移的时间在每步的计数值中设定，事件为开关型触点。不使用的步和事件必须空着（缺省状态）。当启动输入为 ON 时，凸轮定时器被启用。只要当前步的事件为 ON，则当前步的定时器开始计时。当步计数值等于步的设定计数值时，凸轮转移到下一步。当最后一步完成或重置输入为 ON 时，此过程停止。CPU 在编程→运行的模式转换后或是重置输入为 ON 时，凸轮进入预设定的步。

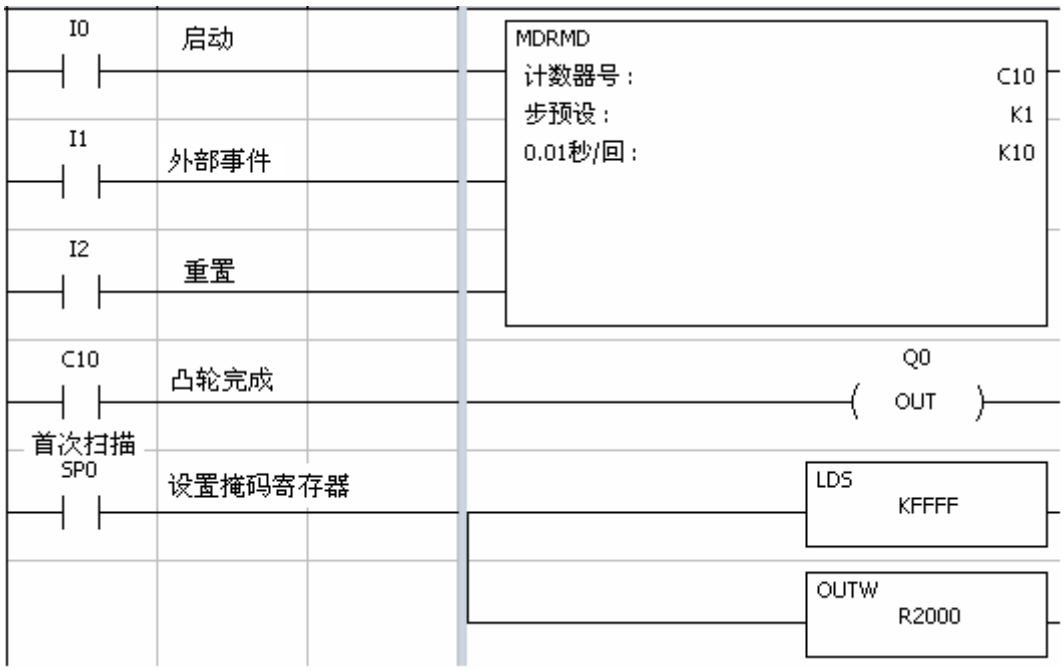
凸轮参数	数据类型	范围
计数器编号	C	0-374
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
事件	I,Q,M,S,T,C,SP	所有附录 1
开关量输出	I,Q,M	
输出屏蔽	R	

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向CA(n+2)中写入预设定的步。其他三个计数器仅做监控用。

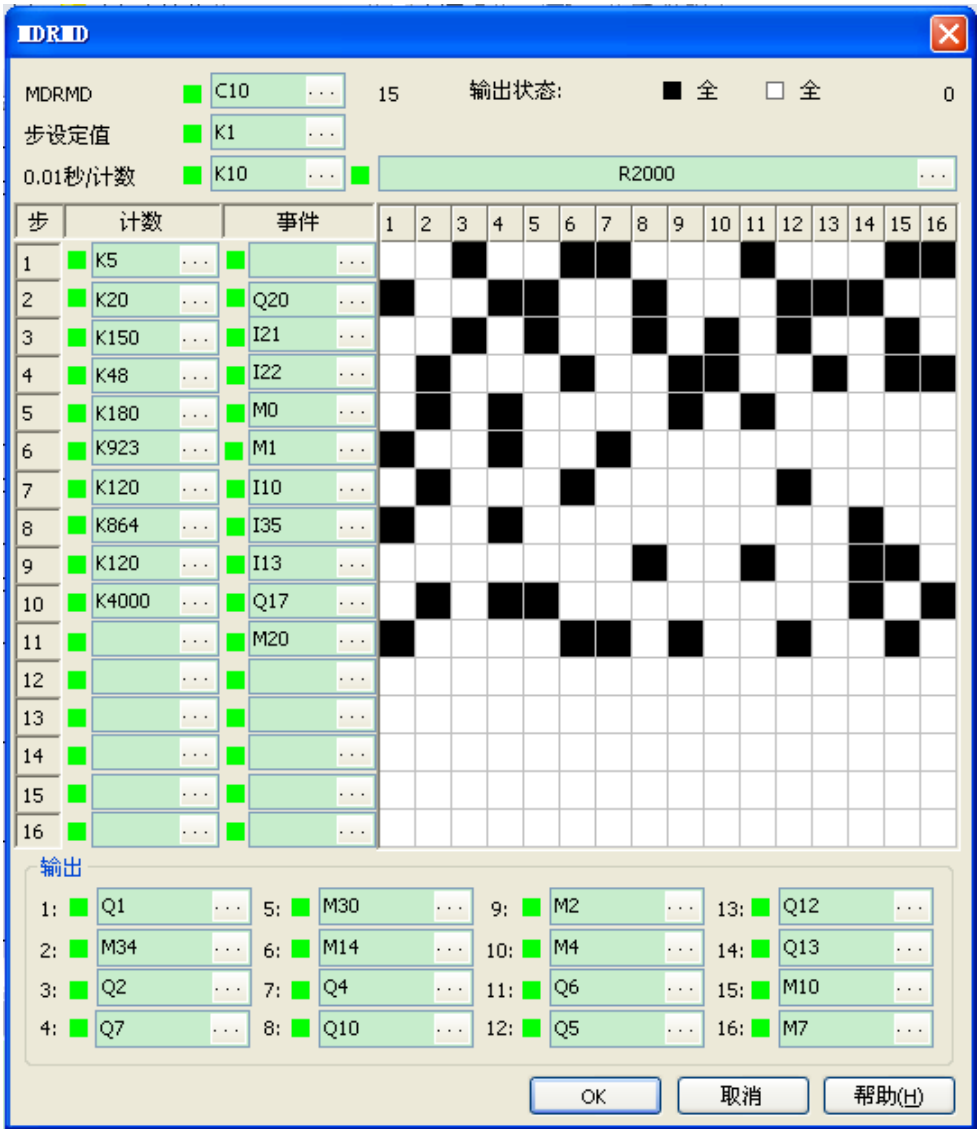
计数器编号	n 的范围	功能	计数器位功能
C(n)	0-374	每步的计数	C(n)=凸轮完成
C(n+1)	1-375	定时器值	C(n+1)未使用
C(n+2)	2-376	预设定的步	C(n+2)未使用
C(n+3)	3-377	当前步	C(n+3)未使用

下面的程序例中，使用了步 1-11 和全部 16 个输出。输出屏蔽字从 R2000 开始。凸轮的最终输出是输出表中输出同屏蔽字逻辑与后的结果。如果需要每次上电时凸轮所有输出为OFF，可在第一个扫描周期向 R2000 中写入 0。程序可随时更新屏蔽字来启用或禁用凸轮的输出。

预设定的步是步 1。时基是 K10×0.01=0.1s。因此，步 1 的停留时间是 5×0.1=0.5s。注意步 1 仅基于时间进行步转移（事件是空着的）。最后一行程序凸轮完成最后一步时完成位 C10 将 Q0 置位。凸轮重置输入可将 C10 复位。

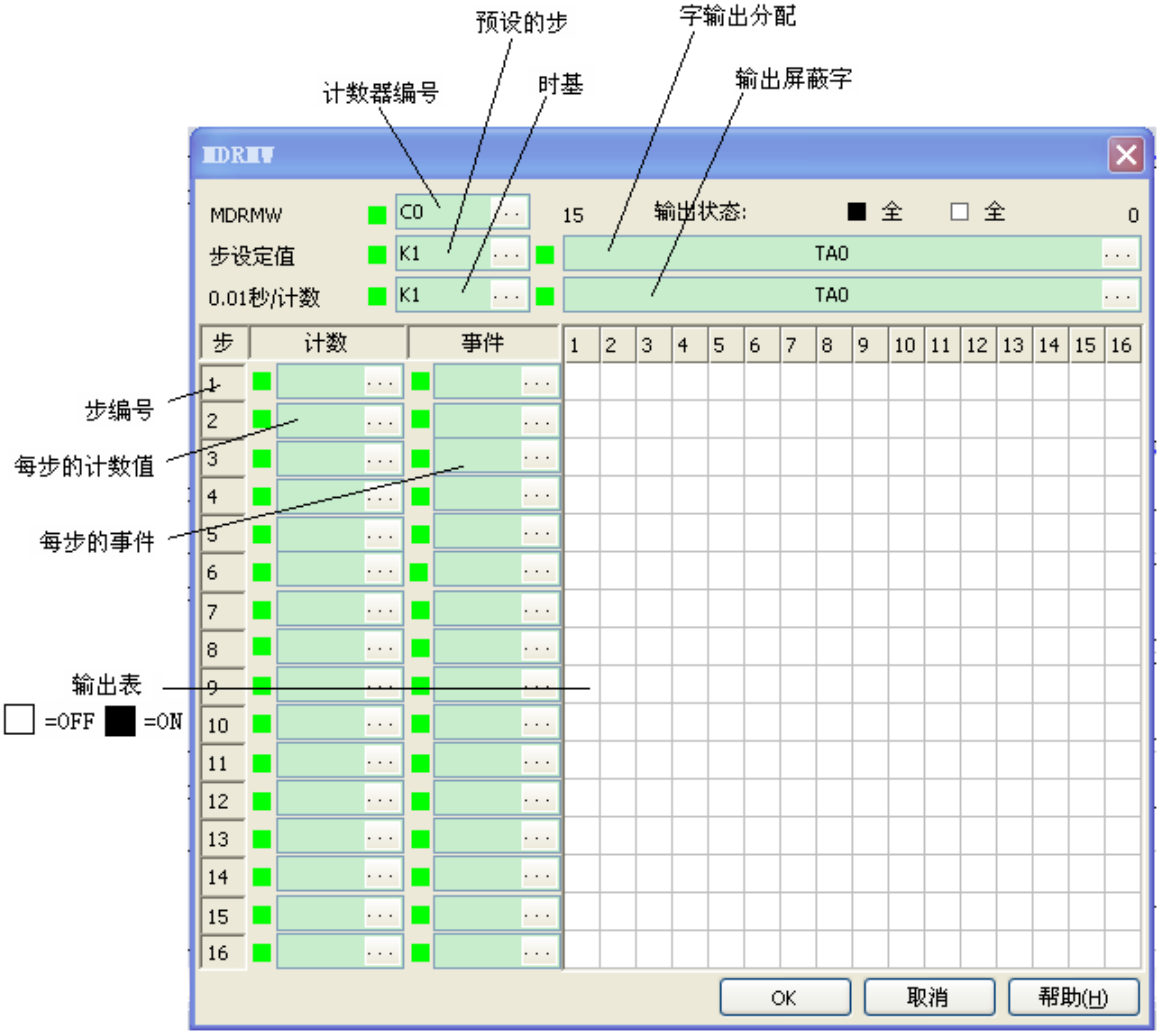


注意：上面的程序例中需要向 R2000-R2012 的 11 个掩码寄存器中写入常数，因为凸轮使用了 11 步。



23.4 带掩码事件驱动型凸轮控制指令（MDRMW）（字输出）

MDRMW 指令与 MDRMD 相比，输出的是一个字，而不是离散点。它根据前面讲的凸轮运行基本规则运行。



MDRMW 指令有 16 步和 16 个输出，每步的凸轮输出同一个输出屏蔽字位逻辑与。输出屏蔽字区域指定了 16 个输出屏蔽字的首个屏蔽字地址。最终的输出在“字输出分配”区域中指定。基于时间和/或事件进行步转移，外部事件输入发生 OFF→ON 的转换时也会发生步转移。步转移的时间在每步的计数值中设定，事件为开关型触点。不使用的步和事件可以空着（缺省状态）。当启动输入为 ON 时，凸轮定时器被启用。只要当前步的事件为 ON，则当前步的定时器开始计时。当步计数值等于步的设定计数值时，凸轮转移到下一步。当最后一步完成或重置输入为 ON 时，此过程停止。CPU 在编程→运行的模式转换后或是重置输入为 ON 时，凸轮进入预设定的步。

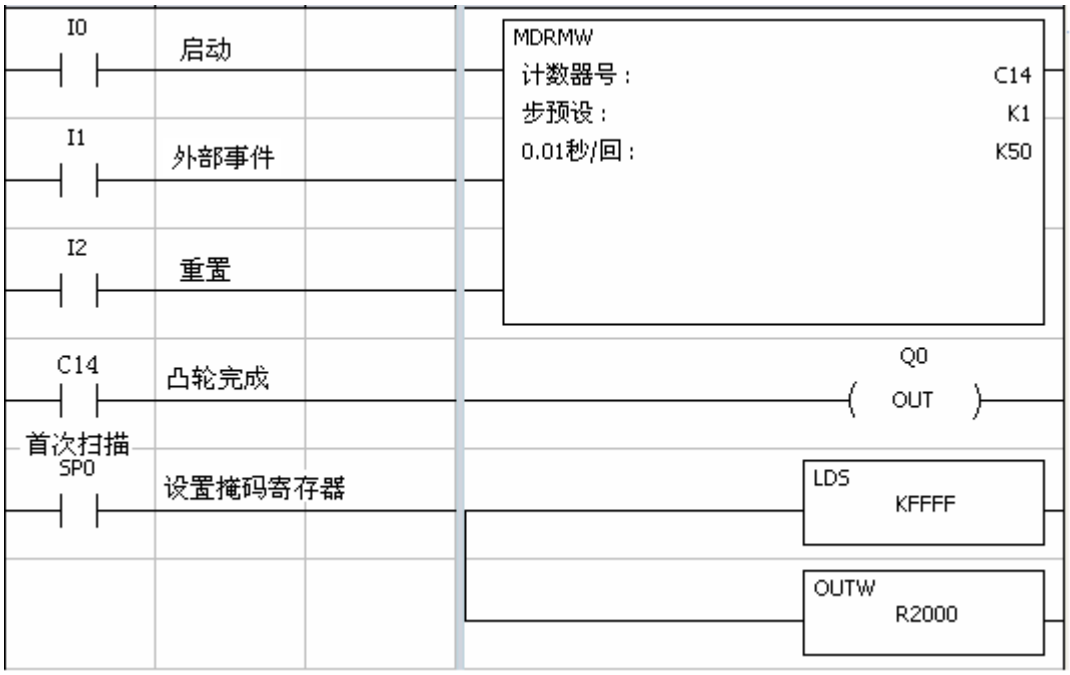
凸轮参数	数据类型	范围
计数器编号	C	0-374
预设定的步	K	1-16
时基	K	0-99.99s
每步的计数值	K	0-9999
事件	I,Q,M,S,T,C,SP	所有（附录 1）
字输出	R	
输出屏蔽	R	

凸轮指令使用四个计数器。可读取计数器的值来监控凸轮的状态。在程序中可随时向 CA(n+2)中写入预设定的步。其他三个计数器仅做监控用。

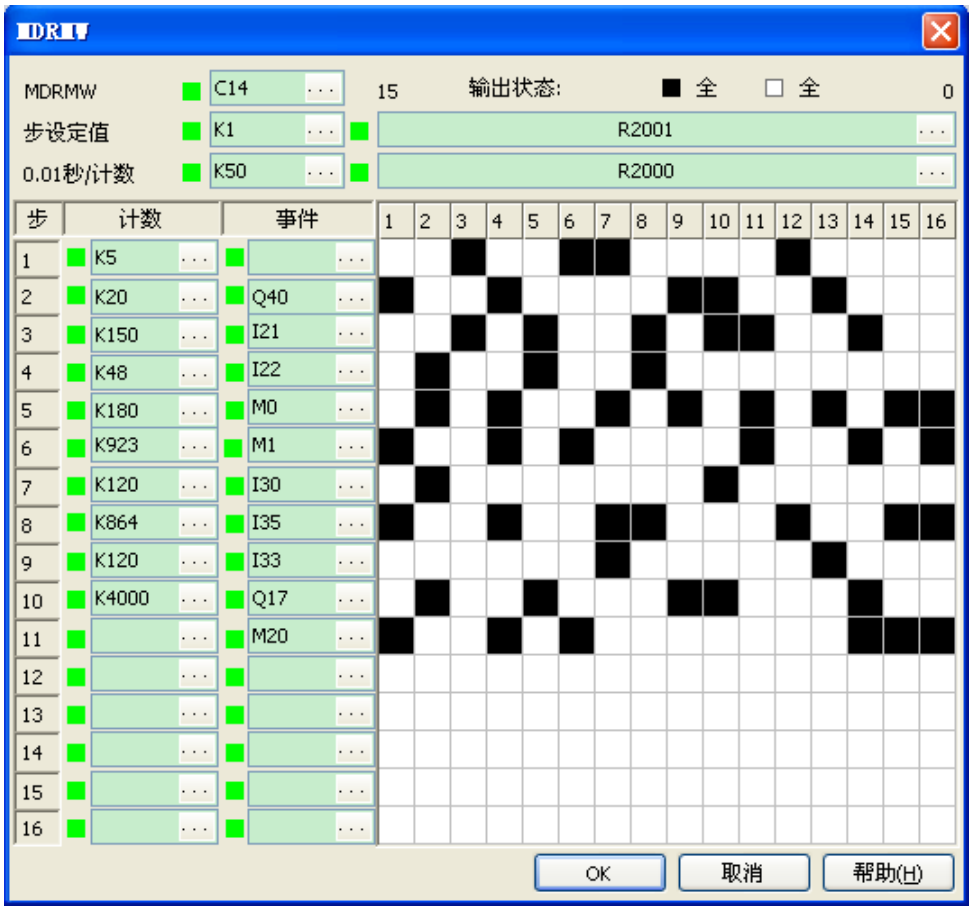
计数器编号	n 的范围	功能	计数器位功能
C(n)	0-374	每步的计数	C(n)=凸轮完成
C(n+1)	1-375	定时器值	C(n+1)未使用
C(n+2)	2-376	预设定的步	C(n+2)未使用
C(n+3)	3-377	当前步	C(n+3)未使用

下面的程序例中，使用了步 1-11 和全部 16 个输出。输出屏蔽字从 R2000 开始。凸轮的最终输出在 R2001 中。输出表中输出同 R2000 中的屏蔽字逻辑与，生成的结果存放在 R2001 中。如果需要每次上电时凸轮所有输出为 OFF，可在第一个扫描周期向 R2000 中写入 0。程序可随时更新屏蔽字来启用或禁用凸轮的输出。

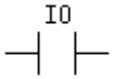
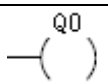
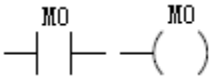
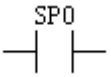
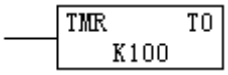
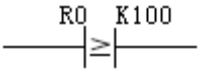
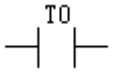
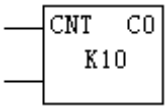
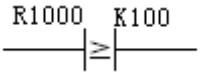
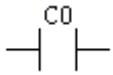
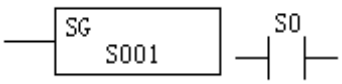
预设定的步是步 1。时基是 K50×0.01=0.5s。因此，步 1 的停留时间是 5×0.5=2.5s。注意步 1 仅基于时间进行步转移（事件是空着的）。最后一行程序凸轮完成最后一步时完成位 C14 将 Q0 置位。凸轮重置输入可将 C14 复位。



注意：上面的程序例中需要向 R2000-R2012 的 11 个掩码寄存器中写入常数，因为凸轮使用了 11 步。



附录 1 D4-454 功能存储器表

功能存储器类型	功能号（八进制）	寄存器地址（八进制）	数量（十进制）	图标
输入	I0-I1777	R40400-R40477	1024	
输出	Q0-Q1777	R40500-R40577	1024	
内部线圈	M0-M3777	R40600-R40777	2048	
特殊线圈	SP0-SP777	R41200-R41237	512	
定时器	T0-T377	R41100-R41117	256	
定时器经过值	无	R0-R377	256W	
定时器状态位	T0-T377	R41100-R41117	256	
计数器	C0-C377	R41140-R41157	256	
计数器经过值	无	R1000-R1377	256W	
计数器状态位	C0-C377	R41140-R41157	256	
用户寄存器	无	R1400-R7377	3072W	没有特定的图标，同许多指令一起使用
		R10000-R36777	11776W	
系统参数寄存器	无	R700-R777	64W	系统定义好的特殊用途寄存器
		R7400-R7777	256W	
		R37000-R37777	512W	
级	S0-S1777	R41000-R41077	1024	
通讯 I/O	GI0-GI3777	R40000-R40177	2048	Link 模块不使用的时候，可以用作内部线圈用
	GQ0-GQ3777	R40200-R40377	2048	

注：每段寄存器地址的最后一个寄存器不能作为 32 位指令的操作数。

附录 2 ASCII 转换表

DEC→HEX→ASCII 转换											
DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII
0	00	NUL	32	20	空格	64	40	@	96	60	
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	“	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

光洋电子(无锡)有限公司
Koyo ELECTRONICS （WUXI） CO., LTD.
地址：江苏省无锡市滨湖区建筑西路599号1栋21层
邮编：214072
电话：0510-85167888 传真：0510-85161393
http：//www.koyoele.com.cn

KEW-M4512
2018年12月