MOTEC 智能驱动器可编程控制器 编程手册

Version 2.1

MOTEC(中国)营业体系

2017-12-07

版本说明

版本号：

2017 年 12 月 07 日发行，第二版，Version2.1。

使用本产品时务必遵照使用说明，以免造成设备或人身伤害。

最新版本的使用说明书可在 www.motec365.com 下载。

联系方式:

MOTEC（中国）营业体系

北京诺信泰伺服科技有限公司

地址：北京市通州区环科中路17号11B（联东U谷西区）

电话：010-56298855-666

传真：010-65546721

邮编：100027

网址：http://www.motec365.com

eMail：motecSupport@sina.com

目 录

前言

使用须知

本手册是编写 MOTEC 智能驱动器内置可编程控制器用户程序的使用说明。

相关文档

有关可编程控制器用户程序编写软件 Motec Painter 的使用说明，请参照《MotecPLC 软件操作说明书》。

有关 MOTEC 智能驱动器电气特性和使用说明,请根据具体驱动器型号参照各自的使用手册进行查看。

有关 MOTEC 智能驱动器使用 MODBUS 总线协议通信说明请参照《MOTEC 智能驱动器 modbus 操作手册》。

有关 MOTEC 智能驱动器使用 CANopen 总线协议通信说明请参照《MOTEC 智能驱动器 CANopen 操作手册》

本手册说明

本手册收录了 MOTEC 智能驱动器内置可编程控制器可以使用的指令,并对编程时注意的事项进行说明，请在充分理解的基础上正确的加以使用。

本手册默认数字除了以“0x，0X”（16 进制）为开头或者以“h”（16 进制），“O”（8 进制），“B”（二进制）为后缀除外，其余都是以 10 进制数据表示。

注意事项

对于本手册中的内容，如果有疑问或者发现错误之处，请与 MOTEC 联系。

由于可编程控制器内置在驱动器内部，可以使用户完成更加灵活的运动控制和逻辑控制，算术运算等功能，更可以将复杂的程序简化成精简的指令方便客户使用，如果您在使用过程中需要定制指令，请联系 MOTEC 的销售或者技术支持人员，我们将会竭诚为您服务。

一、基本指令，高级指令，运动控制指令目录索引

时序控制基本指令

定时器计数器操作指令

程序控制指令

数据操作指令

16 位数据比较操作

32 位数据比较操作

数据赋值转移操作

数据移位指令

16 位整型数据算术运算指令

32 位整型数据算术运算指令

二、使用前须知

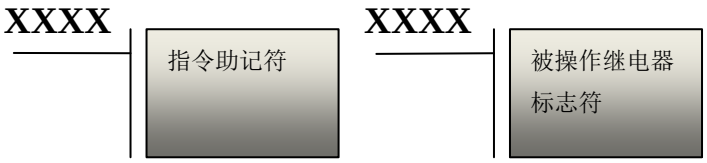
指令结构

MOTEC 智能驱动器内置可编程控制器可以通过梯形图编写程序，并且生成助记符，本内置可编程控制器拥有多种指令，可以循环或者组合一起完成复杂的功能。下面将简要介绍 MOTEC 智能驱动器内置可编程控制器的基本指令和高级指令的指令格式。更多的指令运用方式请参照第二章和第三章的指令说明。

基本指令

基本指令以继电器的时序回路为基本单位，回路中只有 OFF 或者 ON 两种状态，包括进行逻辑与或非等基本指令和沿操作指令，但是具体的程序会根据选择的继电器种类有关。

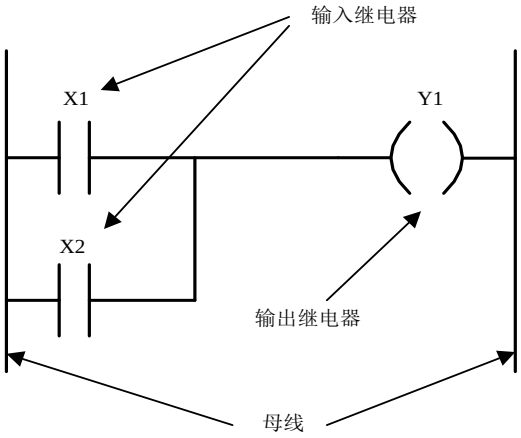
基本指令的操作参数一般只有一个，基本指令的助记符语句形式为



根据基本指令的不同，操作的参数的数目会有所不同，具体详细参数请见每个指令的具体解释。

例如下面是一个电平操作的示例

梯形图 助记符




```
LD    X1
OR    X2
OUT   Y1
```

指令解释（助记符部分）

指令内容	指令说明
LD X1	装载输入继电器 X1 的状态
OR X2	输入继电器 X1 的状态与输入继电器 X2 的状态相或
OUT Y1	将逻辑运算值输出到输出继电器 Y1

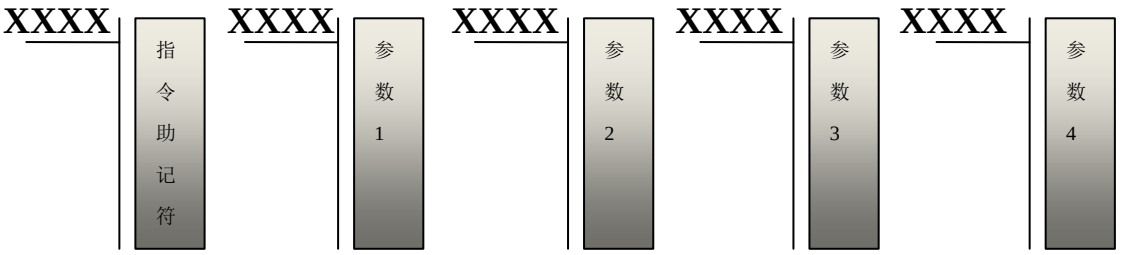
程序功能说明：本程序功能是 X1 和 X2 任意一个为 ON 状态的时候，Y1 为 ON，只有 X1 和 X2 全部为 0FF 状态的时候，Y1 是 OFF 状态，这是一个典型的或操作。

本例是基本指令中时序控制基本指令的应用，另外基本指令还包括栈操作指令，沿操作指令，定时器操作指令，计数器操作指令，程序控制指令几种指令，本手册会在第二章基本指令中，详细介绍每个指令的使用说明并举例。

高级指令

相对于基本指令对于继电器的操作，高级指令的控制的操作单位是寄存器和辅助继电器。可以进行 16 位数据或者更复杂数据格式的操作，根据不同的指令格式，高级指令在使用过程中需要基本指令的配合，虽然在含有高级指令的程序回路中同样只有 ON 或者 OFF 两种状态，但是高级指令本身可以对数据进行逻辑运算或者算术运算，完成复杂的操作。

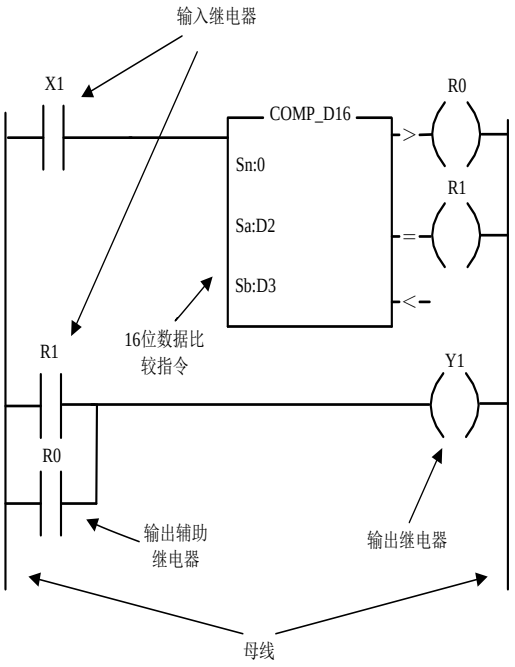
高级指令拥有比基本指令更多的操作参数，根据不同的指令和不同的操作参数的数目有所不同。高级指令的助记符语句形式为



根据高级指令的不同，参数的个数也不同，参数代表的内容也不相同，具体详见每个指令的参数解释。

例如下面是一个使用高级指令进行操作的程序。

梯形图



助记符

LD	X1
DATA_16_COMPARE	FUNC
	1
	0
	D2
	D3
	R0
	R1
LD	R1
OR	R0
OUT	Y1

指令解释（助记符部分）

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
DATA_16_COMPARE	FUNC	输入继电器 X1 有效时，将 D2 与 D3 的值相比
	1	较输出给 R0,R1。当 D2>D3 时，R0 置 ON;当
	0	D2=D3 时，R1 置 ON.
	D2	
	D3	
	R0	
	R1	
LD	R1	装载输入继电器 R1 的状态
OR	R0	将 R1 的状态和辅助继电器 R0 相或
OUT	Y1	将上一步的逻辑值输出到输出继电器 Y1

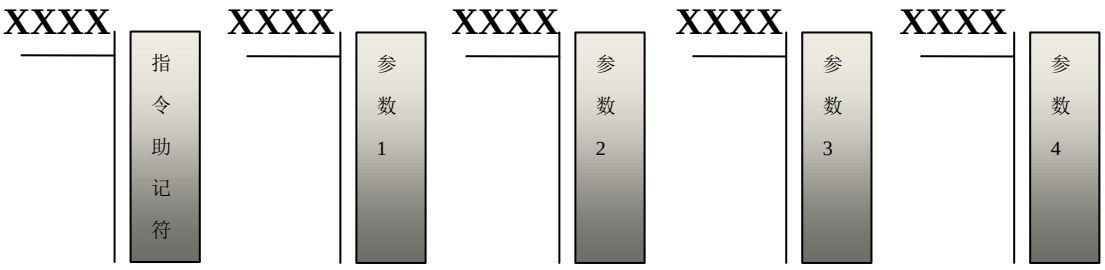
程序说明：本程序使用到了高级指令里面的 16 位数据比较指令。DATA_16_COMPARE 有三个参数，分别是比较双方的寄存器地址或常数和输出辅助继电器，还可进行有符号类型和无符号类型选择。本程序完成的功能是输入继电器 X1 有效时，将 D2 与 D3 的值相比较输出给 R0，R1。然后再将 R1 与 R0 相或的结果输出 Y1。即当 X1 为 ON 时，D2>D3 输出的 R0 的值为 ON 或者 D2=D3 输出的 R0 的值为 ON，Y1 就为 ON。只有当 R1 与 R0 为 OFF 的状态时，Y1 为 OFF。助记符中 16 位数据大于等于比较指令中的“1”表示比较的类型。

本程序即使用了基本指令，也使用了高级指令，使用这两种指令配合，MOTEC 智能驱动器内置可编程控制器可以完成复杂的继电器或者寄存器的逻辑运算或者算术运算。

运动控制指令

对比通用的电机驱动器和可编程控制器，MOTEC 智能驱动器内置可编程控制器是两者的集合和统一，完全可以独立完成复杂的逻辑运算和算术运算，更是与电机驱动器结合在一起，完成复杂的运动控制操作。使运动控制更加方便，节省成本，减少线路，使控制更加集成。

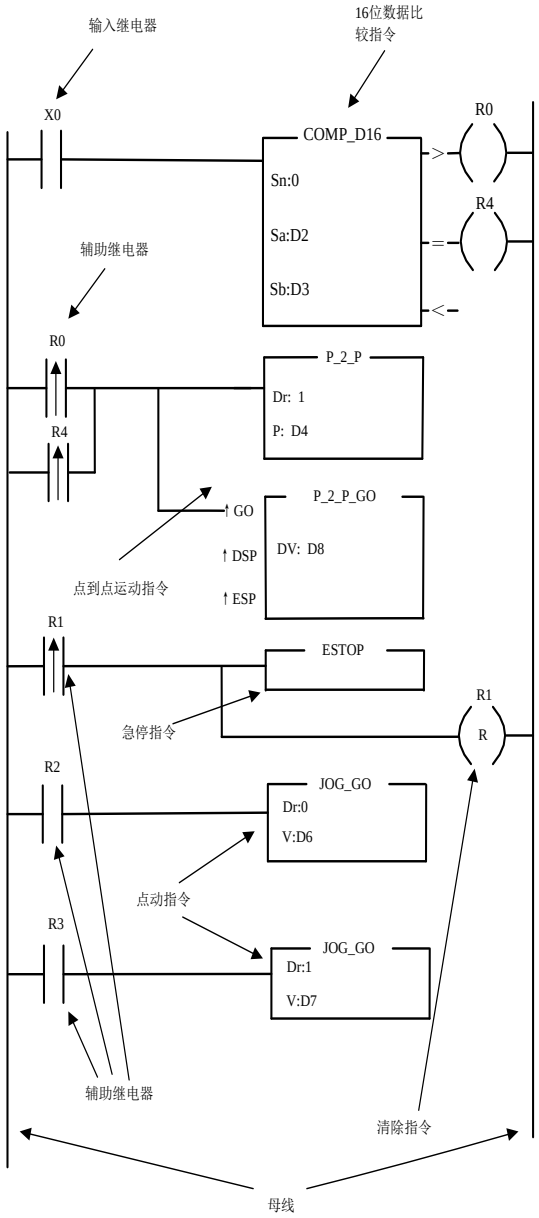
运动控制指令与高级指令类似，拥有不止一个的操作参数，每个参数的数据长度也不一定相同，运动控制指令的助记符语句形式为



根据运动控制指令的不同，参数的个数也不同，参数代表的内容也不相同，具体详见第四章每个指令的参数解释。

例如下面是一个综合基本指令，高级指令和运动控制指令的程序

梯形图 助记符




```
LD                X0
DATA_16_COMPARE   FUNC
                  1
                  0
                  D2
                  D3
                  R0
                  R4
LD_R              R0
OR_R              R4
P_2_P             FUNC
                  1
                  1
                  D4
MPS
P_2_P_GO          FUNC
                  1
                  D8
FIN20             0
FIN30             0
LD_R              R1
ESTOP             FUNC
RESET             R1
LD                R2
JOG_GO            FUNC
                  0
                  1
                  D6
LD                R3
JOG_GO            FUNC
                  1
                  1
                  D7
```

指令解释（助记符部分）

指令内容		指令说明
LD	X0	装载输入继电器 X0 的状态
DATA_16_COMPARE	FUNC	寄存器 D2 与寄存器 D3 做大于等于比较，如
	1	果 D2>D3 输出辅助继电器 R0 为 ON。如果
	0	D2=D3 输出辅助继电器 R4 为 ON。
	D2	
	D3	
	R0	
	R4	
LD_R	R0	装载是否发生输入继电器 R0 的上升沿动作
OR_R	R4	装载是否发生输入继电器 R4 的上升沿动作
P_2_P	FUNC	点到点设置内容，相对运动，运动距离取自寄
	1	存器 D4 和 D5 的值，D5 为 32 位运动距离的
	1	高 16 位，D4 为低 16 位。
	D4	
MPS		
P_2_P_GO	FUNC	点到点位置控制运动启动，寄存器 D8 为减速
	1	停止数据。
	D8	
FIN20	0	没有减速和急停的的继电器。
FIN30	0	
LD_R	R1	装载是否发生输入继电器 R1 的上升沿动作
ESTOP	FUNC	急停电机当前运动
RESET	R1	清除继电器 R1
LD	R2	装载输入继电器 R2 的状态
JOG_GO	FUNC	启动点动运动，正向点动，点动速度取自寄存
	0	器 D6
	1	
	D6	
LD	R3	装载输入继电器 R3 的状态
JOG_GO	FUNC	启动点动运动，负向点动，点动速度取自寄存
	1	器 D7
	1	
	D7	

程序功能说明：当继电器 X0 有效时，可编程控制器内部寄存器 D2 里面的值大于 D3 时，就会进行点到点的设定指令，相对运动，运动距离来自寄存器 D4 和 D5，直接启动电机。这里未设置点到点指令减速和急停继电器。直接检测到 R1 的上升沿，电机立刻停止运动。

系统检测到 R2 为 ON，启动正向点动，点动的转速取自寄存器 D6。

系统检测到 R3 为 ON，启动负向点动，点动的转速取自寄存器 D7。

综合使用了基本指令，高级指令和运动控制指令，可以完成复杂的操作。MOTEC 智能驱动器拥有丰富的外部资源和内部可使用的资源，可以依靠多种指令来使用开关量，16 位

数据，32 位数据，二进制浮点数，10 进制浮点数，完成各种逻辑运算，大小判断，加减乘除等算术运算，数据转移，数据格式的转换，移位等操作。可以完成驱动器位置控制，速度控制，模拟量控制，回零控制，点动控制等多种运动控制功能。

三、编程工具

硬件环境：Windows 操作系统的电脑。
编程线缆：RS-232 或者 RS-485。
电源：根据不同驱动器的内部型号确定。
驱动器：支持内置可编程控制功能的 MOTEC 智能驱动器。

四、编程软件使用方法

详见《MotecPLC 软件操作说明书》。

注意：MOTEC 智能驱动器操作模式有多种，任何一种操作模式下都可以支持内部 PLC 编程。

第 1 章 继电器、特殊继电器、普通寄存器、计数器、定时器、内部寄存器说明

本章将就 MOTEC 智能驱动器内置的可编程控制器的继电器、特殊继电器、普通寄存器、计数器、定时器、内部寄存器进行说明，包括有关硬件接口和各种辅助元件，这些项目是可编程控制器的应用基础。

用户可以访问任何一个继电器，所有的地址全部都是 16 位长度，为了读取方便，本手册在表征驱动器各种资源的地址时，都是用 16 进制的数据方式，例如 0x1001，0x9005 等。

各类元件的编号一览如表 1-1 所示

表 1-1 各类元件的编号一览表

意义	符号	数据格式	标识符
输入继电器	X	1 位	X0-X31
输出继电器	Y	1 位	Y0-Y31
辅助 R 继电器	R	1 位	R0-R2047
驱动器状态继电器	S	1 位	S0-S255
定时器状态继电器	T	1 位	T0-T31
计数器状态继电器	C	1 位	C0-C31
定时器装载值寄存器	TLD	16 位	T0-T31
计数器装载值寄存器	CLD	16 位	C0-C31
16 位辅助寄存器	D	16 位	D0-D512
驱动内部参数	P	16 位	P0-P300
驱动器内部 CANopen 使用通信地址	CANopen	32 位	CANopen 专用

1.1 输入继电器、输出继电器、辅助继电器、状态继电器、特殊继电器说明

MOTEC 智能驱动器内置可编程控制器拥有

— 2016 个普通辅助寄存器，使用次数没有限制，并且可以使用指令保存，上电恢复
— 32 个特殊辅助继电器，用来表示可编程控制器的运行状态
— 256 个运动状态继电器，用来表示电机当前的运行状态
— 32 个定时器状态继电器，表示定时器的状态
— 32 个计数器状态继电器，表示计数器的状态
— 170K 字节存储空间或者更多（根据不同的型号驱动器有相应的区别），
— 可用 RS-232、RS-485、CAN 总线通讯

1.1.1 可编程控制器各类继电器的存储方式说明

MOTEC 智能驱动器内置可编程控制器拥有的多种继电器类型，都是采用二进制存储在驱动器内部。用户可以每一个继电器的标识符和实际地址来访问任何一个继电器。

使用访问开关量的 1 位数据格式的地址，例如可以访问地址 0x1000，来读取输入继电器 X0 的状态，可以访问地址 0x3006 来读取辅助继电器 R6 的状态。

1.1.2 输入继电器

MOTEC 智能驱动器内部集成了多路数字量输入口。输入继电器可以参与程序内部控制，并且输入继电器的值只取决于输入口的状态的 ON 或者 OFF，不能作为输出口使用，下文 X 继电器即代指输入继电器，表 1-2 为 MOTEC 智能驱动器的输入口相关信息。

表 1-2 内部输入口的相关信息

驱动器分类	驱动器型号	输入口个数	数据格式	使用映射地址	程序和通信访问地址
智能步进驱动器	SD253B	INPUT1~ INPUT3	1 位	X0~ X2	0x1000~0x1002
	SD266B	INPUT1~ INPUT8	1 位	X0~ X7	0x1000~0x1007
	SD388B	INPUT1~ INPUT8	1 位	X0~ X7	0x1000~0x1007
	SD3228B	INPUT1~ INPUT8	1 位	X0~ X7	0x1000~0x1007
交流伺服驱动器	β 交流伺服标准版	INPUT1~ INPUT6	1 位	X0~ X5	0x1000~0x1005
	β 交流伺服全功能版	INPUT1~ INPUT12	1 位	X0~ X11	0x1000~0x100B
直流伺服驱动器	ARES	INPUT1~ INPUT8	1 位	X0~ X7	0x1000~0x1007
	DGFLY	INPUT1~ INPUT4	1 位	X0~ X3	0x1000~0x1003
	BEE	INPUT1~ INPUT3	1 位	X0~ X2	0x1000~0x1002
	HBIRD	INPUT1~ INPUT3	1 位	X0~ X2	0x1000~0x1002
	SWLOW	INPUT1~ INPUT6	1 位	X0~ X5	0x1000~0x1005
	EAGLE	INPUT1~ INPUT6	1 位	X0~ X5	0x1000~0x1005
	BEAR	INPUT1~ INPUT8	1 位	X0~ X7	0x1000~0x1007
	HIPPO	INPUT1~ INPUT8	1 位	X0~ X7	0x1000~0x1007
	ELPHT	INPUT1~ INPUT8	1 位	X0~ X7	0x1000~0x1007

输入继电器是 MOTEC 智能驱动器内置可编程控制器从外部接收信号的接口，不能使用程序来控制这些输入继电器。

可编程控制器在程序每个扫描周期开始时，会自动装载当前的输入继电器的开关状态，读入到输入继电器的实际使用地址中，在程序执行过程中，即使输入发生变化，输入继电器的使用值也不会发生变化，而在下一个扫描周期开始时，读入该变化值。在读取的过程，可编程控制器是按照地址由低到高的顺序相继读取输入继电器的值。输入继电器在系统中的信号传输过程如下图 1.1。

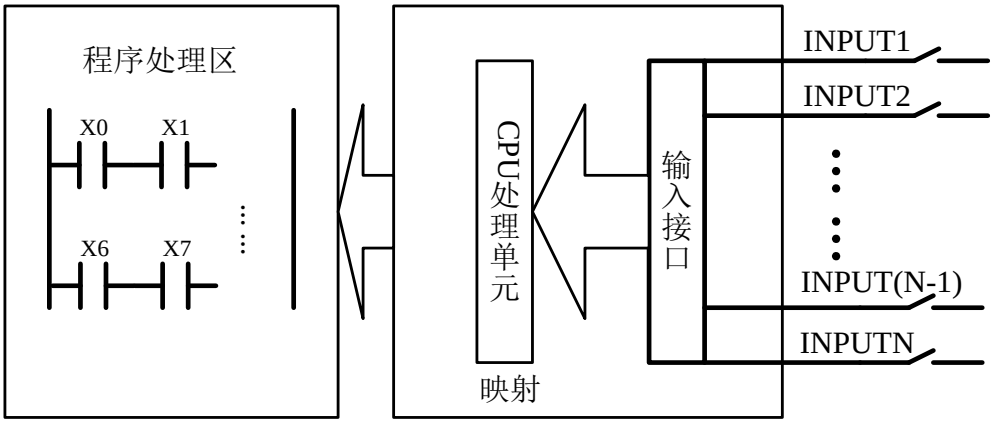


图 1.1 输入继电器信号在系统中的传输

使用限制：不能使用硬件上不存在的 X 继电器，作为输入口继电器，只能作为输入口使用，不能置位或者输出口使用，否则会引起错误报警或者程序不能正常执行。

1.1.3 输出继电器

MOTEC 智能驱动器内部集成了数字量输出口，全部采用了光耦隔离。输出继电器可以参与程序内部控制，并且可以作为程序的输入条件，但是开关状态只与程序的输出有关，不受外部的 ON 或者 OFF 的影响，以下 Y 继电器即代指输出继电器。表 1-3 为 MOTEC 智能驱动器的输出口相关信息。

表 1-3 内部输出口相关信息

驱动器分类	驱动器	输出口个数	数据格式	使用映射地址	程序和通信访问地址
智能步进驱动器	SD253B	OUTPUT 1	1 位	Y0	0x2000
	SD266B	OUTPUT 1~ OUTPUT 3	1 位	Y0~ Y2	0x2000~0x2002
	SD388B	OUTPUT 1~ OUTPUT 3	1 位	Y0~ Y2	0x2000~0x2002
	SD3228B	OUTPUT 1~ OUTPUT 3	1 位	Y0~ Y2	0x2000~0x2002
交流伺服驱动器	β 交流伺服标准版	OUTPUT 1~ OUTPUT 2	1 位	Y0~ Y1	0x2000~0x2001
	β 交流伺服全功能版	OUTPUT 1~ OUTPUT 6	1 位	Y0~ Y5	0x2000~0x2005
直流伺服驱动器	ARES	OUTPUT 1~ OUTPUT 3	1 位	Y0~ Y2	0x2000~0x2002
	DGFLY	OUTPUT 1~ OUTPUT 2	1 位	Y0~ Y1	0x2000~0x2001
	BEE	OUTPUT 1~ OUTPUT 2	1 位	Y0~ Y1	0x2000~0x2001
	HBIRD	OUTPUT 1	1 位	Y0	0x2000
	SWLOW	OUTPUT 1~ OUTPUT 2	1 位	Y0~ Y1	0x2000~0x2001
	EAGLE	OUTPUT 1~ OUTPUT 2	1 位	Y0~ Y1	0x2000~0x2001
	BEAR	OUTPUT 1~ OUTPUT 3	1 位	Y0~ Y2	0x2000~0x2002
	HIPPO	OUTPUT 1~ OUTPUT 3	1 位	Y0~ Y2	0x2000~0x2002
	ELPHT	OUTPUT 1~ OUTPUT 3	1 位	Y0~ Y2	0x2000~0x2002

输出继电器是 MOTEC 智能驱动器内置可编程控制器向外部负载发送信号的接口，可以使用程序来控制输出继电器的状态。

可编程控制器在程序每个扫描周期开始时，会自动将当前输出继电器映射地址内的状态输出到输出继电器。在程序执行过程中，即使当前输出继电器映射的地址内状态发生改变，实际输出继电器的值也不会发生变化，而是在下个扫描周期开始时，更新该变化。

在输出的过程，可编程控制器是按照地址由低到高的顺序改变输出继电器的值，输出继电器在系统中的信号传输过程如下图 1.2。

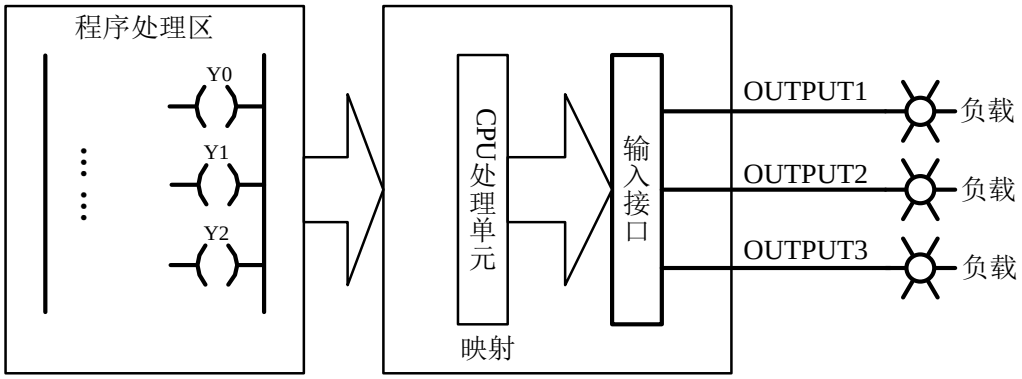


图 1.2 输出继电器信号在系统中的传输

使用限制：Y 继电器作为输出口继电器，只能作为输出口使用，不能作为输入口使用，但是用户程序内部可以使用逻辑控制指令，否则会引起错误报警或者程序不能正常执行。

1.1.4 辅助继电器

辅助继电器仅用于程序内部控制使用，ON 或者 OFF 状态不会产生外部输出，辅助继电器的具体信息如下

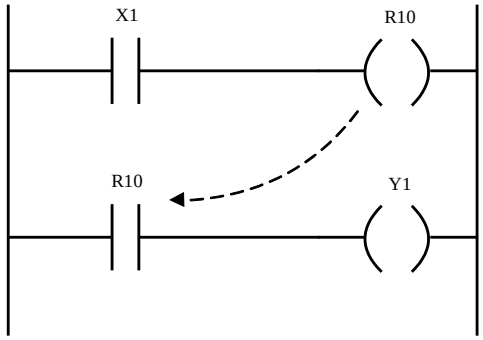
表 1-4 辅助继电器相关信息

项目	个数	数据格式	使用映射地址	程序和通信访问地址
一般辅助继电器	2016	1 位	R0～R2015	0x3000-0x37DF
特殊辅助继电器	32	1 位	R2016～R2047	0x37E0-0x37FF

辅助继电器使用 R 表示，以下 R 继电器即代指辅助继电器。

一般辅助继电器使用限制：一般 R 继电器作为内部辅助继电器，使用次数没有限制，但是如果需要掉电保存，就需要在掉电之前使用保存命令，将当前 R 继电器的值保存在驱动器内部。辅助继电器一般的功能是作为中间变量和标识量在程序中传输，可以被置位和输出程序结果。 例如

梯形图




```
LD    X1
OUT   R10
LD    R10
OUT   Y1
```

指令解释（助记符部分）

助记符

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
OUT	R10	输出当前栈值到 R10
LD	R10	装载辅助继电器 R10 的状态
OUT	Y1	输出当前栈值到 Y1

程序功能说明：本程序就是将 R10 作为一个中间标识，将输入继电器 X1 的状态传递到输出继电器 Y1。

注意：辅助继电器和输入输出继电器不同，辅助继电器的状态在程序执行过程中是会随着程序改变，而不是只有在每个扫描周期前才刷新状态。

特殊辅助继电器指在驱动器内部，被定义了特殊功能的继电器，里面的内容大部分不允许更改，并且里面的值会根据实际程序的运行状态变化。用户可以使用这些继电器完成一些特殊的功能。

特殊辅助继电器的代表的意义如表 1-5

表 1-5 特殊辅助继电器的意义

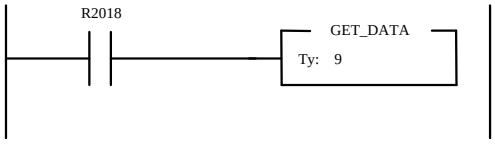
序号	意义	特殊继电器寄存器项目	位数	R/W
R2016	常开继电器	一直为 OFF	1	NO
R2017	常闭继电器	一直为 ON	1	NO
R2018	初始脉冲继电器 ON	刚开始上电为 ON 第二个周期为 OFF	1	NO
R2019	初始脉冲继电器 OFF	刚开始上电为 OFF 第二个周期为 ON	1	NO
R2020	扫描脉冲继电器	ON 和 OFF 周期交替变化，上电初始化为 ON	1	NO
R2021	扫描脉冲继电器	ON 和 OFF 周期交替变化，上电初始化为 OFF	1	NO
R2022	保留	保留	1	NO
R2023	时间脉冲继电器	1ms ON 和 OFF 交替输出高低脉冲	1	NO
R2024		10ms ON 和 OFF 交替输出高低脉冲	1	NO
R2025		100ms ON 和 OFF 交替输出高低脉冲	1	NO
R2026		1s ON 和 OFF 交替输出高低脉冲	1	NO
R2027		1 分钟 ON 和 OFF 交替输出高低脉冲	1	NO
R2028	串口通信超时寄存器	通信如果超时，寄存器置 1，通信正常后恢复	1	NO
R2029	下载程序继电器	程序下载的时候该继电器置 ON，程序在下载结束以后置 OFF	1	NO
R2030	程序执行继电器	1：程序运行；0：程序不运行。该继电器初始化为 1，由用户在需要的时候更改，并且是执行下一步的指令	1	YES
R2031～R2047	保留	作为版本扩展保留使用	1	NO

使用限制：用户可以使用这些继电器作为判断条件，但是将某些特殊继电器作为输出口来使用，会导致用户程序得不到正确的结果或者程序内部的错误，请用户根据自己的实际使用情况来确定。

使用特殊辅助继电器，用户可以实现一些特殊的功能

例如 1：使用特殊继电器 R2018 可以在上电第一个周期进行一些设置并且第二个周期开始不发生改变

梯形图



助记符

```
LD                  R2018
PLC_GET_DATA    FUNC
                    9
```

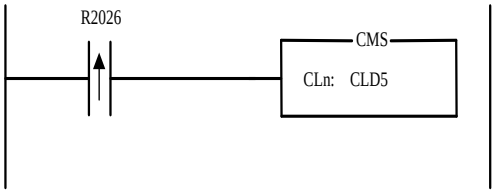
指令解释（助记符部分）

指令内容		指令说明
LD	R2018	装载特殊辅助继电器 R2018 的状态
PLC_GET_DATA	FUNC 9	读取保存在 FLASH 里面的 D 寄存器的值

程序说明：本程序的功能是在第一个扫描周期读取保存在 FLASH 里面的 D 寄存器的值，由于在第二个扫描周期开始前，R2018 的状态由 ON 变成 OFF 状态，由此以后不会再运行 PLC_GET_DATA 指令。

例如 2：使用特殊继电器 R2026 和计数器相配合可以完成长时间的计时功能

梯形图



助记符

LD_R	R2026
CMS	FUNC
	CLD5

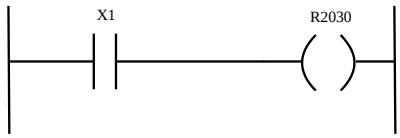
指令解释（助记符部分）

指令内容		指令说明
LD_R	R2026	装载特殊辅助寄存器 R2026 的上升沿
CMS	FUNC	CLD5 计数器计数
	CLD5	

程序说明：本程序是使用辅助寄存器完成长时间计时的一个方法，R2026 是一个 1s ON 和 OFF 交替输出高低脉冲的辅助继电器，如果检测一次 R2026 上升沿，即每两秒钟 C5 计数器减 1，如果将 CLD5 计数器装载寄存器内设置值为 500，既可以完成 500×2×1S，即 1000S 的计时功能。

例如 3：使用特殊继电器 R2030 可以在运行过程中终止程序的运行

梯形图



助记符

LD	X1
OUT	R2030

指令解释（助记符部分）

指令内容		指令说明
LD	X1	装载外部输入口 X1
OUT	R2030	输出到特殊继电器 R2030

程序说明：本程序是可以作为特殊情况下，停止当前 PLC 程序运行的功能，使用外部输入继电器 X1 置位特殊继电器 R2030 ，达到停止 PLC 程序运行的目的，并且当前程序的指针移至停留在 PLC 程序停止之前。

注意：如果使用特殊继电器 R2030 终止当前程序运行，需要用户手动参与启动运行，并且上电初始化的一些继电器将不会被初始化。

1.1.5 状态继电器

状态继电器是指驱动器内部运行状态的指示，以下 S 继电器代指状态继电器，状态继电器与特殊继电器作用类似，S 继电器是表征驱动器内部运动控制状态的继电器。状态继电器有 256 个，现在使用了 12 个，详细信息如表 1-6 所示

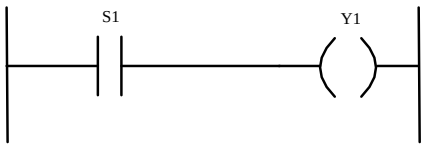
表 1-6 状态继电器表示值

项目	数据格式	程序和通信访问地址	代表意义	继电器状态	驱动器状态
S0	1 位	0x4000	电机当前使能状态	OFF	电机在释放状态
				ON	电机在使能状态
S1	1 位	0x4001	电机运动状态	OFF	当前电机速度为 0
				ON	当前电机速度不为 0
S2	1 位	0x4002	电机寻找零位开关状态	OFF	零位开关不导通
				ON	零位开关导通
S3	1 位	0x4003	回零的方式状态	OFF	寻找零位开关
				ON	寻找绝对零点位置
S4	1 位	0x4004	正硬限位开关状态	OFF	正硬限位未发生
				ON	正硬限位发生
S5	1 位	0x4005	负硬限位开关状态	OFF	负硬限位未发生
				ON	负硬限位发生
S6	1 位	0x4006	正软限位开关状态	OFF	正软限位未发生
				ON	正软限位发生
S7	1 位	0x4007	负软限位开关状态	OFF	负软限位未发生
				ON	负软限位发生
S8	1 位	0x4008	当前运动状态	OFF	运动完成
				ON	运动未完成
S9	1 位	0x4009	位置控制方式标志	OFF	相对模式
				ON	绝对模式
S10	1 位	0x400A	速度方向标志	OFF	正向
				ON	负向
S11	1 位	0x400B	回零方向标志	OFF	正向
				ON	负向

使用限制：S 继电器作为运动状态继电器，严禁用户作为输出继电器使用，只可以作为逻辑判断条件使用。

例如：使用状态继电器可以判断当前的电机是否在运动

梯形图



助记符

```
LD      S1
OUT     Y1
```

指令解释（助记符部分）

指令内容		指令说明
LD	S1	装载状态继电器 S1 的状态
OUT	Y1	输出当前栈值到输出继电器 Y1

程序说明：本程序可以作为确定当前电机停止或者运动状态的说明，装载 S1 的状态到 Y1，就可以驱动灯之类的负载观察电机的运动状态，电机速度不为 0，Y1 为 ON，电机速度为 0，Y1 为 OFF。

1.1.6 定时器状态继电器

MOTEC 智能驱动器内置可编程控制器拥有 32 个 16 位长度的 1ms 递减定时器，最大定时器装载值为 65535。定时器状态继电器对应着相应的定时器装载寄存器，在定时器装载寄存器内值不为 0 的时候，是 OFF 状态，定时器装载寄存器内的值为 0 的时候是 ON 状态，以下 T 继电器代指定时器状态继电器。定时器状态继电器的开关状态只与定时器装载寄存器内的数值有关，不能作为输出继电器使用。

一般来讲，定时器状态继电器要和定时器装载寄存器一起使用，关于定时器状态继电器的具体应用举例，可见 1.2.4 中与定时器装载寄存器一起使用的举例。

注意：定时器装载寄存器上电以后会被初始化成 0，但是相应的 T 继电器被初始化为 OFF 状态，只有在相应的对定时器装载寄存器有操作的时候，才会在定时器装载寄存器内的值不为 0 时变成 OFF 状态，为 0 的时候变成 ON 状态。

1.1.7 计数器状态继电器

MOTEC 智能驱动器内置可编程控制器拥有 32 个 16 位长度的递减计数器，最大计数器装载值为 65535。计数器状态继电器对应着相应的计数器装载寄存器，在计数器装载寄存器内值不为 0 的时候，是 OFF 状态，计数器装载寄存器内的值为 0 的时候是 ON 状态，以下 C 继电器代指计数器状态继电器。C 继电器的开关状态只与计数器装载寄存器内的数值有关，不能作为输出继电器使用。

一般来讲，计数器状态继电器要和计数器装载寄存器一起使用，关于计数器状态继电器的具体应用举例，可见 1.2.5 中与计数器装载寄存器一起使用的举例。

注意：计数器装载寄存器上电以后会被初始化成 0，但是相应的 C 继电器被初始化为 OFF 状态，只有在相应的对计数器装载寄存器有操作的时候，才会在计数器装载寄存器内的值不为 0 时变成 OFF 状态，为 0 的时候变成 ON 状态。

1.2 数据寄存器

MOTEC 智能驱动器内置可编程控制器拥有 500 个 16 位普通数据寄存器，12 个 16 位特殊数据寄存器，驱动器内部使用寄存器，105 个 CANopen 专用 32 位寄存器，除了 CANopen 专用数据寄存器外，其余都可以使用可编程控制器相关指令进行操作，具体信息见表 1-6。

表 1-6 数据寄存器具体信息

项目	个数	标识符	数据长度	编号	程序和通信访问地址
定时器装载寄存器	32	TML	16 位	TLD0-TLD31	0x7000-0x701F
计数器装载寄存器	32	CML	16 位	CLD0-CLD31	0x8000-0x801F
16 位普通数据寄存器	500	D	16 位	D0-D499	0x9000-0x91F3
16 位特殊数据寄存器	12	D	16 位	D500-D511	0x91F4-0x91FF
CANopen 专用寄存器	105	CANopen	32 位	CANopen 专用	对 PLC 暂不开放

1.2.1 不同数据可编程控制器内部存储和表示

在 MOTEC 智能驱动器内置可编程控制器内部，所有的数据是以二进制方式存取的，其中继电器长度为 1 位，数据寄存器的长度为 16 位。

使用数据寄存器，可以表示以下几种数据：

16 位正整数

取值范围是 0～65535，每个单独的数据寄存器就可以表示一个 16 位正整数，程序所使用的数据寄存器的地址即为该数据的存放地址。

Dn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

——Dn 表示正整数 38981。

16 位整数

取值范围是-32768～32767，每个单独的数据寄存器就可以表示一个 16 位整数，该 16 位的最高位是数据的符号位。程序所使用的数据寄存器的地址即为该数据的存放地址。

Dn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

——Dn 表示整数-6213。

32 位正整数

取值范围是 0～4294967295，要表示一个 32 位正整数，需要使用两个 16 位数据寄存器，将地址连续的两个 16 位寄存器作为连接起来存放 32 位数据，指定用于低 16 位寄存器地址。低 16 位寄存器 Dn 作为 32 位数据的低 16 位，低十六位寄存器地址加 1 的十六位寄存器 Dn+1 作为 32 位数据的高 16 位寄存器。

Dn+1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

Dn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

——Dn 作为 32 位数据的低 16 位，Dn+1 作为 32 位数据的高 16 位，Dn+1 和 Dn 共同表示 32 位正整数 2554697797。

32 位整数

取值范围是-2147483648～2147483647，要表示一个 32 位整数，需要使用两个 16 位数据寄存器，将地址连续的两个 16 位寄存器连接起来存放 32 位数据，指定用于低 16 位寄存器地址。低 16 位寄存器 Dn 作为 32 位数据的低 16 位，低十六位寄存器地址加 1 的十六位寄存器 Dn+1 作为 32 位数据的高 16 位寄存器，Dn+1 的最高位作为符号位。

Dn+1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

Dn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

——Dn 作为 32 位数据的低 16 位，Dn+1 作为 32 位数据的高 16 位，Dn+1 和 Dn 共同表示 32 位整数-407214149。

浮点数

MOTEC 智能驱动器内置可编程控制器的数据存储器可以表示浮点数，对于浮点数，有两种表示方式，二进制浮点数和十进制浮点数，下面将分别介绍。

二进制浮点数

MOTEC 智能驱动器内置可编程控制器可以使用单精度浮点数，是一个 32 位数，占用两个连续的普通数据寄存器 Dn 和 Dn+1，其中 Dn 作为 32 位数据的低 16 位，Dn+1 作为 32 位数据的高 16 位。形式如下

Dn+1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	E7	E6	～～～～～～～				E1	E0	A22	A21	～～				
	指数段 8 位								尾数段 23 位						

Dn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														A1	A0

尾数符号

二进制浮点数 ＝ 尾数【A22~A0】×2^【E7~E0】；

——Dn 作为 32 位数据的低 16 位，Dn+1 作为 32 位数据的高 16 位，Dn+1 和 Dn 共同表示一个单精度浮点数，包括符号位 1 位，阶码 8 位，尾数 23 位。其数值范围为-3.4E38～3.4E38，单精度浮点数最多有 7 位十进制有效数字。浮点数的正负由高 16 位寄存器的最高位决定。二进制浮点数可以使用浮点数的逻辑运算或者算术运算的指令参与运算，也可以转换成 10 进制浮点数或者整型数据存储和运算。

十进制浮点数

用户难以读取和判断二进制浮点数，因此，在实际操作中，为了方便用户判断，MOTEC 智能驱动器内置可编程控制器拥有指令可以将二进制浮点数转换成 10 进制浮点数。

10 进制浮点数同样需要使用 32 位来存储，一个 16 为寄存器 Dn 作为尾数数据，Dn 地址的下一个 16 位寄存器 Dn+1 作为指数数据，尾数和指数都是有符号数。

Dn+1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E15	E14	～～～～～～～～～～～～～～～～～～～～～～～												E1	E0
16 位指数段															

Dn

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	～～～～～～～～～～～～～～～～～～～～～～～～～												A1	A0
16 位尾数段															

十进制浮点数 ＝ 尾数【E15~E0】×1000^【A22~A0】；

——Dn 作为 32 位数据的低 16 位，Dn+1 作为 32 位数据的高 16 位，Dn+1 和 Dn 共同表示一个单精度浮点数，由于需要互相转换，10 进制浮点数对于表示范围有自己的限制。

尾数 = ±1000~9999;

指数 = -40～+40；

多余的位数会按照四舍五入自动舍掉。

例如，987654321，在使用 10 进制浮点数时，Dn 存放的内容为 9877，Dn+1 存放的内容为 5。

十进制浮点数只能作为用户读取和显示方便，不能参与实际运算。如需使用 10 进制浮点数进行运算，那么必须将十进制浮点数转换为二进制浮点数之后才能使用。10 进制浮点数同样不能转换成整型，需要先转换成二进制浮点数。

1.2.2 普通数据寄存器

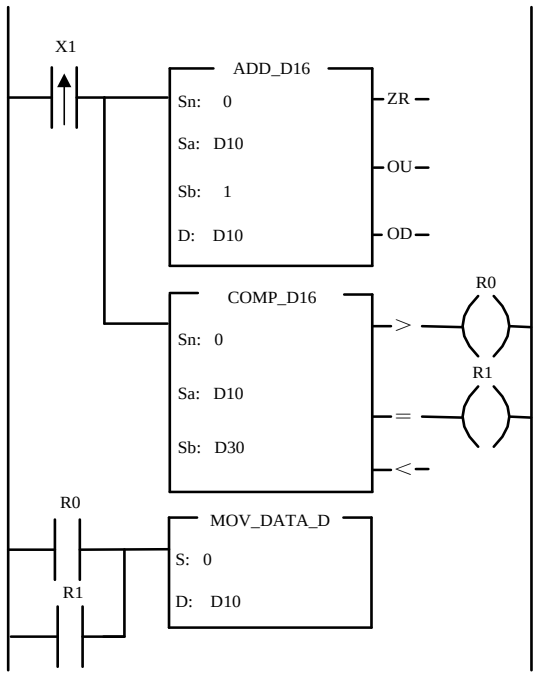
数据寄存器使用 D 寄存器表示，用于存放 16 位长度的数据。用户可以无限制次数的使用和修改，根据使用的数据格式的不同，可以存放 16 位数据整数，32 位数据整数，二进制浮点数和十进制浮点数等。

用户可以使用从 flash 内读取 D 寄存器命令来读取存在 Flash 内部的寄存器值，也可使用存寄存器的值进 Flash 来将当前的寄存器值存进 Flash 用以作为掉电保存。

普通寄存器可以作为输入条件和输出条件参与到程序的运算。

例如

梯形图



助记符

```
LD_R                X1
ADD_D16             FUNC
                    0
                    0
                    D10
                    1
                    D10
DATA_16_COMPARE     FUNC
                    1
                    0
                    D10
                    D30
                    R0
                    R1
LD                  R0
OR                  R1
MOV_DATA_D          FUNC
                     0
                     0
                     D10
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X1	装载输入继电器 X1 的上升沿
ADD_D16	FUNC	将普通寄存器 D10 内的内容加 1，结果放入寄存器 D10 内
	0	
	0	
	D10	
	1	
	D10	
DATA_16_COMPARE	FUNC	寄存器 D10>D30 逻辑运算结果输出到继电器 R0, 寄存器 D10=D30 逻辑运算结果输出到继电器 R1
	1	
	0	
	D10	
	D30	
	R0	
	R1	
LD	R0	装载继电器 R0 状态
OR	R1	装载继电器 R1 状态，与 R0 相或
MOV_DATA_D	FUNC	将寄存器 D10 写入 0
	0	
	0	
	D10	

程序说明：本程序是对普通数据寄存器的操作，程序中的各种指令会在以后详细介绍，本程序的功能就是每次当 X1 有上升沿时，就会在 D10 寄存器加 1，然后重新放入 D10 中，直到进行数据逻辑运算，D10>=D30 时，就将 D10 的内容清零。程序能完成此类循环结构的功能。

1.2.3 特殊数据寄存器

特殊数据寄存器同样用 D 来表示，特殊数据寄存器是在 D 寄存器地址后增加的一些特殊功能的寄存器，具体信息见表 1-7。

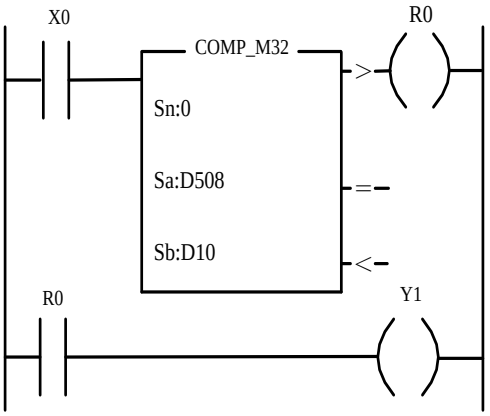
表 1-7

项目	编号	说明	数据长度	程序和通信访问地址
密码	D500~D501	保留	32 位	0x91F4-0x91F5
堆栈值	D502~D503	用来保存当前程序运行的堆栈值	32 位	0x91F6-0x91F7
当前程序指针	D504~D505	保存当前用户的程序指针位置	32 位	0x91F8-0x91F9
周期时间	D506~D507	用来保存上一个程序循环的时间，时间单位是 us	32 位	0x91FA-0x91FB
电机当前位置	D508~D509	电机当前位置	32 位	0x91FC-0x91FD

使用限制：但是这些寄存器只能作为输入条件参与运算，不能作为输出条件，特殊数据寄存器都是使用两个 16 位数据寄存器作为一个 32 位数据寄存器，低地址的 16 位寄存器作为 32 位中的低 16 位，高地址的 16 位数据寄存器作为 32 位数据的高 16 位。
用户可以使用这些特殊寄存器进行运算。

例如

梯形图



助记符

LD	X0
DATA_32_COMPARE	FUNC
	1
	0
	D508
	D10
	R0
LD	R0
OUT	Y1

指令解释（助记符部分）

指令内容		指令说明
LD	X0	装载 X0 的状态
DATA_32_COMPARE	FUNC	特殊数据寄存器 D508，D509 与 D10，D11 进
	1	行大于比较，结果输出到 R0
	0	
	D508	
	D10	
	R0	
LD	R0	装载继电器 R0 状态
OUT	Y1	输出到输出继电器 Y1

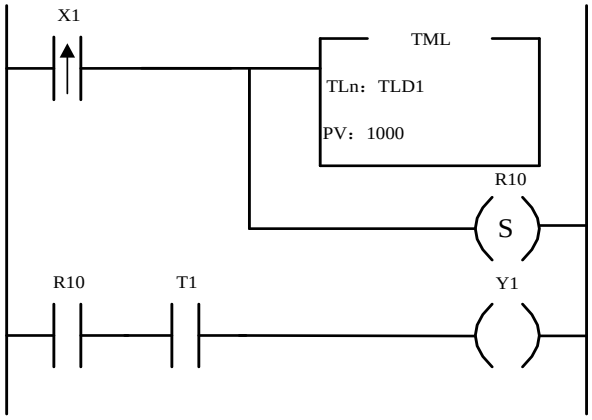
程序说明：本程序使用特殊数据寄存器 D508，特殊数据寄存器 D508 和 D509 两个 16 位数据寄存器作为一个 32 位数据寄存器存储当前电机实际的运动距离，程序的功能是当 X0 为 ON 时，电机的实际位置大于由两个 16 位数据寄存器 D10 和 D11 组成的 32 位数据寄存器里面的内容时，Y1 就会输出 ON 状态。

1.2.4 定时器装载寄存器

MOTEC 智能驱动器内置可编程控制器内部具有 32 个时基为 1MS 的定时器。定时器寄存器是一个 16 位的递减定时器，使用 TML 表示，当定时器装载寄存器内的数据不为 0 时，该寄存器内的值会每 1ms 时间减 1，相应的定时器状态继电器为 OFF，直到减到 0 为止，停止计时，相应的定时器状态继电器为 ON。使用定时器可以方便的进行计时。

例如

梯形图



助记符

```
LD_R    X1
TML     FUNC
          0
        TLD1
        1000
SET     R10
LD      R10
AND     T1
OUT     Y1
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X1	装载输入继电器 X1 的上升沿
TML	FUNC	将常数 1000 写入定时器 TLD1 的装载寄存器内
	0	
	TLD1	
	1000	
SET	R10	将辅助继电器 R10 置 ON
LD	R10	装载辅助继电器 R10
AND	T1	当前栈值与定时器 T1 的状态继电器 T1 相与
OUT	Y1	将当前的栈值输出到输出继电器 Y1

程序说明：本程序使用定时器 T1，系统 X1 检测到上升沿，启动定时器 T1，并且置位辅助继电器 R10，等到定时器状态继电器 T1 为 ON 时，输出 ON 到继电器 Y1。程序功能是 X1 有上升沿时，过 1000ms 即 1s 以后会输出 ON 到 Y1。

注意：T 继电器只有在定时器装载寄存器内的值不为 0 时才会变成 OFF 状态，所以会在程序中加入辅助继电器 R10，在装载定时器初值时才置位 R10，保证系统在没有必要的情况下会输出 ON 到 Y1。

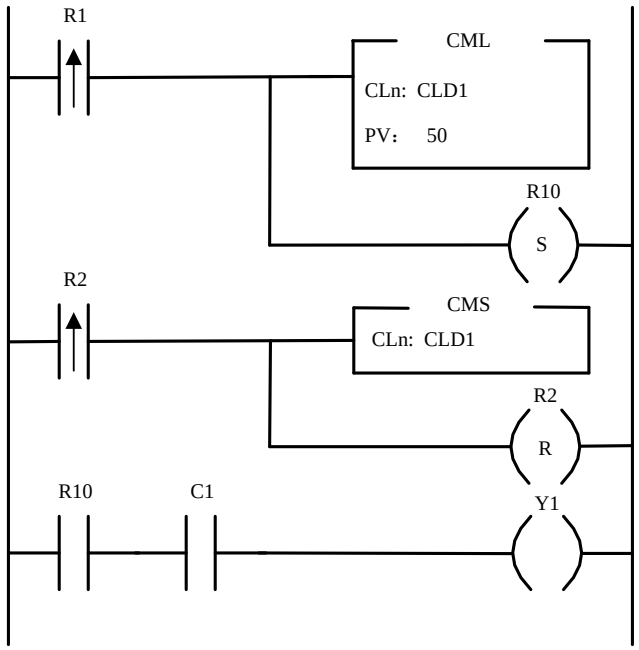
1.2.5 计数器装载寄存器

MOTEC 智能驱动器内置可编程控制器内部具有 32 个递减计数器。计数器装载寄存器的长度为 16 位，使用 CML 表示，当计数器装载寄存器内的数据不为 0 时，相应的计数器状态继电器为 OFF，每次使用了计数器计数指令以后，计数器装载寄存器内的数值减 1，直到该寄存器内的数值为 0，停止计数，相应的计数器状态继电器为 ON。

使用计数器可以方便的进行计数。

例如

梯形图



助记符

```
LD_R    R1
CML     FUNC
          0
        CLD1
        50
SET     R10
LD_R    R2
CMS     FUNC
        CLD1
RESET   R2
LD      R10
AND     C1
OUT     Y1
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	R1	装载输入继电器 R1 的上升沿
CML	FUNC	向计数器 CLD1 装载寄存器内写入常数
	0	50
	CLD1	
	50	
SET	R10	将辅助继电器 R10 置位为 ON
LD_R	R2	装载输入继电器 R2 的上升沿
CMS	FUNC	计数器 CLD1 的装载寄存器内的数值减 1
	CLD1	
RESET	R2	清除继电器 R2
LD	R10	装载辅助继电器 R10 的状态
AND	C1	当前栈值与计数器 C1 的状态继电器相与
OUT	Y1	将当前栈值输出到输出继电器 Y1

程序说明：本程序使用计数器 C1，输入辅助继电器 R1 的上升沿会触发计数器 C1 的初始值写入，输入继电器 R2 的上升沿会触发计数器 C1 的递减计数，直到输入继电器 R2 的上升沿达到 50 次的时候，会输出 ON 到输出继电器 Y1。

注意：C 继电器只有在计数器装载寄存器内的值不为 0 时才会变成 OFF 状态，所以会在程序中加入辅助继电器 R10，在装载定时器初值时才置位 R10，保证系统不是在计数器为 0 的时候不会一直输出 ON 到 Y1。

1.2.6 驱动器内部数据寄存器

MOTEC 智能驱动器内置可编程控制器内部数据寄存器，用来存放一些驱动器的相关的数据，使用 P 表示，一些地址的 P 寄存器会有读写限制，P 寄存器的详细信息请详见各个型号的参数表：

智能步进驱动器请参考《MOTEC 智能步进驱动器参数表说明 V2.1》；

直流伺服驱动器请参考《MOTEC 直流伺服驱动器(标准直流)参数表说明 V2.1》；

交流伺服驱动器请参考《MOTECβ 交流伺服驱动器参数表说明 V2.1》；

P 寄存器可以参与到 16 位数据寄存器的逻辑运算和算术运算，并可以将 P 寄存器的数据转移到 D 寄存器进行更多的操作。

注意：P 寄存器程序和通信访问地址为 0-Pmax(终止地址与驱动器型号有关)。

第 2 章 基本指令

基本指令的操作对象是各类继电器，本章主要讲基本指令的语句结构和使用方式。

2.1 时序控制基本操作指令

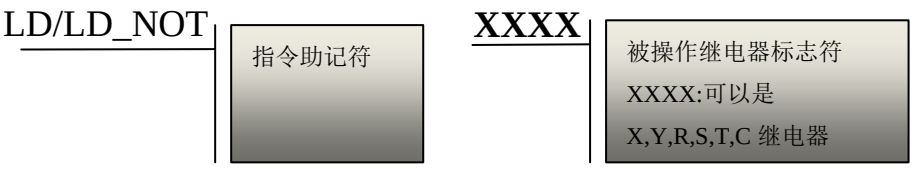
LD/LD_NOT：装载/取反装载指令

LD：开始逻辑运算，装载指定继电器当前的状态，可视为是装载常开继电器状态指令。

LD_NOT：开始逻辑运算，装载指定继电器的当前状态并且取反，可视为是装载常闭继电器状态指令。

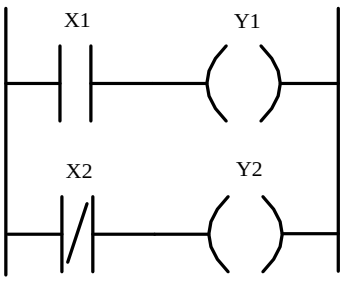
LD/LD_NOT 指令可以操作所有的继电器，同样包括输出继电器和特殊继电器等。

LD/LD_NOT 指令的助记符语句结构为



例如：

梯形图



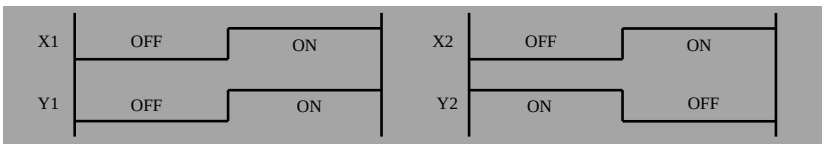
助记符

LD	X1
OUT	Y1
LD_NOT	X2
OUT	Y2

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
OUT	Y1	输出到输出继电器 Y1
LD_NOT	X2	装载输入继电器 X2 的状态并且取反
OUT	Y2	输出到输出继电器 Y2

程序说明：如果当前 X1 为 ON,X2 为 ON，那么 Y1 就为 ON，由于 LD_NOT 会将装载的继电器值取反，所有 Y2 的状态为 OFF。如下图



注意：LD/LD_NOT 指令必须从母线开始，针对不同的继电器（常开或者常闭），用户需要根据自己的功能要求灵活使用 LD 指令或者 LD_NOT 指令。

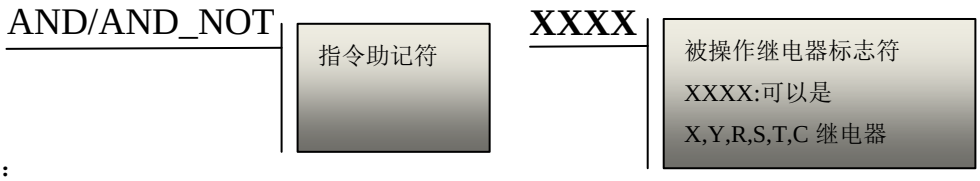
AND/AND_NOT：与/与非指令

AND：与操作指令，该指令会将指定的继电器的状态与当前程序栈值进行相与的操作；

AND_NOT：与非操作指令，该指令会将制定的继电器的状态与当前程序栈值进行相与非的操作。

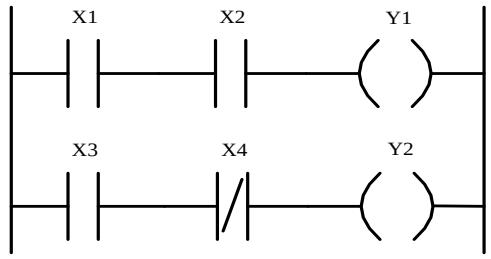
AND/AND_NOT 指令可以操作所有的继电器，同样包括输出继电器和特殊继电器等。

AND/AND_NOT 指令的助记符语句结构为



例如：

梯形图



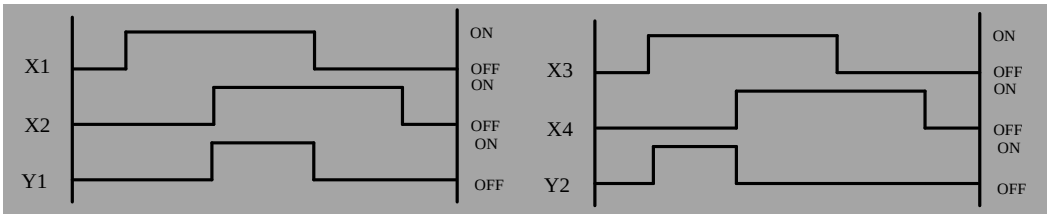
助记符

LD	X1
AND	X2
OUT	Y1
LD	X3
AND_NOT	X4
OUT	Y2

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
AND	X2	输入继电器 X2 的状态与当前栈值相与
OUT	Y1	输出到输出继电器 Y1
LD	X3	装载输入继电器 X3 的状态
AND_NOT	X4	输入继电器 X4 的状态与当前栈值相与非
OUT	Y2	输出到输出继电器 Y2

程序说明：该程序将输入继电器 X1 的值与 X2 的值相与，再输出到输出继电器 Y1。将输入继电器 X3 的值与 X4 的值相与非，再输出到输出继电器 Y2。例如，如果 X1,X2,X3,X4 全为 ON 状态，则 Y1 为 ON，Y2 为 OFF。如下图



注意：AND/AND_NOT 指令不能从母线开始，针对不同的继电器（常开或者常闭），用户需要根据自己的功能要求灵活使用 AND 指令或者 AND_NOT 指令。

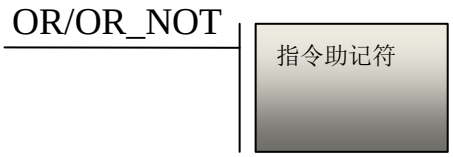
OR/OR_NOT：或/或非指令

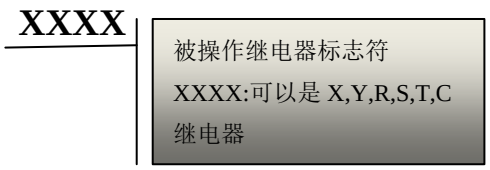
OR：或操作指令，该指令会将指定的继电器的状态与当前程序栈值进行相或的操作；

OR_NOT：或非操作指令，该指令会将指定的继电器的状态与当前程序栈值进行相或非的操作。

OR / OR _NOT 指令可以操作所有的继电器，同样包括输出继电器和特殊继电器等。

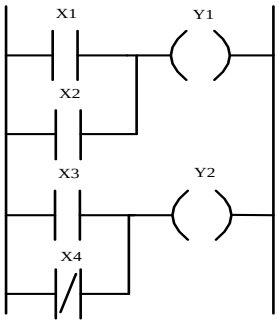
OR / OR _NOT 指令的助记符语句结构为





例如

梯形图



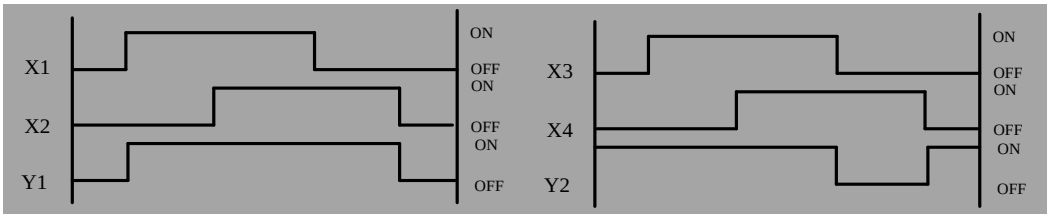
助记符

LD	X1
OR	X2
OUT	Y1
LD	X3
OR_NOT	X4
OUT	Y2

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
OR	X2	输入继电器 X2 的状态与当前栈值相或
OUT	Y1	输出到输出继电器 Y1
LD	X3	装载输入继电器 X3 的状态
OR_NOT	X4	输入继电器 X4 的状态与当前栈值相或非
OUT	Y2	输出到输出继电器 Y2

程序说明：该程序将输入继电器 X1 的值与 X2 的值相或，再输出到输出继电器 Y1。将输入继电器 X3 的值与 X4 的值相或非，再输出到输出继电器 Y2。例如，如果 X1,X2,全为 OFF 状态，X3,X4 为 ON 状态，则 Y1 为 OFF，Y2 为 ON。如下图



注意：OR/OR_NOT 指令可以从母线开始，针对不同的继电器（常开或者常闭），用户需要根据自己的功能要求灵活使用 OR 指令或者 OR _NOT 指令。

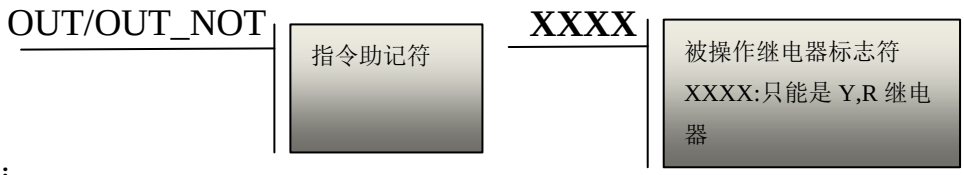
OUT/OUT_NOT：输出/取反输出指令

OUT：输出逻辑运算结果到指定的继电器，可视为是输出常开继电器状态指令。

OUT_NOT：当前逻辑运算结果取反并且输出到指定的继电器，可视为是输出常闭继电器状态指令。

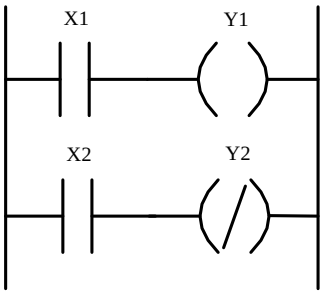
OUT / OUT _NOT 指令不能操作所有的继电器，只可以操作输出继电器和普通辅助继电器。

OUT / OUT _NOT 指令的助记符语句结构为



例如：

梯形图



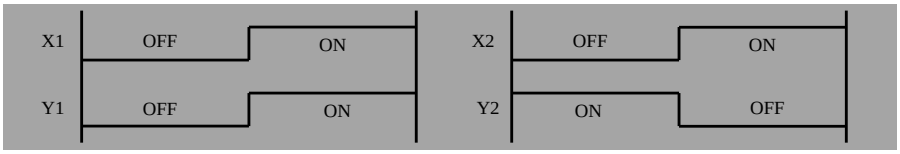
助记符

LD	X1
OUT	Y1
LD	X2
OUT_NOT	Y2

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
OUT	Y1	输出到输出继电器 Y1
LD	X2	装载输入继电器 X2 的状态
OUT_NOT	Y2	取反并输出到输出继电器 Y2

程序说明：如果当前 X1 为 ON,X2 为 ON，那么 Y1 就为 ON，由于 OUT_NOT 会将装载的继电器值取反，所以 Y2 的状态为 OFF。如下图



注意：OUT/OUT_NOT 指令必须以母线结束，针对不同的继电器（常开或者常闭），用户需要根据自己的功能要求灵活使用 OUT 指令或者 OUT_NOT 指令。

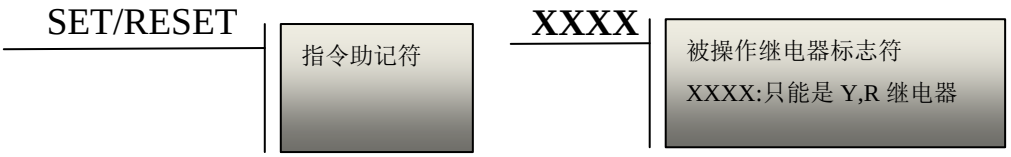
SET/RESET：置位/清除指令

SET：满足执行条件时，输出 ON 到指定继电器，并且保持 ON 的状态。

RESET：满足执行条件时，输出 OFF 到指定继电器，并且保持 OFF 的状态。

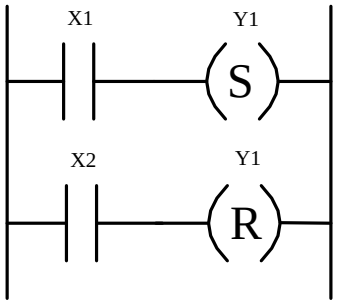
SET/ RESET 指令类似于 OUT / OUT _NOT 指令，不能操作所有的继电器，只可以操作输出继电器和普通辅助继电器。置位指令和清除指令会使继电器持续保持输出状态，与 OUT / OUT _NOT 不同。

SET/ RESET 指令的助记符语句结构为



例如：

梯形图



助记符

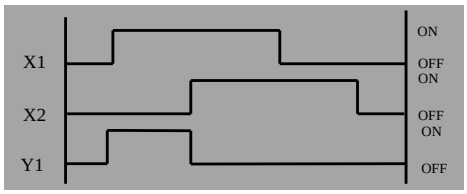
LD	X1
SET	Y1
LD	X2
RESET	Y1

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
SET	Y1	置位输出继电器 Y1
LD	X2	装载输入继电器 X2 的状态
RESET	Y1	清除输出继电器 Y1

程序说明：如果系统检测到 X1 为 ON 状态，则会强制置位 Y1 为 ON 状态，并且即使 X1 由 ON 变为 OFF，Y1 同样会保持 ON 状态。

如果系统检测到 X2 为 ON 状态，则会强制清除 Y1 为 OFF 状态，并且即使 X2 由 ON 变为 OFF，Y1 同样会保持 OFF 状态，如下图。

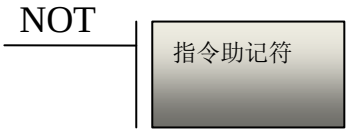


注意：SET/ RESET 指令必须以母线结束，用户需要根据自己的功能要求灵活使用 SET 指令或者 RESET 指令。

NOT：取反指令

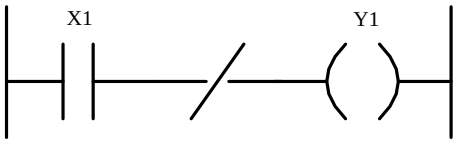
NOT：对当前的执行条件进行取反操作，如果当前栈值为 ON，则输出 OFF，如果当前栈值为 OFF，则输出 ON。NOT 指令没有指令参数。

NOT 指令的助记符语句结构为



例如：

梯形图 助记符



LD X1
NOT
OUT Y1

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
NOT		当前栈值取反
OUT	Y1	输出到输出继电器 Y1

程序说明：如果 X1 为 ON，则经过一次取反，输出到 Y1 为 OFF。

注意：NOT 指令不能在母线开始或者母线结束处。

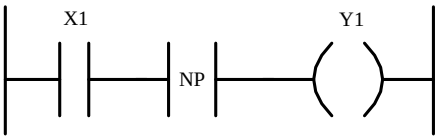
NOP：空指令

NOP：空指令，不会进行任何操作，对程序的执行无任何影响。

NOP 指令的助记符语句结构为

例如：

梯形图 助记符



LD X1
NOP
OUT Y1

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
NOP		空指令
OUT	Y1	输出到输出继电器 Y1

程序说明：如果 X1 为 ON，则经过 NOP 空指令，当前栈值无变化，输出到 Y1 为 ON。

2.2 沿操作指令

沿操作是检测到继电器的上升沿（由 OFF 变为 ON）或者检测到继电器的下降沿（由 ON 变为 OFF）的操作，与时序基本控制指令相同，都有装载、与、或、非等逻辑运算操作。与之不同的是，沿操作指令只在一个周期内有效。

LD_R/ LD_F：装载上升沿/装载下降沿

LD_R：仅装载信号的上升沿，该装载操作只发生在一个扫描周期。

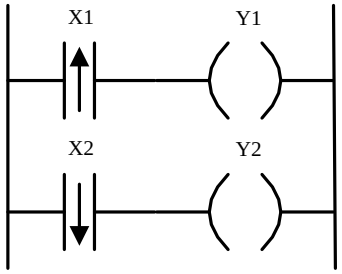
LD_F：仅装载信号的下降沿，该装载操作只发生在一个扫描周期。

LD_R/ LD_F 指令可以操作所有的继电器，同样包括输出继电器和特殊继电器等。

LD_R/ LD_F 指令的助记符语句结构为

例如：

梯形图



助记符

LD_R	X1
OUT	Y1
LD_F	X2
OUT	Y2

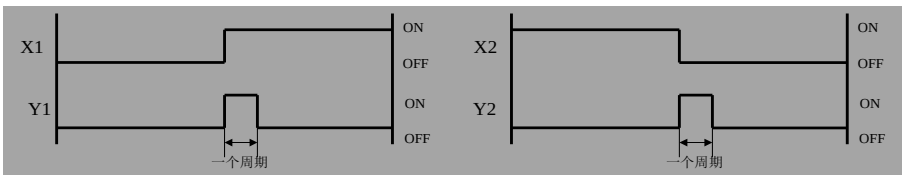
指令解释（助记符部分）：

指令内容		指令说明
LD_R	X1	装载输入继电器 X1 的上升沿状态
OUT	Y1	输出到输出继电器 Y1
LD_F	X2	装载输入继电器 X2 的下降沿状态
OUT	Y2	输出到输出继电器 Y2

程序说明：如果在某个周期检测到 X1 由 OFF 变为 ON，则输出 Y1 为 ON，并且下一个周期 X1 持续为 OFF 时，Y1 为 OFF。

如果在某个周期检测到 X2 由 ON 变为 OFF，则输出 Y2 为 ON，并且下一个周期 X2 持续为 OFF 时，Y2 为 OFF。

如下图



注意：LD_R/ LD_F 指令必须从母线开始，针对不同的继电器（常开或者常闭），用户需要根据自己的功能要求灵活使用 LD_R 指令或者 LD_F 指令。

AND_R/AND_F：与上升沿有效相与/与下降沿有效相与

AND_R：与操作指令，该指令会将指定的继电器的上升沿状态与当前程序栈值进行相与的操作；

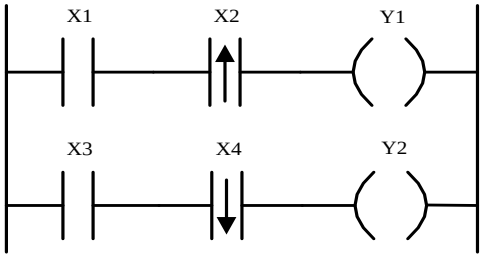
AND_F：与操作指令，该指令会将指定的继电器的下降沿状态与当前程序栈值进行相与的操作。

AND_R / AND_F 指令可以操作所有的继电器，同样包括输出继电器和特殊继电器等。

AND_R / AND_F 指令的助记符语句结构为

例如：

梯形图



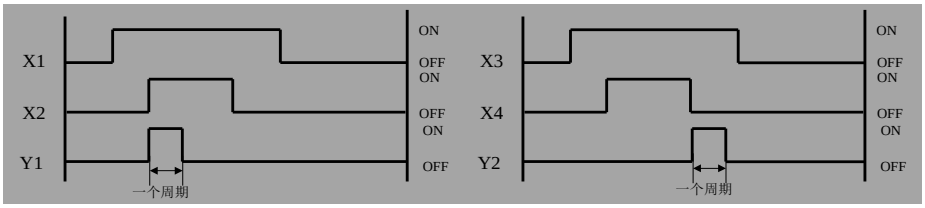
助记符

LD	X1
AND_R	X2
OUT	Y1
LD	X3
AND_F	X4
OUT	Y2

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
AND_R	X2	输入继电器 X2 的上升沿状态与当前栈值相与
OUT	Y1	输出到输出继电器 Y1
LD	X3	装载输入继电器 X3 的状态
AND_F	X4	输入继电器 X4 的下降沿状态与当前栈值相与
OUT	Y2	输出到输出继电器 Y2

程序说明：该程序将输入继电器 X1 的值与 X2 的上升沿值相与，再输出到输出继电器 Y1。如果 X1 为 ON，只有在 X2 由 OFF 到 ON 变化时，会输出 Y1 为 ON，并且在下一个周期输出 Y1 为 OFF。将输入继电器 X3 的值与 X4 的下降沿值相与，再输出到输出继电器 Y2。如果 X3 为 ON，只有在 X4 由 ON 到 OFF 变化时，会输出 Y2 为 ON，并且在下一个周期输出 Y2 为 OFF。如下图



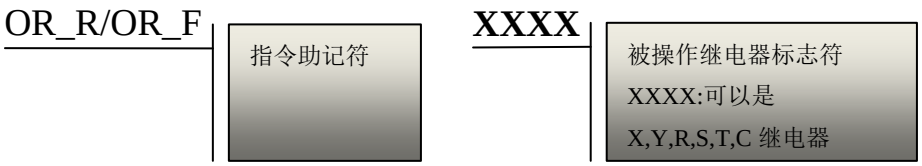
注意：AND_R / AND_F 指令不能从母线开始，针对不同的继电器（常开或者常闭），用户需要根据自己的功能要求灵活使用 AND_R 指令或者 AND_F 指令。

OR_R/OR_F：与上升沿有效相或/与下降沿有效相或

OR_R：或操作指令，该指令会将指定的继电器的上升沿状态与当前程序栈值进行相或的操作；

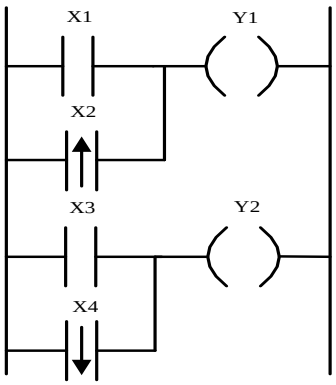
AOR_F：或操作指令，该指令会将制定的继电器的下降沿状态与当前程序栈值进行相或的操作，该指令可以操作所有的继电器，同样包括输出继电器和特殊继电器等。

OR_R / OR_F 指令的助记符语句结构为



例如：

梯形图



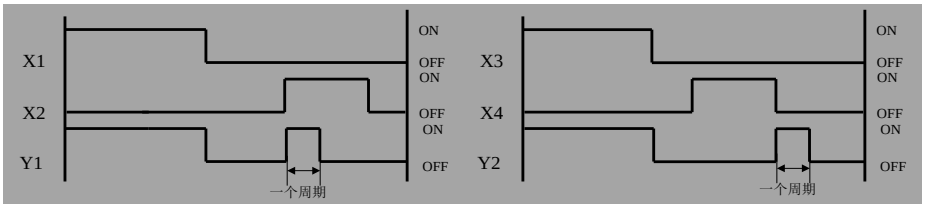
助记符

LD	X1
OR_R	X2
OUT	Y1
LD	X3
OR_F	X4
OUT	Y2

指令解释（助记符部分）

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
OR_R	X2	输入继电器 X2 的上升沿状态与当前栈值相或
OUT	Y1	输出到输出继电器 Y1
LD	X3	装载输入继电器 X3 的状态
OR_F	X4	输入继电器 X4 的下降沿状态与当前栈值相或
OUT	Y2	输出到输出继电器 Y2

程序说明：该程序将输入继电器 X1 的值与 X2 的上升沿值相或，再输出到输出继电器 Y1。如果 X1 为 OFF，只有在 X2 由 OFF 到 ON 变化时，会输出 Y1 为 ON，并且在下一个周期输出 Y1 为 OFF。将输入继电器 X3 的值与 X4 的下降沿值相或，再输出到输出继电器 Y2。如果 X3 为 OFF，只有在 X4 由 ON 到 OFF 变化时，会输出 Y2 为 ON，并且在下一个周期输出 Y2 为 OFF，如下图。



注意：OR_R / OR_F 指令可以从母线开始，针对不同的继电器（常开或者常闭），用户需要根据自己的功能要求灵活使用 OR_R 指令或者 OR_F 指令。

DR/DF：上升沿检测/下降沿检测

DR：当检测到输入触发信号的上升沿时，仅将触点闭合一个扫描周期。

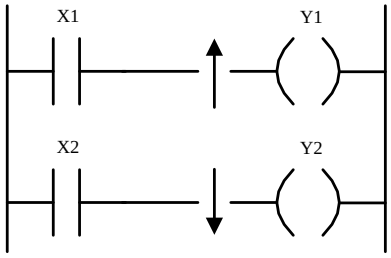
DF：当检测到输入触发信号的下降沿时，仅将触点闭合一个扫描周期。

DR / DF 指令可以放在程序中用来检测系统的沿变化。

DR / DF 指令的助记符语句结构为

例如：

梯形图



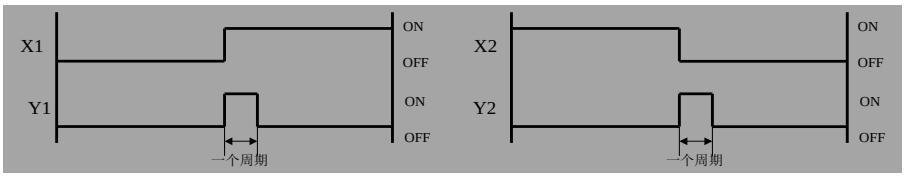
助记符

```
LD      X1
DR
OUT     Y1
LD      X2
DF
OUT     Y2
```

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
DR		检测上升沿信号
OUT	Y1	输出到 Y1
LD	X2	装载输入继电器 X2 的状态
DF		检测下降沿信号
OUT	Y2	输出到 Y2

程序说明：在某个扫描周期内，检测到本周期与下个周期相比，产生了上升沿或者下降沿，DR / DF 就会触发一个 ON 的信号。当 X1 有上升沿出现时，Y1 会在当前周期为 ON 并且在下一个周期为 OFF，当 X2 有下降沿出现时，Y2 会在当前周期为 ON 并且在下一个周期为 OFF，如下图。



注意：DR / DF 指令不能在母线开始或者母线结束处，只能检测到上升沿或者下降沿，检测是以本周期当前的栈值和上个扫描周期时的栈值相比较，是否有上升沿或者下降沿。

2.3 块操作指令

ANDB：块相与

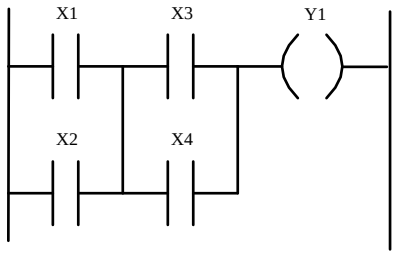
ANDB：将多个逻辑块串联相与使用。

ANDB 指令是将两个不同的逻辑块串联起来，并分别计算两个逻辑块的值，并且将两个值相与。

ANDB 指令助记符语句结构为

例如：

梯形图



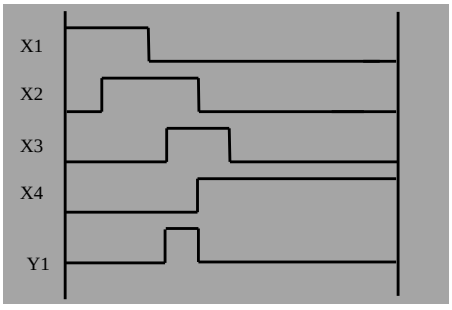
助记符

LD	X1
OR	X2
LD	X3
OR	X4
ANDB	
OUT	Y1

指令解释(助记符部分)

指令内容		指令说明
LD	X1	装载 X1 的状态
OR	X2	X2 的状态与当前栈值相或
LD	X3	装载 X3 的状态
OR	X4	X4 的状态与当前栈值相或
ANDB		将两个块相与
OUT	Y1	输出当前栈值到 Y1

程序说明：该程序就是将 X1 与 X2 的相或得出的逻辑状态与 X3 和 X4 相或的逻辑状态进行块相与的操作，X1 和 X2 之间任何一个为 ON，并且 X3 与 X4 任何一个为 ON 时，Y1 为 ON，如下图。



注意：连续使用块操作时，需要注意每个不同逻辑块的时序问题，块操作的操作对象只有两个，一个是当前的块逻辑计算的值和上一个块逻辑计算的值，用户如果使用该操作，需要合理块操作指令的位置。

ORB：块相或

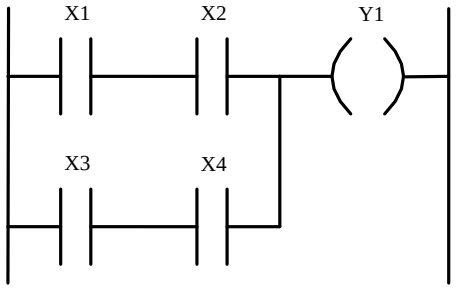
ORB：将多个逻辑块串联相或使用。

ORB 指令是将两个不同的逻辑块串联起来，并分别计算两个逻辑块的值，并且将两个值相或。

ORB 指令助记符语句结构为

例如：

梯形图



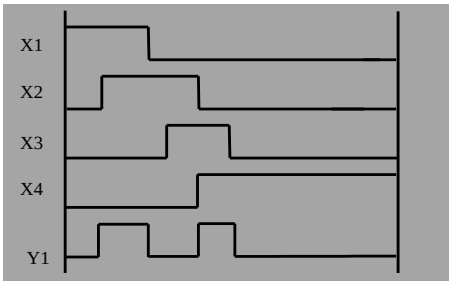
助记符

LD	X1
AND	X2
LD	X3
AND	X4
ORB	
OUT	Y1

指令解释(助记符部分)

指令内容		指令说明
LD	X1	装载 X1 的状态
AND	X2	X2 的状态与当前栈值相与
LD	X3	装载 X3 的状态
AND	X4	X4 的状态与当前栈值相与
ORB		将两个块相或
OUT	Y1	输出当前栈值到 Y1

程序说明：该程序就是将 X1 与 X2 的相与得出的逻辑状态与 X3 和 X4 相与的逻辑状态进行块相或的操作，X1 和 X2 全为 ON，Y1 为 ON，或者 X3 与 X4 全为 ON 时，Y1 为 ON，如下图。



注意：连续使用块操作时，需要注意每个不同逻辑块的时序问题，块操作的操作对象只有两个，一个是当前的块逻辑计算的值和上一个块逻辑计算的值，用户如果使用该操作，需要合理使用块操作指令的位置。

2.4 栈操作指令

栈指令是指要把计算结果存入堆栈或者取出堆栈，方便多次使用某一步的运算结果。包括 MPS，MRD，MPP 三条指令，三条指令要一起使用才能实现相应的功能。

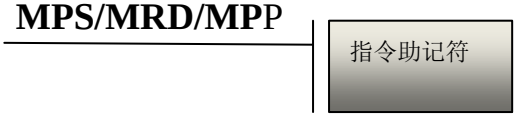
MPS/MRD/MPP：压入堆栈/读取堆栈保留当前栈值/读取堆栈清除当前栈值

MPS：MPS 指令是储存逻辑运算结果进栈，并且继续执行一条指令；

MRD：MRD 指令是读取所存储的上一次逻辑运算结果，并且利用这个结果进行后面程序中的运算。

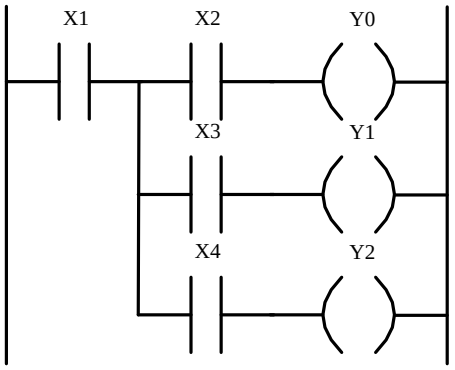
MPP：MPP 指令是读取所存储的上一次逻辑运算结果，利用这个结果进行后面的程序运算，并且会还要清除离他最近的一次 MPS 所存储进入栈值的运算结果。

MPS/MRD/MPP 指令的助记符语句结构为



例如：

梯形图



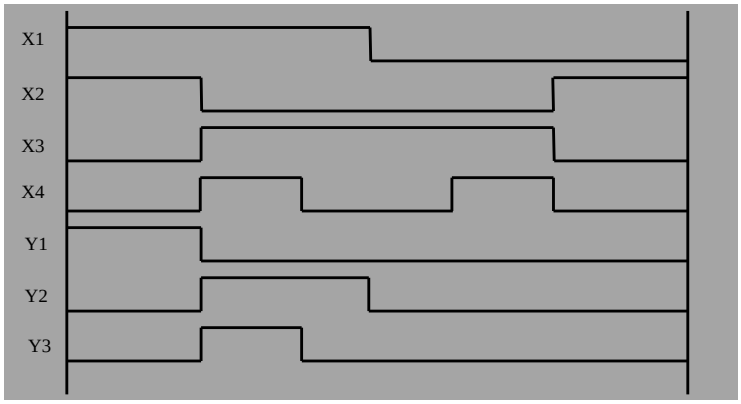
助记符

LD	X1
MPS	
AND	X2
OUT	Y0
MRD	
AND	X3
OUT	Y1
MPP	
AND	X4
OUT	Y2

指令解释（助记符部分）：

指令内容		指令说明
LD	X1	装载 X1 的状态
MPS		将当前结果存入栈中
AND	X2	X2 状态与当前栈值相与
OUT	Y0	输出到 Y0
MRD		提取上一次存储的栈值
AND	X3	X3 状态与当前栈值相与
OUT	Y1	输出到 Y1
MPP		提取上一次存储的栈值并删除
AND	X4	X4 状态与当前栈值相与
OUT	Y2	输出到 Y2

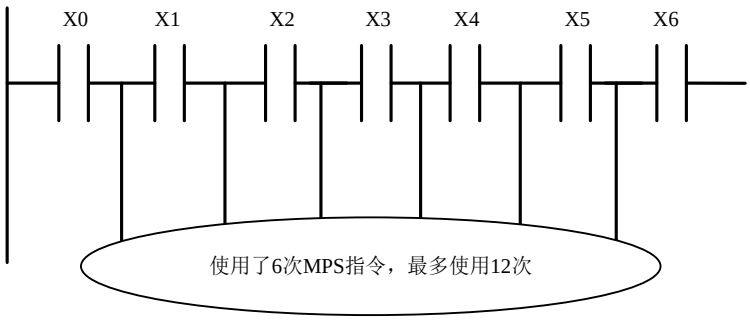
程序说明：该程序使用入栈和出栈指令，将 X1 的状态压入堆栈，再提取栈值与 X2，X3，X4 相与分别输出到 Y0，Y1，和 Y2。该程序可以保存 X1 的状态，再分别与另外几个输入继电器相与。如下图



使用限制：MPS 指令使用次数是有限制的，在出现下一条 MPP 指令之前，MPS 指令只能使用 12 次，超出这个限制，程序会不能正常运行。

例如

梯形图



助记符

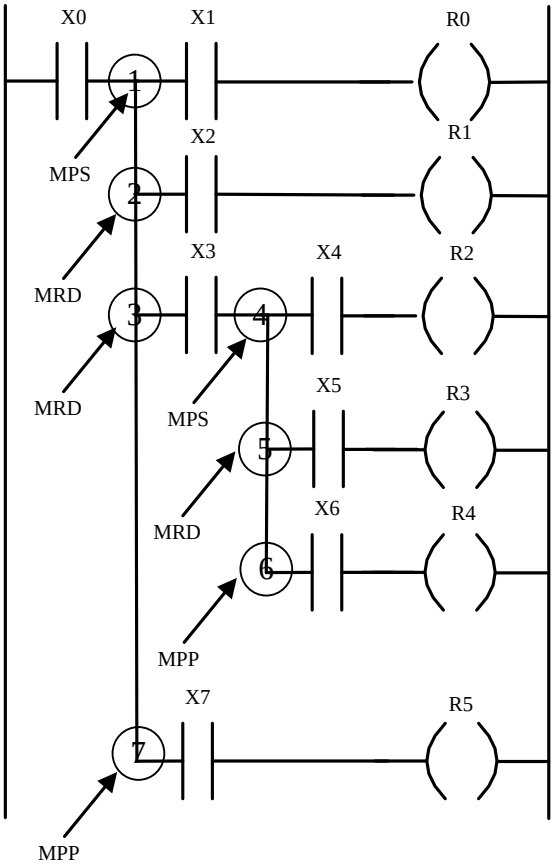
LD	X0
MPS	
AND	X1
MPS	
AND	X2
MPS	
AND	X3
MPS	
AND	X4
MPS	
AND	X5
MPS	
AND	X6

MRD 指令没有次数限制，但是 MRD 至少在使用过一次 MPS 指令之后使用，MRD 指令每次读取的都是距离该指令最近的一次的 MPS，指令存储的逻辑运算结果，并且该 MPS 指令存储的逻辑运算结果并没有使用 MPP 指令清除，如果已经被清除了，那么 MRD 读取的就是上一次未被清除的 MPS 指令存入的结果。

MPP 指令是与 MPS 指令相结合使用，每一个 MPS 指令就会有一个 MPP 指令，同样，只有前面有过 MPS 指令，才能使用 MPP 指令，MPP 指令每次都是读取距离该指令最近一次的 MPS 指令存储的逻辑运算结果并且会删除该栈值。

例如：

梯形图



助记符

LD	X0
MPS	
AND	X1
OUT	R0
MRD	
AND	X2
OUT	R1
MRD	
AND	X3
MPS	
AND	X4
OUT	R2
MRD	
AND	X5
OUT	R3
MPP	
AND	X6
OUT	R4
MPP	
AND	X7
OUT	R5

程序中,使用了多次 MPS,MRD 和 MPP 指令，这些指令有自己相应的编号，在程序中，①处为 MPS，②处为 MRD，③处为 MRD，④处为 MPS，⑤处为 MRD，⑥处为 MPP，⑦处为 MPP。其中，②处的 MRD，③处的 MRD，⑦处的 MPP 全部取值为①处 MPS 存储进的 X0 的当前状态。⑤处为 MRD，⑥处为 MPP 取值为④处 MPS 存入的结果，即为 X0 与 X3 的状态相与的结果。

某段程序，如果使用了 MPS 指令，根据程序的实际情况，可以不使用 MRD 指令，但是必须在以下的程序中使用 MPP 指令来清除存入的运算结果，否则会导致程序错误。

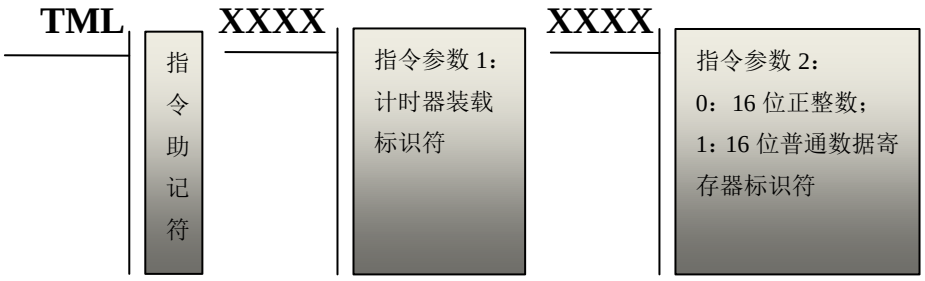
2.5 定时器操作指令

MOTEC 智能驱动器内置可编程控制器拥有 32 个 16 位长度的 1ms 递减定时器，最大定时器装载值为 65535。定时器操作指令就是对驱动器内部定时器的操作，主要是对定时器内部值的装载和清除。MOTEC 智能驱动器内置可编程控制器有两条指令可以对定时器进行操作，TML 指令和 TMC 指令。

TML：定时器装载初值指令

TML：将一个参数或者一个 16 位普通数据寄存器内的 16 位正整数装载到指定定时器装载寄存器中，并且如果该数据不为 0，就开始计时，相应的定时器状态继电器会变成 OFF 状态。

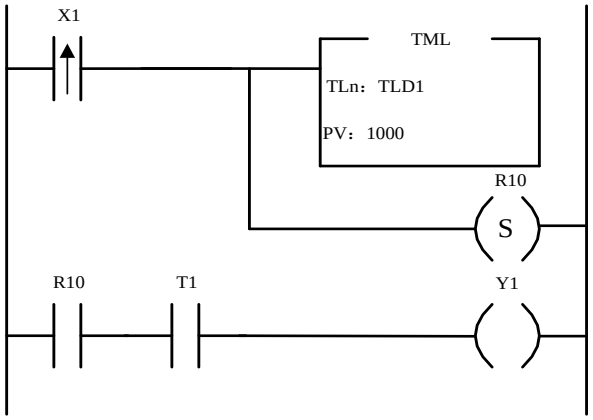
TML 指令的助记符语句结构为



例如在第一章 1.2.4 节中的举例的程序

例如

梯形图



助记符

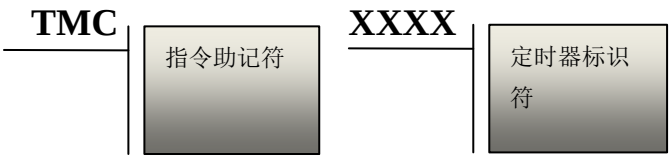
```
LD_R    X1
TML     FUNC
          0
        TLD1
        1000
SET     R10
LD      R10
AND     T1
OUT     Y1
```

注意：MOTEC 智能驱动器内置可编程控制器在上电以后，会自动将所有定时器的装载寄存器置 0，所有定时器的状态继电器为 OFF 状态。如果需要在程序处多次使用同一个定时器，需要设置一个辅助寄存器，来保证程序能够能够正确执行。如示例程序中的 R10，需要在定时器装载寄存器装入初值时将 R10 置位，并且根据程序需要，可以再加入 R10 清除的指令。

TMC：定时器清零指令

TMC：将指定的定时器装载寄存器的值清零，在清零之后，会触发一个定时完成，自动将该定时器的状态继电器置为 ON。

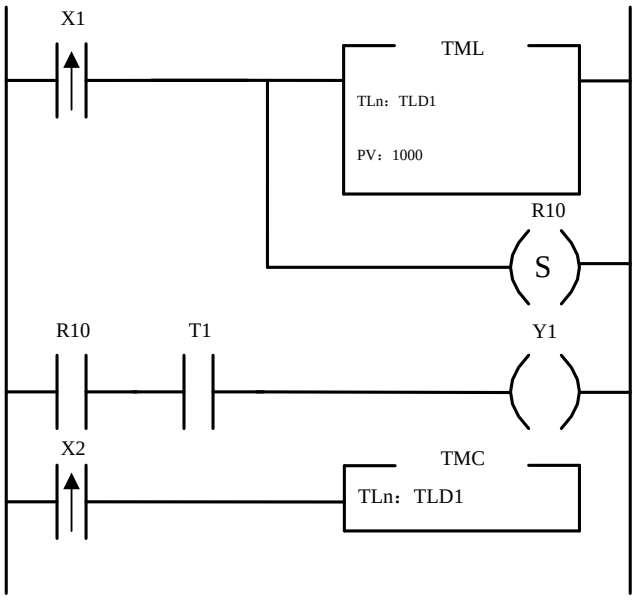
TMC 指令的助记符语句结构为



定时器清零指令会自动中断当前的定时。

例如：

梯形图



助记符

```
LD_R    X1
TML     FUNC
          0
        TLD1
        1000
SET     R10
LD      R10
AND     T1
OUT     Y1
LD_R    X2
TMC     FUNC
        TLD1
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X1	装载输入继电器 X1 的上升沿
TML	FUNC	将常数 1000 写入定时器 TLD1 的装载寄存器内
	0	
	TLD1	
	1000	
SET	R10	将辅助继电器 R10 置 ON
LD	R10	装载辅助继电器 R10
AND	T1	与定时器 T1 状态继电器 T1 相与
OUT	Y1	将当前的栈值输出到输出继电器 Y1
LD_R	X2	装载输入继电器 X2 的上升沿
TMC	FUNC	将定时器 L1 的装载寄存器清零
	L1	

程序说明：本程序使用定时器 L1，系统 X1 检测到上升沿，启动定时器 L1，并且置位辅助继电器 R10，等到定时器状态继电器 T1 为 ON 时，输出 ON 到输出继电器 Y1。程序功能是 X1 有上升沿时，过 1000ms 即 1s 以后会输出 ON 到 Y1。如果系统检测到 X2 的上升沿，就会将 L1 的定时器装载寄存器清零，会马上输出 ON 到 Y1。

注意：TMC 指令会自动中断当前定时。

2.6 计数器操作指令

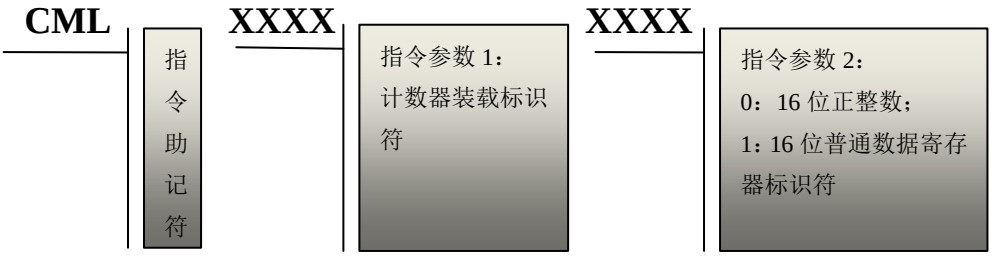
MOTEC 智能驱动器内置可编程控制器内部具有 32 个递减计数器。计数器装载寄存器是一个 16 位的寄存器，最大计数为 65535。MOTEC 智能驱动器内置可编程控制器有三条指令可以对计数器进行操作，CML 指令，CMS 指令，CMC 指令。

CML：计数器装载初值指令 ；CMS：计数器递减指令

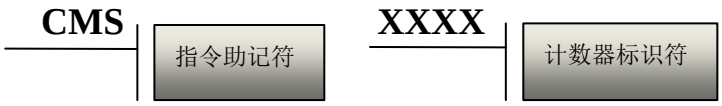
CML：将一个参数或者一个 16 位普通数据寄存器内的 16 位正整数的装载进指定计数器装载寄存器中，并且如果该数据不为 0，相应的计数器状态继电器会变成 OFF 状态。

CMS：启动指定计数器的递减操作，计数器与定时器不同，计数器只有在使用该指令以后才开始计数，并且每一次计数都要使用该指令。

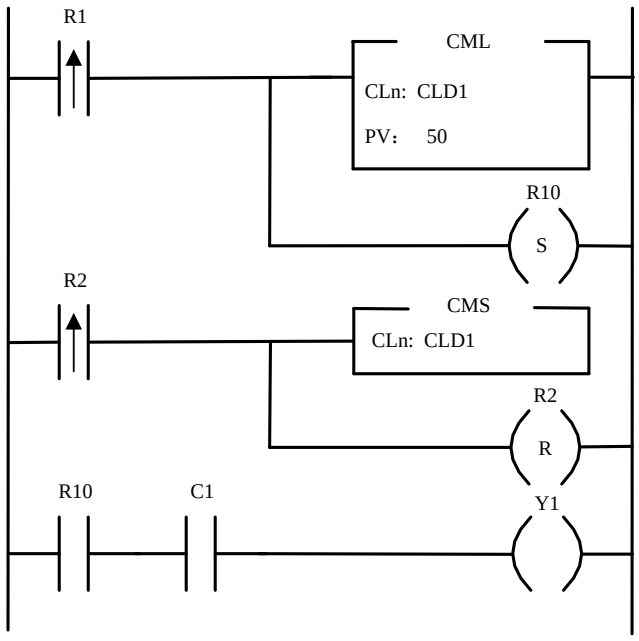
CML 指令的助记符语句结构为



CMS 指令的助记符语句结构为



例如在第一章 1.2.5 节中的举例的程序



助记符

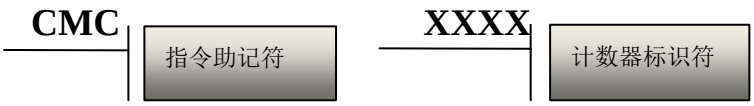
```
LD_R    R1
CML     FUNC
          0
        CLD1
        50
SET     R10
LD_R    R2
CMS     FUNC
        CLD1
RESET   R2
LD      R10
AND     C1
OUT     Y1
```

注意：MOTEC 智能驱动器内置可编程控制器在上电以后，会自动将所有计数器的装载寄存器置 0，所有计数器的状态继电器为 OFF 状态。对某个计数器的装载寄存器内装载非 0 的正整数数值之前，如果需要在程序处多次使用同一个计数器，需要设置一个辅助寄存器，来保证程序能够能够正确执行。如示例程序中的 R10，需要在定时器装载寄存器装入初值时将 R10 置位，并且根据程序需要，可以再加入 R10 清除的指令。

CMC：计数器清零指令

CMC：将指定的计数器装载寄存器的值清零，在清零之后，会触发一个定时完成，自动将该计数器的状态继电器置为 ON。

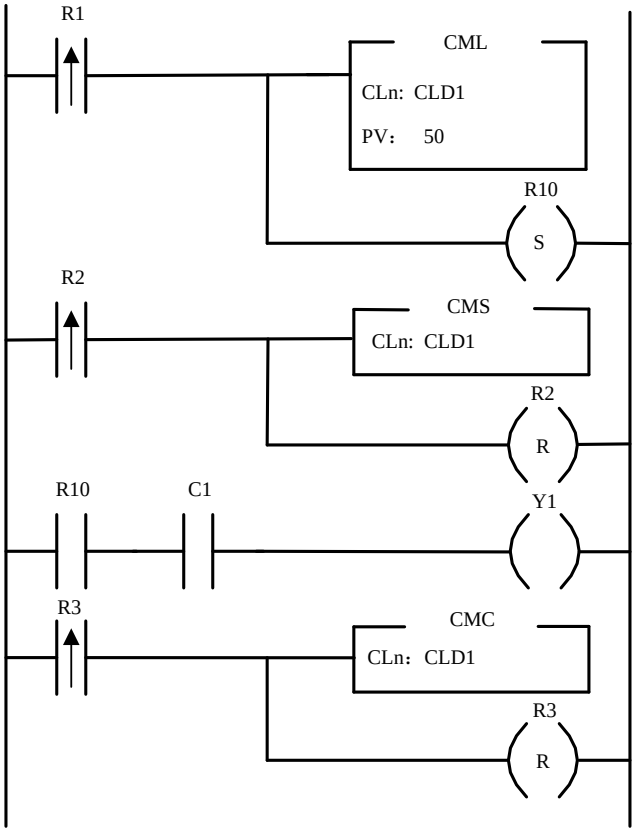
CMC 指令的助记符语句结构为



计数器清零指令会自动中断当前的定时。

例如：

梯形图



助记符

LD_R	R1
CML	FUNC
	0
	CLD1
	50
SET	R10
LD_R	R2
CMS	FUNC
	CLD1
RESET	R2
LD	R10
AND	C1
OUT	Y1
LD_R	R3
CMC	FUNC
	CLD1
RESET	R3

指令解释（助记符部分）

指令内容		指令说明
LD_R	R1	装载输入继电器 R1 的上升沿
CML	FUNC	向计数器 CLD1 的装载寄存器内写入常
	0	数 50
	CLD1	
	50	
SET	R10	将辅助继电器 R10 置位为 ON
LD_R	R2	装载输入继电器 R2 的上升沿
CMS	FUNC	计数器 CLD1 的装载寄存器内的数值减 1
	CLD1	
RESET	R2	清除继电器 R2
LD	R10	装载辅助继电器 R10 的状态
AND	C1	当前栈值与计数器 C1 的状态继电器相与
OUT	Y1	将当前栈值输出到输出继电器 Y1
LD_R	R3	装载输入继电器 R3 的上升沿
CMC	FUNC	将计数器 CLD1 的装载寄存器清零
	CLD1	
RESET	R3	清除继电器 R3

程序说明：本程序使用计数器 CLD1，输入继电器 R1 的上升沿会触发计数器 CLD1 的初始值写入，输入继电器 R2 的上升沿会触发计数器 CLD1 的递减计数，直到输入继电器 R2 的上升沿达到 50 次的时候，会输出 ON 到输出继电器 Y1。如果系统检测到 R3 的上升沿，就会将 CLD1 的计数器装载寄存器清零，会马上输出 ON 到 Y1。注意：TMC 指令会自动中断当前定时。

2.7 程序控制指令

程序控制指令包括跳转到某一行程序，跳转到子程序，子程序返回，程序结束等指令。

JUMP/ JUMP_NOT: 条件满足跳转指令/条件不满足跳转指令

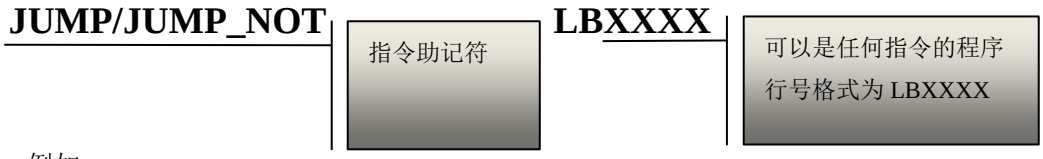
LB:程序标号

JUMP：当条件满足时将当前程序跳转到指定的程序标号处。

JUMP_NOT：当条件不满足时将当前程序跳转到指定的程序标号处。

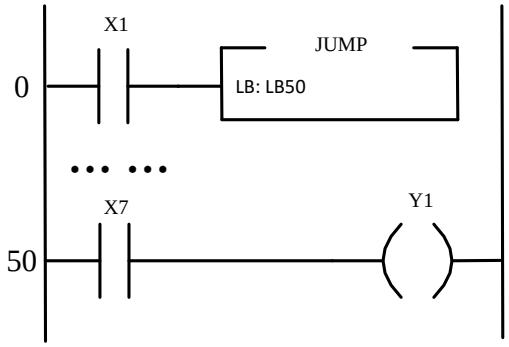
LB:程序标号，没有指令参与，只是确定当前的指令在程序中的位置。LB 指令是程序确定以后，从母线开始的指令的地址，程序行号。

JUMP / JUMP_NOT 指令的助记符语句结构为



例如：

梯形图



助记符

```
LD          X1
JUMP        FUNC
            LB50

……

LD          X7
OUT         Y1
```

指令解释（助记符部分）

指令内容		指令说明
LD	X1	装载继电器 X1 的状态
JUMP	FUNC LB50	跳转到第 50 行指令
……		
LD	X7	装载 X7 的状态
OUT	Y1	输出到 Y1

程序说明：当 X1 为 ON 时，跳转到行号为 50 的程序处，装载 X7 的状态，输出到 Y1 中。

注意：如果某段程序打算作为程序跳转的目的程序，必须将该处程序从母线开始，否则将不能正确的跳转。

如果仅仅使用单次跳转，请使用沿操作。JUMP_NOT 指令与 JUMP 指令相反。

CALL/ CALL_NOT:条件满足调用子程序/条件不满足调用子程序

SUB：子程序标号

RETURN：子程序返回

CALL：条件满足时，会中断当前程序运行，自动跳转到指定子程序。

CALL_NOT：条件不满足时，会中断当前程序运行，自动跳转到指定子程序。

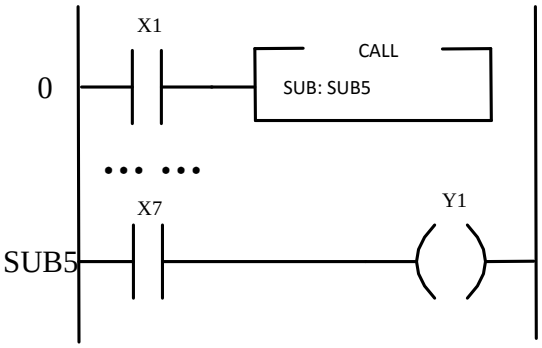
SUB：子程序标号，在进入子程序的时候使用。

RETURN：用于从子程序返回到主程序。

CALL/CALL_NOT 指令的助记符语句结构为

例如：

梯形图 助记符




```
LD          X1
CALL_FUN    FUNC
            SUB5

……

LD          X7
OUT         Y1
RETURN
```

指令解释（助记符部分）

指令内容		指令说明
LD	X1	装载继电器 X1 的状态
FUNC	CALL_FUN	进入子程序 SUB5 中
	SUB5	
……		
SUB5		子程序 SUB5 起始标识
LD	X7	装载 X7 的值
OUT	Y1	输出到 Y1
RETURN		返回

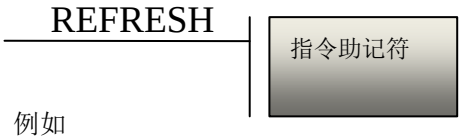
注意：RETURN 子程序返回指令无需再加子程序里，PLC 软件子程序执行完后会自动返回。在编程软件中，子单元是以序号 1 开始，在程序编写中，子程序序号是以 SUB0 开始，请注意。

CALL_NOT 与 CALL 指令相反。

REFRESH：立即刷新继电器

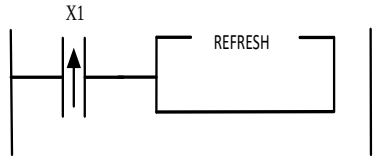
REFRESH ：该指令刷新所有辅助继电器。

REFRESH 指令的助记符语句结构为



例如

梯形图



助记符

LD_R	X1
REFRESH_XYTC	FUNC

指令解释（助记符部分）

指令内容		指令说明
LD_R	X1	检测 X1 上升沿
REFRESH_XYTC	FUNC	刷新继电器

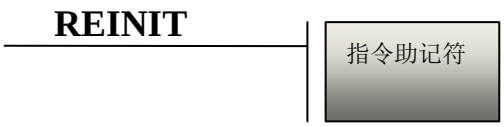
程序说明：该程序检测 X1 的上升沿，触发将刷新所有继电器，包括输入输出继电器，定时器状态继电器和计数器状态继电器。

使用限制：REFRESH_XYTC 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

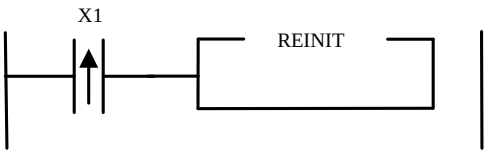
REINIT：重新初始化 PLC

REINIT：该指令将继电器和寄存器初始化。

REINIT 指令的助记符语句结构为



例如：梯形图



助记符

LD_R	X1
PLC_REINIT	FUNC

指令解释（助记符部分）

指令内容		指令说明
LD_R	X1	检测 X1 上升沿
PLC_REINIT	FUNC	PLC 初始化

程序说明：该程序检测 X1 的上升沿，触发将继电器、寄存器初始化，恢复到刚开始上电的状态。

使用限制：INTIT_START 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

注意：初始化 PLC 后，会将当前的栈值清零，除了特殊辅助继电器和特殊数据寄存器以及驱动器内部寄存器 P 之外，其余所有辅助继电器 R 和数据寄存器 D，定时器装载寄存器 TLD，计数器装载寄存器 CLD,定时器状态继电器 T,计数器状态继电器 C 都会被复位成 OFF 或者清零。用户需要谨慎使用该指令。

第 3 章 高级指令

高级指令的控制的操作单位是寄存器。可以进行 16 位数据或者更复杂数据格式的操作，本章主要讲基本指令的语句结构和使用方式。

3.1 数据操作比较指令

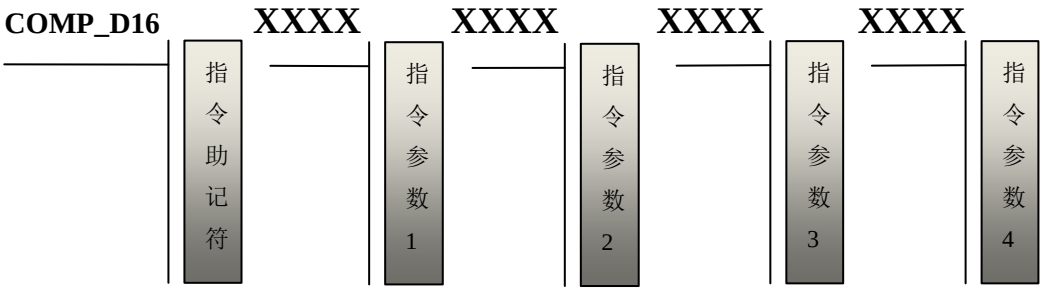
数据操作比较分为 16 位数据操作比较和 32 位数据操作比较。

3.1.1 16 位数据比较指令

COMP_D16:将两个 16 位整型数据进行比较

COMP_D16：该指令将两个 16 位数据进行比较，可判断两个 16 位数据相等操作/大于等于操作/小于等于操作/大于操作/小于操作/不等于操作，且可进行有符号和无符号整型比较选择。

COMP_D16 的指令的助记符语句结构为



指令参数 1：指令参数 1 表示有符号或无符号整型数据

指令参数 2：指令参数 2 表示 D 寄存器地址

指令参数 3：指令参数 3 表示 D 寄存器地址或常数

指令参数 4：指令参数 4 表示输出辅助继电器(R 继电器或 Y 继电器)

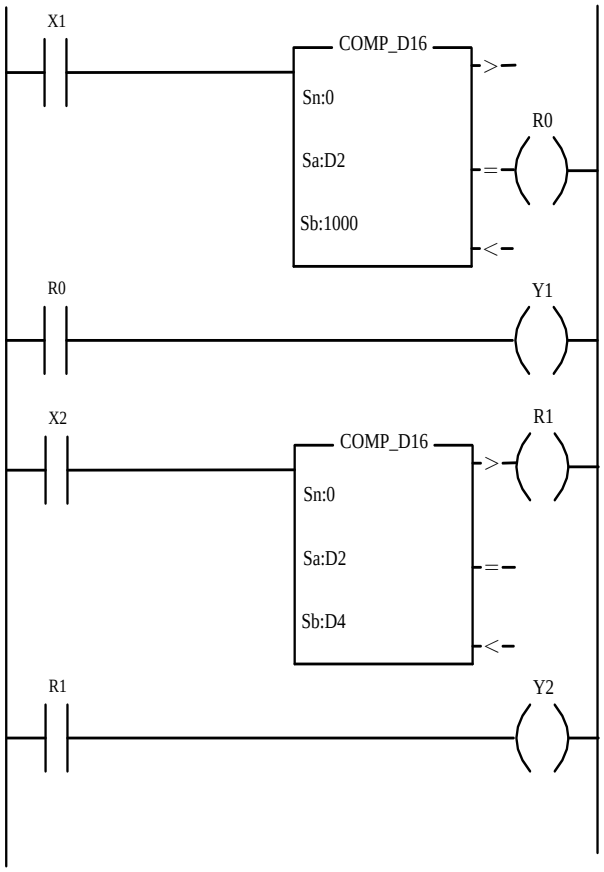
以 COMP_D16 指令为例，列出下列比较图表

参数 1	参数 2	参数 3	参数 4	判断条件	输出
无符号/有符号	16 位 D 寄存器地址（Dm）	16 位 D 寄存器地址（Dn）	结果输出继电器(R 或 Y)	Dm>Dn	“>”端置 ON
				Dm=Dn	“=”端置 ON
				Dm<Dn	“<”端置 ON
无符号/有符号	16 为 D 寄存器地址（Dm）	常数（K）	结果输出继电器（R 或 Y）	Dm > k	“>”端置 ON
				Dm = k	“=”端置 ON
				Dm < k	“<”端置 ON

——n、m 代指标识号，实际应用中为数字

例如

梯形图



助记符

LD	X1
DATA_16_COMPARE	FUNC
	0
	0
	D2
	1000
	R0
LD	R0
OUT	Y1
LD	X2
DATA_16_COMPARE	FUNC
	1
	0
	D2
	D4
	R1
LD	R1
OUT	Y2

指令解释(助记符部分)

指令内容		指令说明
LD	X1	装载 X1 的状态
DATA_16_COMPARE	FUNC	寄存器 D2 中的内容和常数 1000 做是否
	0	相等的比较，比较结果输出继电器 R0
	0	
	D2	
	1000	
	R0	
LD	R0	装载 R0 的状态
OUT	Y1	将当前栈值输出到 Y1
LD	X2	装载 X2 的状态
DATA_16_COMPARE	FUNC	D2 中的内容和 D4 中的内容做是否大于
	1	的比较，比较结果输出继电器 R1
	0	
	D2	
	D4	
	R1	
LD	R1	装载 R1 的状态
OUT	Y2	将当前栈值输出到 Y2

程序说明：该程序当 X1 为 ON 时，就会判断 D2 存放的数据是不是等于 1000，并通过辅助继电器 R0 输出到 Y1；当 X2 为 ON 时，就会判断 D2 存放的数据是不是大于 D4 中存放的数据，并通过辅助继电器 R1 输出到 Y2。

假设 D2 中存放的 16 位有符号数是-1000，D4 中存放的 16 位有符号数为-1200，那么 X1 和 X2 都为 ON 时，

-1000 ≠ 1000 ； Y1 输出 OFF；

-1000 > -1200 ； Y2 输出 ON；

使用限制 COMP_D16 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器。助记符中比较指令的 0 或 1 指的是比较的双方对象，无太大的意义。

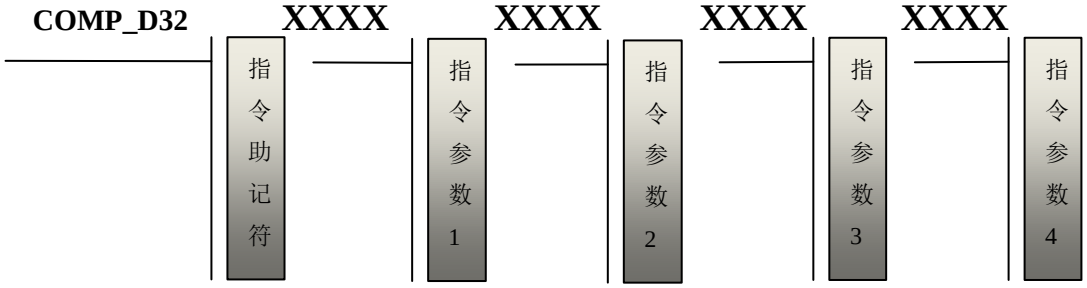
注意：该指令的输出只有大于、等于、和小于三种，如果使用大于等于，请在判断小于输出继电器的取反，如果使用小于等于，请在判断大于输出继电器的取反，如果使用不等于，请在判断等于输出继电器的取反。

3.1.2 32 位数据比较指令

COMP_D32:将两个 32 位整型数据进行比较

COMP_D32：该指令将两个 32 位数据进行比较，可判断两个 32 位数据相等操作/大于等于操作/小于等于操作/大于操作/小于操作/不等于操作，且可进行有符号和无符号整型比较选择。两个连续的 16 位数据寄存器组成的一个 32 位数据。

COMP_D32 的指令的助记符语句结构为



指令参数 1：指令参数 1 表示有符号或无符号整型数据

指令参数 2：指令参数 2 表示 32 位数据的低 16 位 D 寄存器地址

指令参数 3：指令参数 3 表示 32 位数据的低 16 位 D 寄存器地址或常数

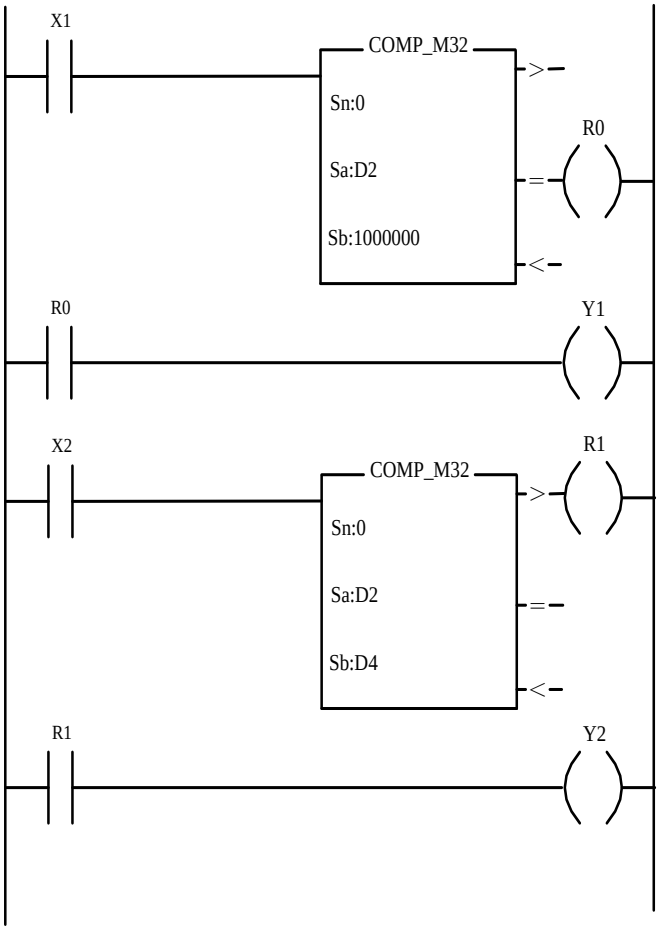
指令参数 4：指令参数 4 表示输出辅助继电器(R 继电器或 Y 继电器)

以 **COMP_D32** 指令为例，列出下列比较图表

参数 1	参数 2	参数 3	参数 4	判断条件	输出
无符号/有符号	32 位 D 寄存器地址（Dn）	32 为 D 寄存器地址（Dm）	结果输出继电器（R 或 Y）	Dm>Dn	“>”端置 ON
				Dm=Dn	“=”端置 ON
				Dm<Dn	“<”端置 ON
无符号/有符号	32 位 D 寄存器地址（Dn）	常数（K）	结果输出继电器（R 或 Y）	Dm ＞ k	“>”端置 ON
				Dm ＝ k	“=”端置 ON
				Dm ＜ k	“<”端置 ON

——n、m 代指标识号，实际应用中为数字

例如：梯形图



助记符

```
LD                  X1
DATA_32_COMPARE     FUNC
                    0
                    0
                    D2
                    1000000
                    R0
LD                  R0
OUT                 Y1
LD                  X2
DATA_32_COMPARE     FUNC
                    1
                    0
                    D2
                    D4
                    R1
LD                  R1
OUY                 Y2
```

指令解释(助记符部分)

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
DATA_32_COMPARE	FUNC	D2、D3 组成的 32 位数据和 1000000 做
	0	是否相等的比较,比较结果输出到辅助继
	0	电器 R0
	D2	
	1000000	
	R0	
LD	R0	装载辅助继电器 R0 的状态
OUT	Y1	将当前栈值输出到 Y1
LD	X2	装载输入继电器 X2 的状态
DATA_32_COMPARE	FUNC	D2、D3 组成的 32 位数据和 D4、D5 组成
	1	的 32 位数据做是否大于比较，比较结果
	0	输出到辅助继电器 R1
	D2	
	D4	
	R1	装载继电器 R1 的状态
LD	R1	
OUY	Y2	将当前栈值输出到 Y2

程序说明：该程序当 X1 为 ON 时，就会判断 D2、D3 组成的 32 位数据是不是等于 1000000，并通过辅助继电器 R0 输出到 Y1；当 X2 为 ON 时，就会判断 D2、D3 组成的 32 位数据是不是大于 D4、D5 组成的 32 位数据，并通过辅助继电器 R1 输出到 Y2。

假设 D2、D3 组成的 32 位有符号数据是-1000000，D4、D5 组成的 32 位有符号数据为-1200000，那么 X1 和 X2 都为 ON 时，

-1000000 ≠ 1000000； Y1 输出 OFF；

-1000000 > -1200000 ； Y2 输出 ON；

使用限制：COMP_D32 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且 32 位数据比较指令的操作对象只能是 D 寄存器，不能是 P 寄存器。

注意：该指令的输出只有大于、等于、和小于三种，如果使用大于等于，请在判断小于输出继电器的取反，如果使用小于等于，请在判断大于输出继电器的取反，如果使用不等于，请在判断等于输出继电器的取反。

3.2 数据赋值转移指令

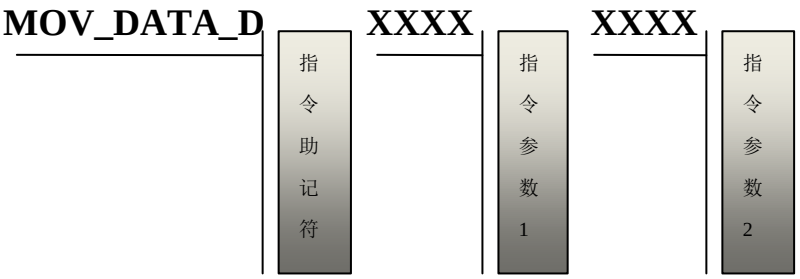
数据赋值指令可以将数据在常数、普通 16 位数据寄存器、特殊数据寄存器，驱动器内部数据寄存器之间相互传输，可以传输 16 位数据和 32 位数据。

3.2.1 16 位数据赋值转移指令

MOV_DATA_D：将 16 位数据赋值给 16 位 D 寄存器内

MOV_DATA_D：该指令可以将 16 位常数或者 16 位数据寄存器内的数据传输给 D 寄存器。

MOV_DATA_D 指令的助记符语句结构为



指令参数 1 表示传输对象的数据：16 位常数或普通数据寄存器的地址 Dn；

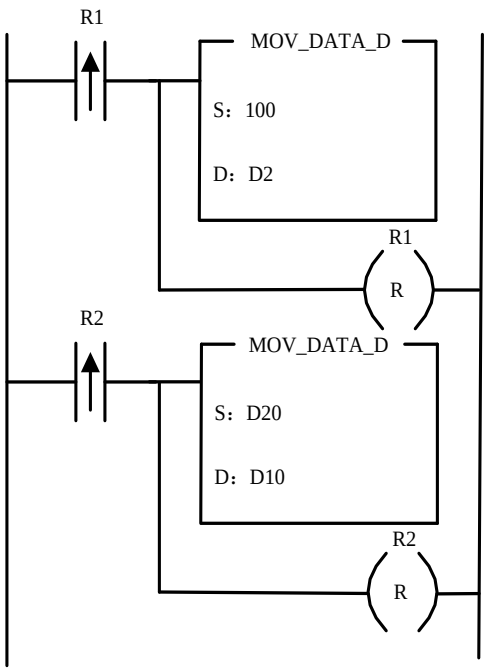
指令参数 2 表述传输到的目标寄存器地址，为 Dm

具体见下表

参数 1	参数 2	指令操作
16 位常数 k	16 位 D 寄存器地址（Dm）	Dm = k
16 位 D 寄存器地址（Dn）		Dm = Dn

例如

梯形图



助记符

```
LD_R          R1
MOV_DATA_D    FUNC
              0
              100
              D2
RESET         R1
LD_R          R2
MOV_DATA_D    FUNC
              1
              D20
              D10
RESET         R2
```

指令解释(助记符部分)：

指令内容		指令说明
LD_R	R1	检测继电器 R1 的上升沿
MOV_DATA_D	FUNC	将常数 100 赋值到寄存器 D2
	0	
	100	
	D2	
RESET	R1	清除继电器 R1
LD_R	R2	检测继电器 R2 的上升沿
MOV_DATA_D	FUNC	将寄存器 D20 的内容赋值到寄存器 D10
	1	
	D20	
	D10	
RESET	R2	清除继电器 R2

程序说明：该程序检测 R1 有上升沿到来，将 16 位常数 100 赋值到普通数据寄存器 D2 中，R2 有上升沿，就将普通数据寄存器 D20 的内容赋值给普通数据寄存器 D10。运算结果为 D2 = 100，D10 = D20;

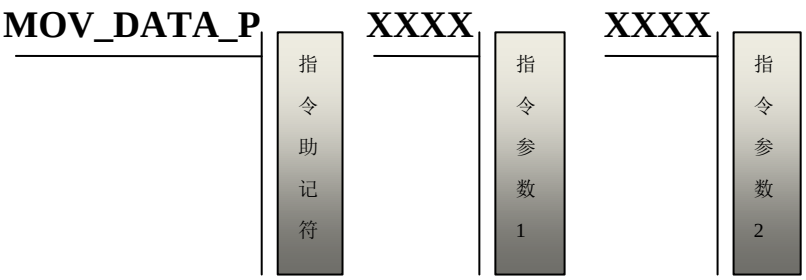
使用 MOV_DATA_D 可以将 16 位数据灵活的在普通数据寄存器内相互赋值。

使用限制：MOV_DATA_D 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

MOV_DATA_P：将 16 位常数赋值给 16 位 P 寄存器内

MOV_DATA_P：该指令可以将 16 位常数传输给 P 寄存器，操作的对象只能是常数。

MOV_DATA_P 指令的助记符语句结构为



指令参数 1 为 16 位常数

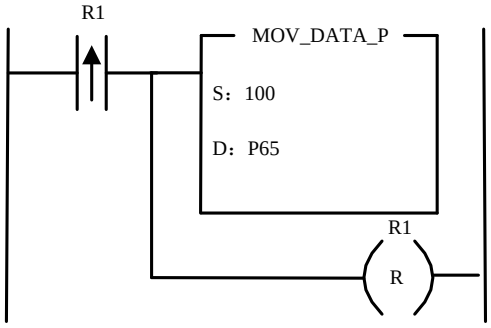
指令参数 2 表示传输的目标 P 寄存器的地址

具体见下表

参数 1	参数 2	指令操作
16 位常数 k	16 位 P 寄存器地址（Pm）	Pm = k

例如

梯形图 助记符




```
LD_R          R1
MOV_DATA_P    FUNC
              100
              P65
RESET         R1
```

指令解释(助记符部分)：

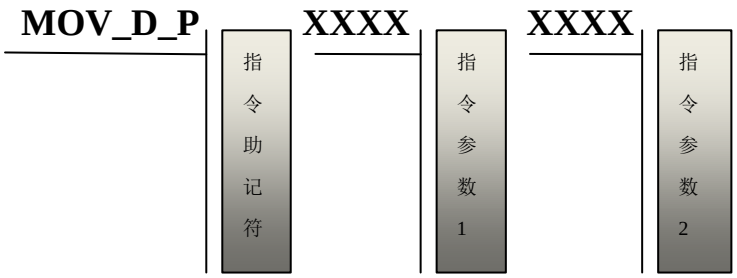
指令内容		指令说明
LD_R	R1	检测继电器 R1 的上升沿
MOV_DATA_P	FUNC	将常数 100 赋值到寄存器 P65
	100	
	P65	
RESET	R1	清除继电器 R1

程序说明：该程序检测 R1 有上升沿到来，将 16 位常数 100 赋值到驱动器内部数据寄存器 P65 中， P65 = 100。用户可以通过该指令对驱动器内部的控制参数做修改。

MOV_D_P：普通数据寄存器 D 与驱动器内部数据寄存器 P 之间数据转移

MOV_D_P：该指令可以将数据在 D 寄存器与 P 寄存器之间相互转移。

MOV_D_P 指令的助记符语句结构为



指令参数 1 表示 D 寄存器地址；

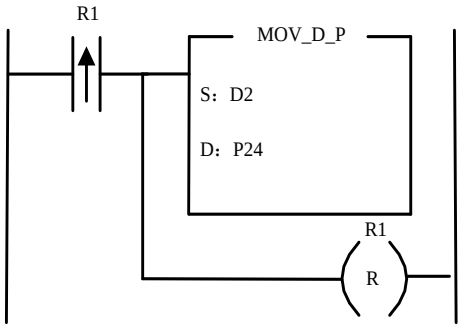
指令参数 2 表示 P 寄存器地址；

具体见下表

参数 1	参数 2	指令操作
16 位 D 寄存器地址（Dn）	P 寄存器地址（Pm）	Pm = Dn

例如

梯形图



助记符

```
LD_R        R1
MOV_D_P     FUNC
            0
            D2
            P24
RESET       R1
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测继电器 R1 的上升沿
MOV_D_P	FUNC	将寄存器 D2 的值存储到 P24 寄存器中
	0	
	D2	
	P24	
RESET	R1	清除继电器 R1

程序说明：检测 R1 有上升沿到来，将普通数据寄存器 D2 里面的数据赋值到驱动器内部数据寄存器 P24 中。用户可以通过该指令对驱动器内部的控制参数做修改。

注意：由于驱动器内部寄存器有很多驱动器自己预定义的参数，该指令可以方便的读取驱动器内部的寄存器数据参与到相关的运算或者比较。

MOV_P_D：驱动器内部数据寄存器 P 与普通数据寄存器 D 之间数据转移

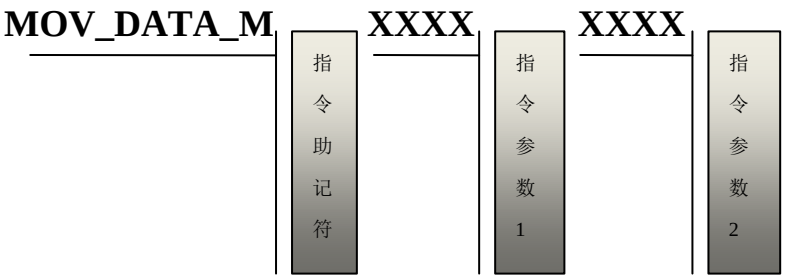
这条指令与上述 MOV_D_P 恰恰相反，这里就不再详细说明了。

3.2.2 32 位数据赋值转移指令

MOV_DATA_M：将 32 位数据赋值给 32 位 D 寄存器内

MOV_DATA_M：该指令可以将 32 位常数或者由两个连续的 16 位数据寄存器组成的 32 位数据内的数据传输给两个连续的 16 位 D 寄存器。

MOV_DATA_M 指令的助记符语句结构为



指令参数 1 表示传输的对象：32 位常数或两个连续的普通数据寄存器的低 16 位的地址 Dn；

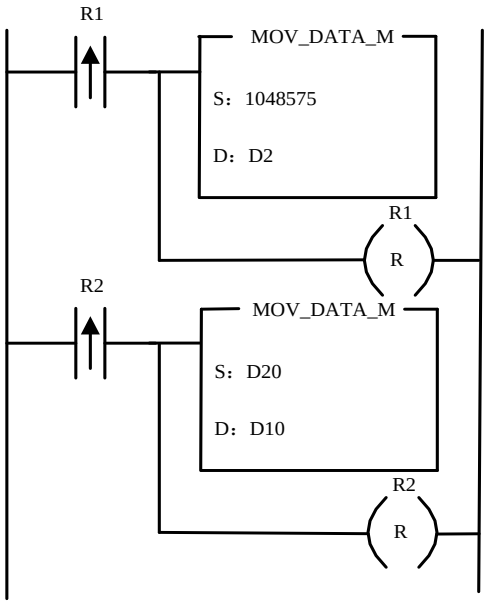
指令参数 2 表述传输到的目标寄存器地址为两个连续的普通数据寄存器的低 16 位地址 Dm

具体见下表

参数 1	参数 2	指令操作
32 位常数 k	32 位的低 16 位 D 寄存器地址（Dm）	Dm = k 的低 16 位
		Dm+1 = k 的高 16 位
32 位的低 16 位 D 寄存器地址（Dn）		Dm = Dn
		Dm+1 = Dn+1

例如：

梯形图



助记符

```
LD_R          R1
MOV_DATA_M    FUNC
              0
              1048575
              D2
RESET         R1
LD_R          R2
MOV_DATA_M    FUNC
              1
              D20
              D10
RESET         R2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测继电器 R1 的上升沿
MOV_DATA_M	FUNC	将常数 1048575 赋值到寄存器 D2
	0	
	1048575	
	D2	
RESET	R1	清除继电器 R1 的状态
LD_R	R2	检测继电器 R2 的上升沿
MOV_DATA_M	FUNC	将寄存器 D20 的内容赋值到寄存器 D10
	1	
	D20	
	D10	
RESET	R2	清除继电器 R2 状态

程序说明：该程序检测 R1 有上升沿到来，将 32 位常数 1048575 赋值到普通数据寄存器 D2 和 D3 中，R2 有上升沿，就将普通数据寄存器 D20 的内容赋值给普通数据寄存器 D10。D21 的内容赋值给 D11，运算结果为D2 = 65535，D3 = 15，D10 =D20，D11 = D21。

使用限制：MOV_DATA_M 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。用户不能通过 32 位数据的指令对 P 寄存器进行操作，如果需要操作 P 寄存器的 32 位数据，请通过 D 寄存器操作，结束之后在转移到 P 寄存器中。

3.3 数据移位指令

数据移位是指以位为单位，整体数据左移或者右移一个数据位，移位指令包括单独移位和循环移位两种，操作的数据可以是 16 位数据或者 32 位数据。

注意：单独左移或者单独右移是相对于循环左移或者循环右移来讲。

3.3.1 16 位数据单独移位指令

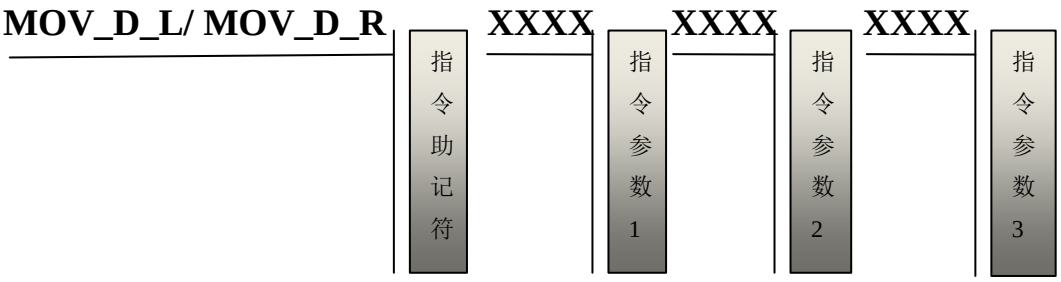
16 位数据单独移位指令包括单独左移和单独右移指令，可以将一个普通数据寄存器 D 存放的 16 位数据进行单独左移或者单独右移操作，并且存放到一个目标普通寄存器 D 内。

MOV_D_L/MOV_D_R：16 位数据单独左移 n 位/16 位数据单独右移 n 位

MOV_D_L：该指令可以将一个普通数据寄存器 Dn 内的数据单独左移特定的位数，并且存储到指定的普通数据寄存器 Dm 内，该操作不能循环，移到最高位以外的数据将被舍弃，移出的低位由 0 补齐。

MOV_D_R：该指令可以将一个普通数据寄存器 Dn 内的数据单独右移特定的位数，并且存储到指定的普通数据寄存器 Dm 内。该操作不能循环，移到最低位以外的数据将被舍弃，移出的高位由 0 补齐。

MOV_D_L/ MOV_D_R 指令的助记符语句结构为



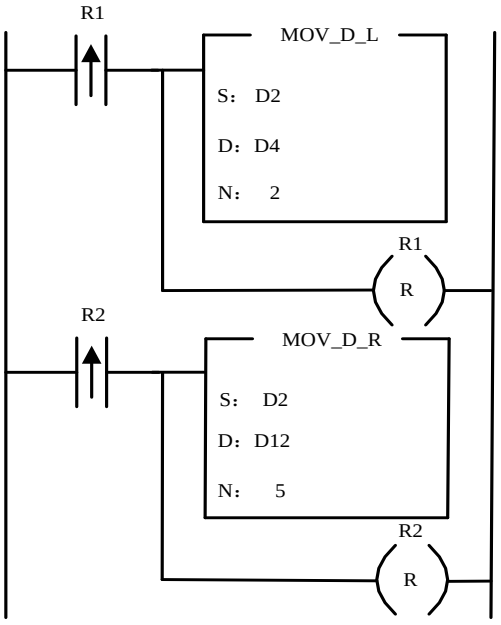
指令参数 1 表示需要移位的数据寄存器的地址 Dn；

指令参数 2 表示移位之后存储的目标数据寄存器的地址 Dm；

指令参数 3 表示移位的个数；

例如

梯形图



助记符

```
LD_R        R1
MOV_D_L     FUNC
            D2
            D4
            2
RESET       R1
LD_R        R2
MOV_D_R     FUNC
            D2
            D12
            5
RESET       R2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测继电器 R1 的上升沿
MOV_D_L	FUNC	将 D2 中的数据单独左移 2 位并将结果存入 D4 中
	D2	
	D4	
	2	
RESET	R1	清除继电器 R1
LD_R	R2	检测继电器 R2 的上升沿
MOV_D_R	FUNC	将 D2 中的数据单独右移 5 位并将结果存入 D12 中
	D2	
	D12	
	5	
RESET	R2	清除继电器 R2

程序说明：该程序可以检测 R1 的上升沿，触发将 D2 寄存器里的数据单独左移 2 位存入 D4 中，检测 R2 的上升沿，触发将 D2 寄存器里的数据单独右移 5 位到 D12 中。

例如下表

假设 D2 存放的数据为

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	0	0	1	0	1	1	1	1	0

对于程序中的左移指令，经过左移两位以后，将结果存放到 D4 寄存器内，那么 D4 寄存器内的数据为

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	0	1	1	1	1	0	0	0

其中，第 14 位和第 15 位的数据被舍弃，第 0 位和第 1 位的数据由 0 补齐。

对于程序中的右移指令，经过右移五位以后，将结果存放到 D12 寄存器内，那么 D12 寄存器内的数据为

D12

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	0	1	0	0	1	0

其中，第 0 位数据到第 4 位数据被舍弃，第 11 位数据到第 15 位数据由 0 补齐。

使用限制：MOV_D_L/ MOV_D_R 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

注意：该指令中，指令参数 3 的取值范围是 0～15，如果该值大于 15，将会只取该值的低 4 位数据。

3.3.2 16 位数据循环移位指令

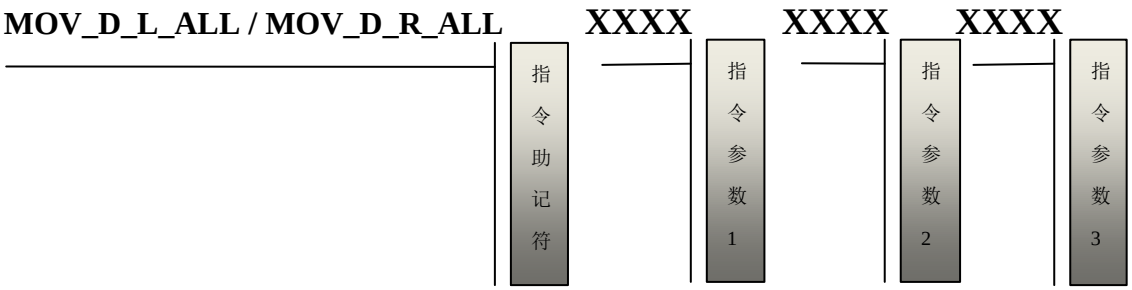
16 位数据循环移位指令包括循环左移和循环右移指令，可以将一个普通数据寄存器 D 存放的 16 位数据进行循环左移或者循环右移操作，并且存放到一个目标普通寄存器 D 内。

MOV_D_L_ALL/MOV_D_R_ALL：16 位数据循环左移 n 位/16 位数据循环右移 n 位

MOV_D_L_ALL：该指令可以将一个普通数据寄存器 Dn 内的数据循环左移特定的位数，并且存储到指定的普通数据寄存器 Dm 内，移到最高位以外的数据 将被补充到移出的低位中。

MOV_D_R_ALL：该指令可以将一个普通数据寄存器 Dn 内的数据循环右移特定的位数，并且存储到指定的普通数据寄存器 Dm 内，移到最低位以外的数据将被补充到移出的高位中。

MOV_D_L_ALL / MOV_D_R_ALL 指令的助记符语句结构为



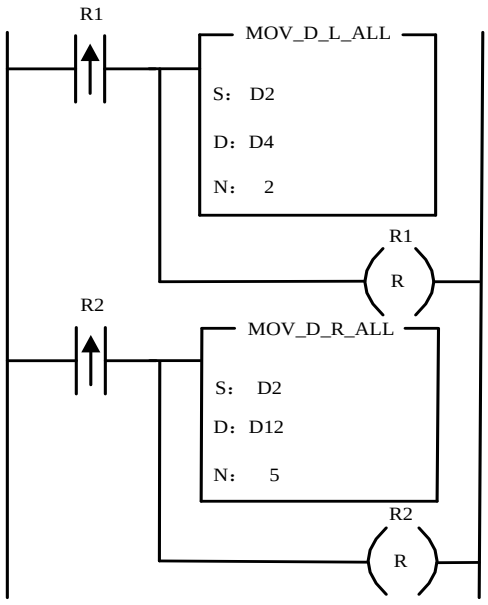
指令参数 1 表示需要移位的数据寄存器的地址 Dn；

指令参数 2 表示移位之后存储的目标数据寄存器的地址 Dm；

指令参数 3 表示移位的个数；

例如

梯形图




```
LD_R          R1
MOV_D_L_ALL   FUNC
              D2
              D4
              2
RESET         R1
LD_R          R2
MOV_D_R_ALL   FUNC
              D2
              D12
              5
RESET         R2
```

助记符

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测继电器 R1 的上升沿
MOV_D_L_ALL	FUNC	将寄存器 D2 中的数据循环左移 2 位并将结果
	D2	存入普通数据寄存器 D4 中
	D4	
	2	
RESET	R1	清除继电器 R1
LD_R	R2	检测继电器 R2 的上升沿
MOV_D_R_ALL	FUNC	将寄存器 D2 中的数据循环右移 5 位并将结果
	D2	存入寄存器 D12 中
	D12	
	5	
RESET	R2	清除继电器 R2

程序说明：该程序可以检测 R1 的上升沿，触发将 D2 寄存器里的数据循环左移 2 位存入 D4 中，检测 R2 的上升沿，触发将 D2 寄存器里的数据循环右移 5 位到 D12 中。

例如下表

假设 D2 存放的数据为

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	0	0	1	0	1	1	1	1	0

对于程序中的左移指令，经过左移两位以后，将结果存放到 D4 寄存器内，那么 D4 寄存器内的数据为

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	0	1	1	1	1	0	1	0

其中，第 14 位和第 15 位的数据被循环移到第 0 位和第 1 位。

对于程序中的右移指令，经过右移五位以后，将结果存放到 D12 寄存器内，那么 D12 寄存器内的数据为

D12

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	0	1	0	0	1	0

其中，第 0 位数据到第 4 位数据被循环移到第 11 位数据到第 15 位。

使用限制：MOV_D_L_ALL / MOV_D_R_ALL 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

注意：该指令中，指令参数 3 的范围取值是 0～15，如果该值大于 15，将会只取该值的低 4 位数据。

3.3.3 32 位数据单独移位指令

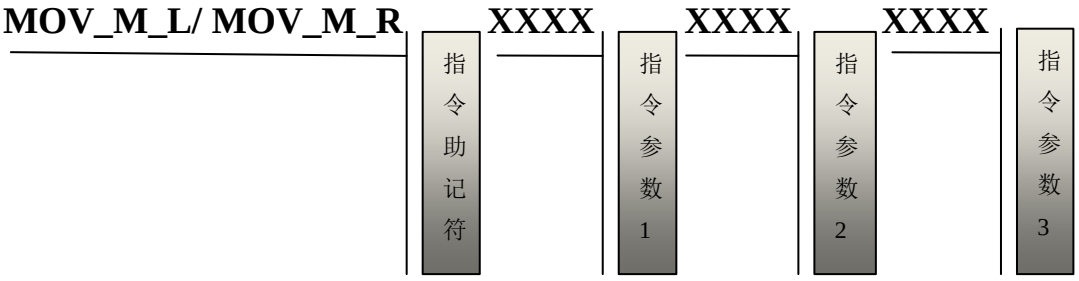
32 位数据单独移位指令包括单独左移和单独右移指令，可以将两个普通数据寄存器 D 组成的一个 32 位数据寄存器存放的 32 位数据进行单独左移或者单独右移操作，并且按照低位在前，高位在后的原则存放到两个目标普通寄存器 D 内。

MOV_M_L/MOV_M_R：32 位数据单独左移 n 位/32 位数据单独右移 n 位

MOV_M_L：该指令可以将两个普通数据寄存器 Dn 和 Dn+1 内的数据单独左移特定的位数，并且存储到指定的普通数据寄存器 Dm 和 Dm+1 内，该操作不能循环，移到最高位以外的数据将被舍弃，移出的低位由 0 补齐。

MOV_M_R：该指令可以将两个普通数据寄存器 Dn 和 Dn+1 内的数据单独右移特定的位数，并且存储到指定的普通数据寄存器 Dm 和 Dm+1 内，该操作不能循环，移到最低位以外的数据将被舍弃，移出的高位由 0 补齐

MOV_M_L/ MOV_M_R 指令的助记符语句结构为



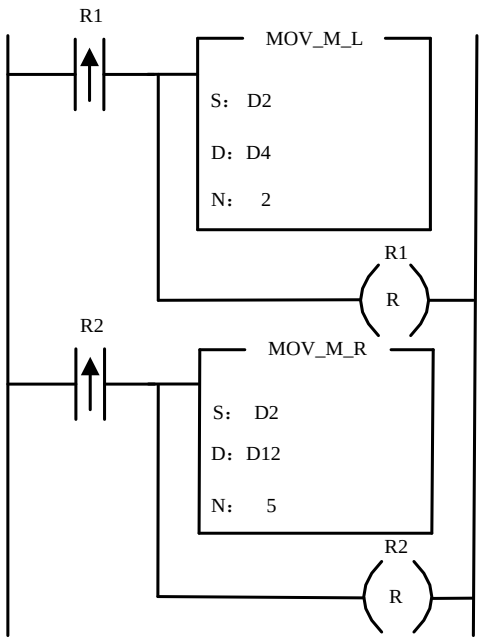
指令参数 1 表示需要移位的 32 位数据寄存器的低 16 位地址 Dn；

指令参数 2 表示移位之后存储的 32 位目标数据寄存器的低 16 位地址 Dm；

指令参数 3 表示移位的个数；

例如

梯形图



助记符

LD_R	R1
MOV_M_L	FUNC
	D2
	D4
	2
RESET	R1
LD_R	R2
MOV_M_R	FUNC
	D2
	D12
	5
RESET	R2

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测辅助继电器 R1 的上升沿
MOV_M_L	FUNC	将寄存器 D2 和 D3 组成的 32 位数据单独左移
	D2	2 位，并将结果存入寄存器 D4 和 D5 中
	D4	
	2	
RESET	R1	清除继电器 R1
LD_R	R2	检测继电器 R2 的上升沿
MOV_M_R	FUNC	将寄存器 D2 和 D3 组成的 32 位数据单独右移
	D2	5 位并将结果存入寄存器 D12 和 D13 中
	D12	
	5	
RESET	R2	清除继电器 R2

例如下表

假设 D2 和 D3 存放的数据为

D3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

——D2 作为 32 位数据的低 16 位，D3 作为 32 位数据的高 16 位

对于程序中的左移指令，经过左移两位以后，将结果存放到 D4 和 D5 寄存器内，那么 D4 和 D5 寄存器内的数据为

D5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	0	0	0	1	0	1	1	0

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	0	0	0	1	0	1	0	0

——D4 作为 32 位数据的低 16 位，D5 作为 32 位数据的高 16 位

其中，D3 的第 14 位和第 15 位的数据被舍弃，D2 第 0 位和第 1 位的数据由 0 补齐，然后组成一个 32 位数据放入 D5 和 D4 中。

对于程序中的右移指令，经过右移五位以后，将结果存放到 D12 和 D13 寄存器内，那么 D12 和 D13 寄存器内的数据为

D13

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

D12

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

——D12 作为 32 位数据的低 16 位，D13 作为 32 位数据的高 16 位

其中，D2 第 0 位数据到第 4 位数据被舍弃，D3 第 11 位数据到第 15 位数据由 0 补齐。然后组成一个 32 位数据放入 D13 和 D12 中。

使用限制：MOV_M_L/ MOV_M_R 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

注意：该指令中，指令参数 3 的范围取值是 0～31，如果该值大于 31，将会只取该值的低 5 位数据。

3.3.4 32 位数据循环移位指令

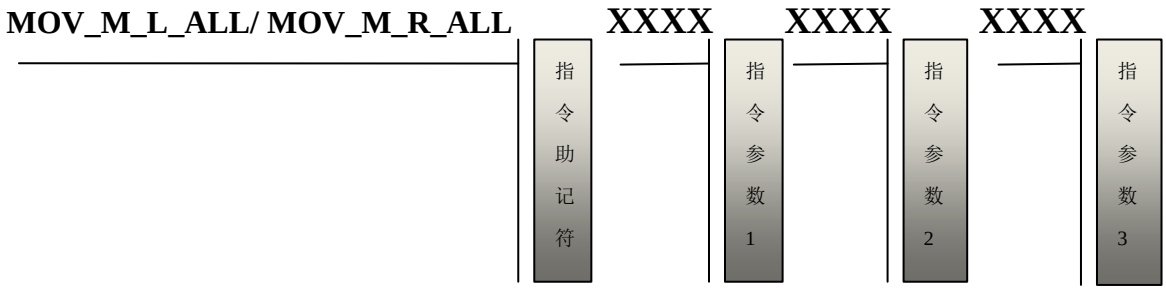
32 位数据循环移位指令包括循环左移和循环右移指令，可以将两个普通数据寄存器 D 组成的一个 32 位数据寄存器存放的 32 位数据进行循环左移或者循环右移操作，并且按照低位在前，高位在后的原则存放到两个目标普通寄存器 D 内。

MOV_M_L_ALL/MOV_M_R_ALL：32 位数据循环左移 n 位/32 位数据循环右移 n 位

MOV_M_L_ALL：该指令可以将两个普通数据寄存器 Dn 和 Dn+1 内的数据循环左移特定的位数，并且存储到指定的普通数据寄存器 Dm 和 Dm+1 内，移到最高位以外的数据将被补充到移出的低位中。

MOV_M_R_ALL：该指令可以将两个普通数据寄存器 Dn 和 Dn+1 内的数据循环右移特定的位数，并且存储到指定的普通数据寄存器 Dm 和 Dm+1 内，移到最低位以外的数据将被补充到移出的高位中。

MOV_M_L_ALL/ MOV_M_R_ALL 指令的助记符语句结构为



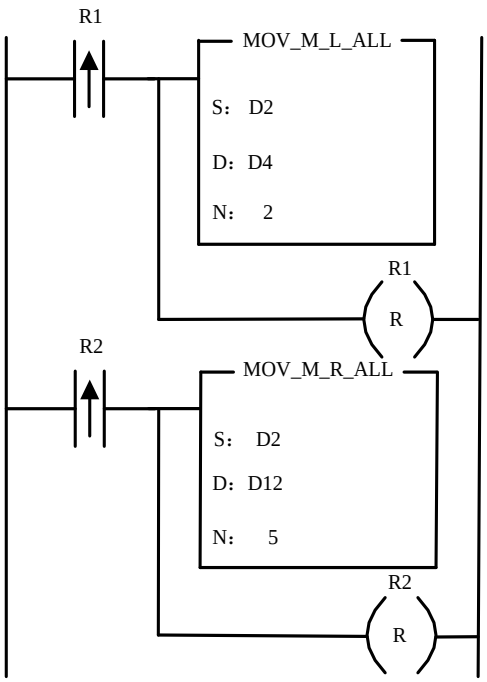
指令参数 1 表示需要移位的 32 位数据寄存器的低 16 位地址 Dn；

指令参数 2 表示移位之后存储的 32 位目标数据寄存器的低 16 位地址 Dm；

指令参数 3 表示循环移位的个数；

例如

梯形图



助记符

```
LD_R          R1
MOV_M_L_ALL   FUNC
              D2
              D4
              2
RESET         R1
LD_R          R2
MOV_M_R_ALL   FUNC
              D2
              D12
              5
RESET         R2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测继电器 R1 的上升沿
MOV_M_L_ALL	FUNC	将寄存器 D2 和 D3 组成的 32 位数据循环左移
	D2	2 位并将结果存入寄存器 D4 和 D5 中
	D4	
	2	
RESET	R1	清除继电器 R1
LD_R	R2	检测继电器 R2 的上升沿
MOV_M_R_ALL	FUNC	将 D2 和 D3 组成的 32 位数据循环右移 5 位并
	D2	将结果存入 D12 和 D13
	D12	
	5	
RESET	R2	清除继电器 R2

例如下表

假设 D2 和 D3 存放的数据为

D3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

——D2 作为 32 位数据的低 16 位，D3 作为 32 位数据的高 16 位

对于程序中的左移指令，经过左移两位以后，将结果存放到 D4 和 D5 寄存器内，那么 D4 和 D5 寄存器内的数据为

D5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	0	0	0	1	0	1	1	0

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	0	0	0	1	0	1	0	1

——D4 作为 32 位数据的低 16 位，D5 作为 32 位数据的高 16 位

其中，D3 的第 14 位和第 15 位的数据被循环移位到 D2 的第 0 位和第 1 位，然后组成一个 32 位数据放入 D5 和 D4 中。

对于程序中的右移指令，经过右移五位以后，将结果存放到 D12 和 D13 寄存器内，那么 D12 和 D13 寄存器内的数据为

D13

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	0	1	1	0	0	0	0	1	0

D12

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	0	1	1	0	0	0	0	1	0

——D12 作为 32 位数据的低 16 位，D13 作为 32 位数据的高 16 位

其中，D2 第 0 位数据到第 4 位数据被循环移位到 D3 第 11 位数据到第 15 位，然后组成一个 32 位数据放入 D13 和 D12 中。

使用限制：MOV_M_L_ALL/ MOV_M_R_ALL 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

注意：该指令中，指令参数 3 的范围取值是 0～31，如果该值大于 31，将会只取该值的低 5 位数据。

3.4 整型数据算术运算指令

MOTEC 智能驱动器内置可编程控制器内部整型数据分为 16 位整型数据和 32 位整型数据，都可以进行算术运算。该算术运算指令能操作 D 寄存器和 P 寄存器。

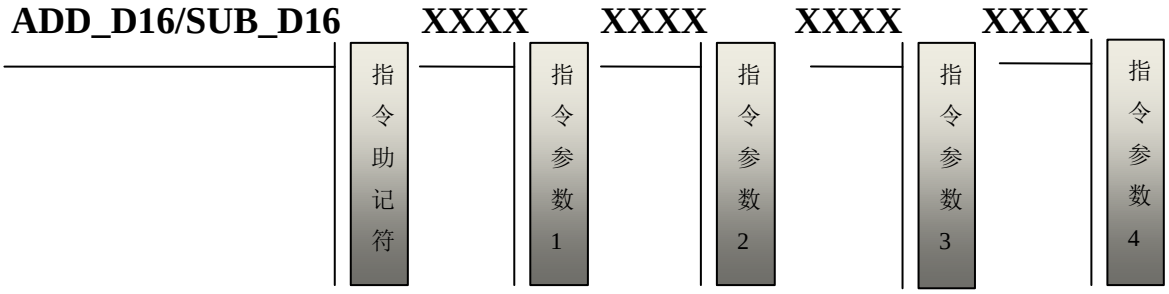
3.4.1 16 位数据算术运算指令

ADD_D16/ SUB_D16：16 位数据加法操作/16 位数据减法操作

ADD_D16：对一个数据寄存器内的数据进行加法操作，相加的对象可以是 16 位数据常数或者 D 寄存器内部的数据。

SUB_D16：对一个数据寄存器内的数据进行减法操作，相减的对象可以是 16 位数据常数或者 D 寄存器内部的数据。

ADD_D16/ SUB_D16 指令的助记符语句结构为



指令参数 1：指令参数一表示有符号或无符号整型数据：

指令参数 2：表示第一个加数/被减数的寄存器地址：

指令参数 3：表示第二个加数/减数的寄存器地址或者常数：

指令参数 4：表示存放运算结果的 D 寄存器地址：

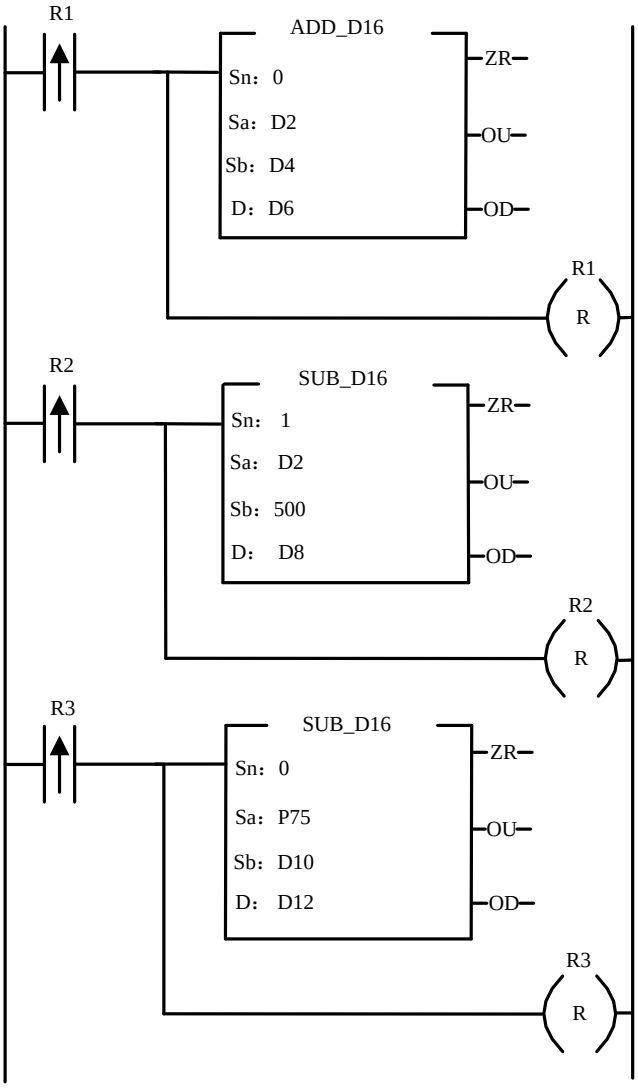
具体见下表

指令参数 1	指令参数 2	指令参数 3	指令参数 4	运算结果	
				加法操作	减法操作
有符号/无符号	普通数据寄存器地址（Dn）	16 位常数(k)	输出寄存器地址(Da)	Da = Dn + k	Da = Dn - k
	普通数据寄存器地址（Dn）	16 位 D 寄存器地址(Dm)		Da = Dn + Dm	Da = Dn - Dm
	驱动器内部数据寄存器地址（Pn）	16 位常数(k)		Da = Pn + k	Da = Pn - k
	驱动器内部数据寄存器地址（Pn）	16 位 D 寄存器地址(Dm)		Da = Pn + Dm	Da = Pn - Dm

目标寄存器只能存放 16 位数据，有符号范围为-32768～32767，无符号范围为 0~65535，运算结果超出该最大范围时，数值取自超出数值，而且运算结果有三个状态输出，分别是结果等于零、超出最大值、低于最小值，可以是辅助继电器 R，也可是输出继电器 Y，可以用该输出继电器来判断数据运算的是否溢出。请用户注意，目标寄存器只能是普通数据寄存器 D，如果目标寄存器的地址超出范围，会产生错误。

例如

梯形图



助记符

```
LD_R        R1
ADD_D16     FUNC
            1
            0
            D2
            D4
            D6
RESET       R1
LD_R        R2
SUB_D16     FUNC
            0
            1
            D2
            500
            D8
RESET       R2
LD_R        R3
SUB_D16     FUNC
            3
            0
            P75
            D10
            D12
RESET       R3
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测辅助继电器 R1 的上升沿
ADD_D16	FUNC	将寄存器 D2 内的数据和寄存器 D4 的数据相
	1	加，并且将结果存入寄存器 D6 中
	0	
	D2	
	D4	
	D6	
RESET	R1	清除继电器 R1
LD_R	R2	检测辅助继电器 R2 的上升沿
SUB_D16	FUNC	将寄存器 D2 内的数据和常数 500 相减，并且
	0	将结果存入寄存器 D8 中
	1	
	D2	
	500	
	D8	
RESET	R2	清除继电器 R2
LD_R	R3	检测继电器 R3 的上升沿
SUB_D16	FUNC	将寄存器 P75 内的数据和寄存器 D10 的数据
	3	相减，并且将结果存入寄存器 D12 中
	0	
	P75	
	D10	
	D12	
RESET	R3	清除继电器 R3

程序说明：如果检测到 R1 的上升沿，假设 D2 内存放的数据是 100，D4 存放的数据是 200，那么 D6 = D2 + D4 = 100 + 200 = 300;

如果检测到 R2 的上升沿，假设 D2 存放的数据是 100,那么

D8 = D2 – 500 = 100 –500 = -400;

如果检测到 R3 的上升沿，假设 P75 内存放的数据是 1000， D10 存放的数据是 500，那么

D12 = P75 – D10 = 1000 – 500 = 500;

使用限制：ADD_D16/ SUB_D16 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

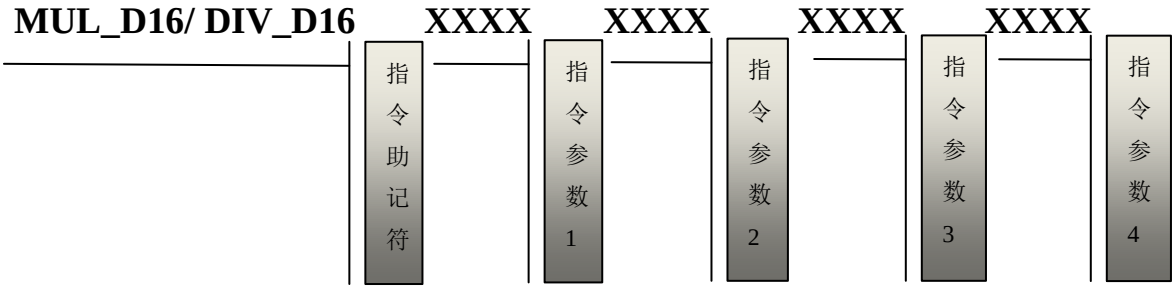
注意：如果驱动器的运算结果超出范围，那么目标寄存器里面的数据取自超出数据，而且运算结果有三个状态输出，分别是结果等于零，超出 16 位数据最大值、低于 16 位数据的最小值三种计算结果状态，可以是辅助继电器 R，也可是输出继电器 Y，可以用该输出继电器来判断数据运算的是否溢出。

MUL_D16/DIV_D16：16 位数据乘法操作/16 位数据除法操作

MUL_D16：对一个数据寄存器内的数据进行乘法操作，相乘的对象可以是 16 位数据常数或者 D 寄存器内部的数据。

DIV_D16：对一个数据寄存器内的数据进行除法操作，相除的对象可以是 16 位数据常数或者 D 寄存器内部的数据。

MUL_D16/ DIV_D16 指令的助记符语句结构为



指令参数 1：指令参数一表示有符号或无符号整型数据：

指令参数 2：表示第一个乘数/被除数的寄存器地址：

指令参数 3：表示第二个乘数/除数的寄存器地址或者常数：

指令参数 4：表示存放运算结果的 D 寄存器地址：

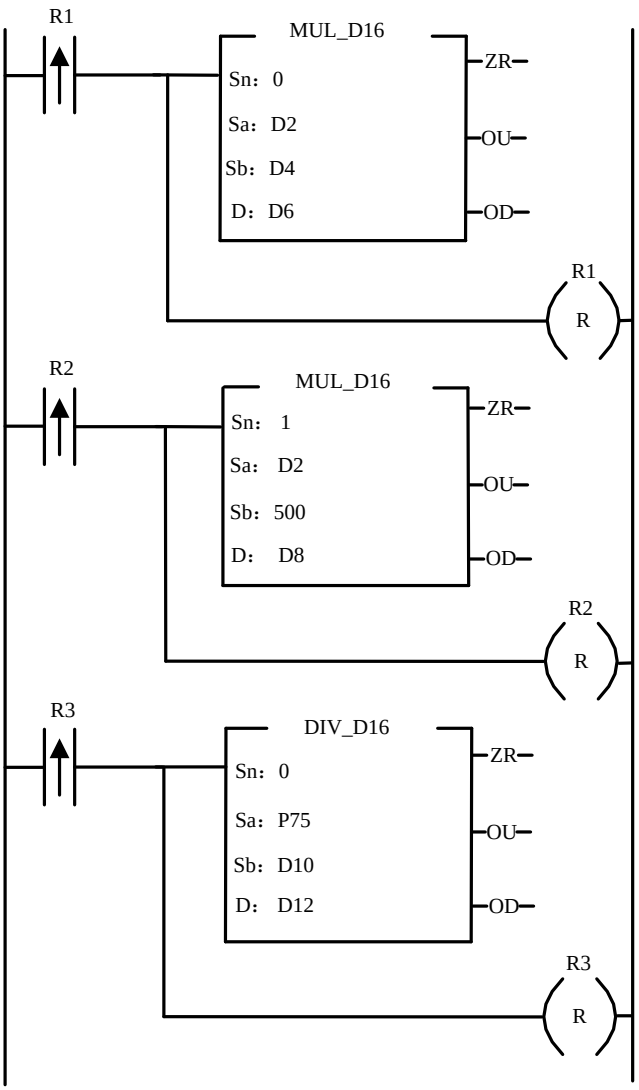
具体见下表

指令参数 1	指令参数 2	指令参数 3	指令参数 4	运算结果	
				乘法操作	除法操作
有符号/无符号	普通数据寄存器地址（Dn）	16 位常数(k)	输出寄存器地址(Da)	Da=Dn×k	Da=Dn/k
	普通数据寄存器地址（Dn）	16 位 D 寄存器地址(Dm)		Da=Dn×Dm	Da=Dn/Dm
	驱动器内部数据寄存器地址（Pn）	16 位常数(k)		Da=Pn×k	Da=Pn/k
	驱动器内部数据寄存器地址（Pn）	16 位 D 寄存器地址(Dm)		Da=Pn×Dm	Da=Pn/Dm

对于乘法/除法操作，有符号范围为-32768～32767，无符号范围为 0~65535，运算结果超出该最大范围时，数值取自超出部分，而且运算结果有三个状态输出，分别是结果等于零、超出最大值、低于最小值，可以是辅助继电器 R，也可是输出继电器 Y，可以用该输出继电器来判断数据运算的是否溢出。请用户注意目标寄存器只能是普通数据寄存器 D，如果目标寄存器的地址超出范围，会产生错误。

例如

梯形图



助记符

LD_R	R1
MUL_D16	FUNC
	1
	0
	D2
	D4
	D6
RESET	R1
LD_R	R2
MUL_D16	FUNC
	0
	1
	D2
	500
	D8
RESET	R2
LD_R	R3
DIV_D16	FUNC
	3
	0
	P75
	D10
	D12
RESET	R3

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测辅助继电器 R1 的上升沿
MUL_D16	FUNC	将寄存器 D2 内的数据和寄存器 D4 的数据相
	1	乘，并且将结果存入寄存器 D6 中
	0	
	D2	
	D4	
	D6	
RESET	R1	清除继电器 R1
LD_R	R2	检测辅助继电器 R2 的上升沿
MUL_D16	FUNC	将寄存器 D2 内的数据和常数 500 相乘，并且
	0	将结果存入寄存器 D8 中
	1	
	D2	
	500	
	D8	
RESET	R2	清除继电器 R2
LD_R	R3	检测辅助继电器 R3 的上升沿
DIV_D16	FUNC	将寄存器 P75 内的数据和寄存器 D10 的数据
	3	相除，并且将结果存入寄存器 D12 中
	0	
	P75	
	D10	
	D12	
RESET	R3	清除继电器 R3

程序说明：如果检测到 R1 的上升沿，假设 D2 内存放的数据是 2，D4 存放的数据是 1000，

D2 × D4 = 2 × 1000 = 2000;

那么 D6 里面存放的就是 2000,

如果检测到 R2 的上升沿，假设 D2 存放的数据是 2,

D2 × 500 =2 × 500 =1000;

那么 D8 里面存放的就是 1000，

如果检测到 R3 的上升沿，假设 P75 内存放的数据是 1000， D10 存放的数据是 2，那么

D12 = P75/D10 = 1000 / 2 = 500;

使用限制：MUL_D16/ DIV_D16 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

注意：如果驱动器的运算结果超出范围，那么目标寄存器里面的数据取自超出数据，而且运算结果有三个状态输出，分别是结果等于零，超出 16 位数据最大值、低于 16 位数据的最小值三种计算结果状态，可以是辅助继电器 R，也可是输出继电器 Y。

在除法操作中，如果出现了除数为 0 的情况，除法操作将不能被执行，并且将三个状态输出全部置为 OFF 状态。

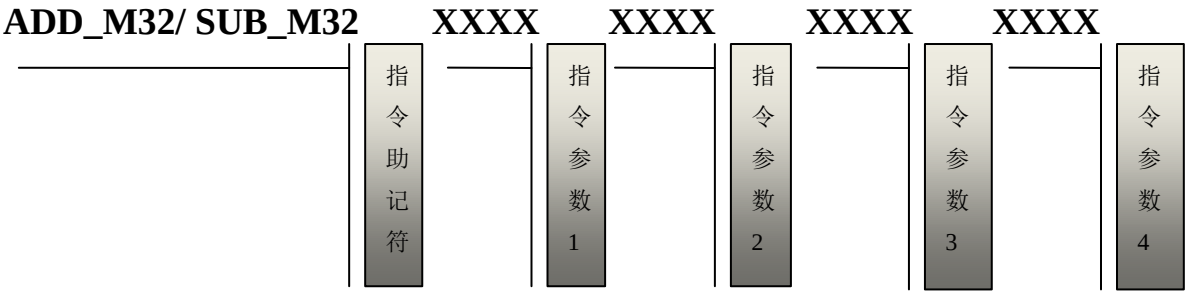
3.4.2 32 位数据算术运算指令

ADD_M32/ SUB_M32：32 位数据加法操作/32 位数据减法操作

ADD_D32：对一个数据寄存器内的数据进行加法操作，相加的对象可以是 32 位数据常数或者 D 寄存器内部的数据，如果使用寄存器，只能是 D 寄存器。

SUB_D32：对一个数据寄存器内的数据进行减法操作，相减的对象可以是 32 位数据常数或者 D 寄存器内部的数据，如果使用寄存器，只能是 D 寄存器。

ADD_M32/ SUB_M32 指令的助记符语句结构为



指令参数 1：指令参数一表示有符号或无符号整型数据：

指令参数 2：表示第一个加数/被减数的低 16 位 D 寄存器地址：

指令参数 3：表示第二个加数/减数的低 16 位 D 寄存器地址或者常数：

指令参数 4：表示存放运算结果低 16 位的 D 寄存器地址：

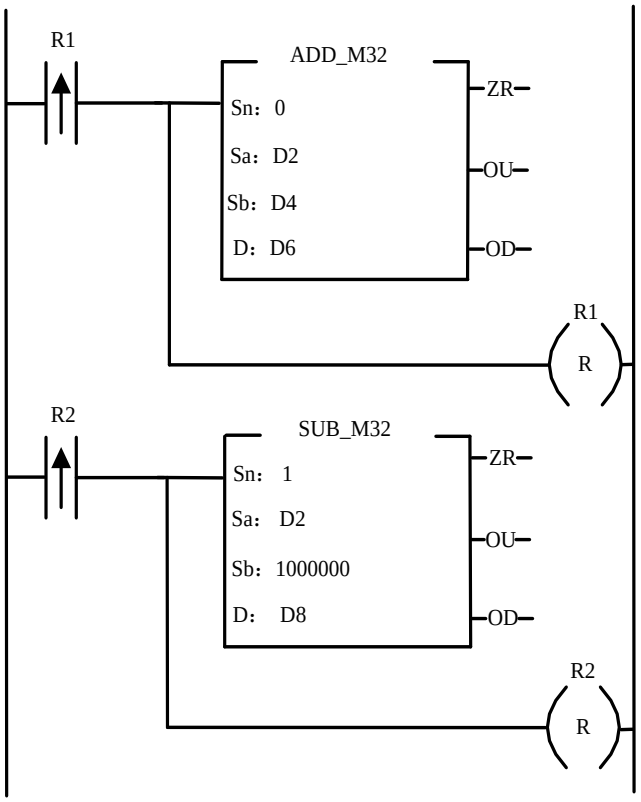
具体见下表

指令参数 1	指令参数 2	指令参数 3	指令参数 4
有符号/无符号	普通数据寄存器地址（Dn）	32 位常数(k)	输出寄存器地址 (Da)
	普通数据寄存器地址（Dn）	普通数据寄存器地址(Dm)	

目标寄存器存放运算结果的低 16 位数据，目标寄存器地址加一的寄存器存放运算结果高 16 位的数据，并且在程序运算过程中是以 32 位有符号的整型数据来运算的，有符号取值范围是-2147483648～2147483647，无符号取值范围为 0~4294967295，运算结果超出该最大范围时，数值取自超出部分，而且运算结果有三个状态输出，分别是运算结果等于零、超出最大值、低于最小值，可以是辅助继电器 R，也可是输出继电器 Y，可以用该输出继电器来判断数据运算的是否溢出。请用户注意目标寄存器只能是普通数据寄存器 D，如果目标寄存器的地址超出范围，会产生错误。

例如

梯形图



助记符

LD_R	R1
ADD_M32	FUNC
	1
	0
	D2
	D4
	D6
RESET	R1
LD_R	R2
SUB_M32	FUNC
	0
	0
	D2
	1000000
	D8
RESET	R2

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测辅助继电器 R1 的上升沿
ADD_M32	FUNC	将寄存器 D2 和 D3 内的 32 位数据和寄存器
	1	D4 和 D5 内的 32 位数据相加，并且将结果存
	0	入寄存器 D6 和 D7 中
	D2	
	D4	
	D6	
RESET	R1	清除继电器 R1
LD_R	R2	检测辅助继电器 R2 的上升沿
SUB_M32	FUNC	将寄存器 D2 和 D3 内的 32 位数据和常数
	0	100000 相减，并且将结果存入寄存器 D8 和
	0	D9 中
	D2	
	1000000	
	D8	
RESET	R2	清除继电器 R2

程序说明：如果检测到 R1 的上升沿，假设 D2 和 D3 内存放的 32 位数据是 500000，D4 存放的数据是 60000，

500000 + 60000 = 560000；

那么 D6 中存放 560000 的低 16 位，0x8B80，D7 中存放 560000 的高 16 位，0x0008，

如果检测到 R2 的上升沿，假设 D2 和 D3 内存放的数据是 500000，

500000 - 100000 = 400000；

那么那么 D8 中存放 400000 的低 16 位，0x1A80，D9 中存放 560000 的高 16 位，0x0006，

使用限制：ADD_M32/ SUB_M32 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且 ADD_M32/ SUB_M32 的操作对象只能是 D 寄存器，不能是 P 寄存器。

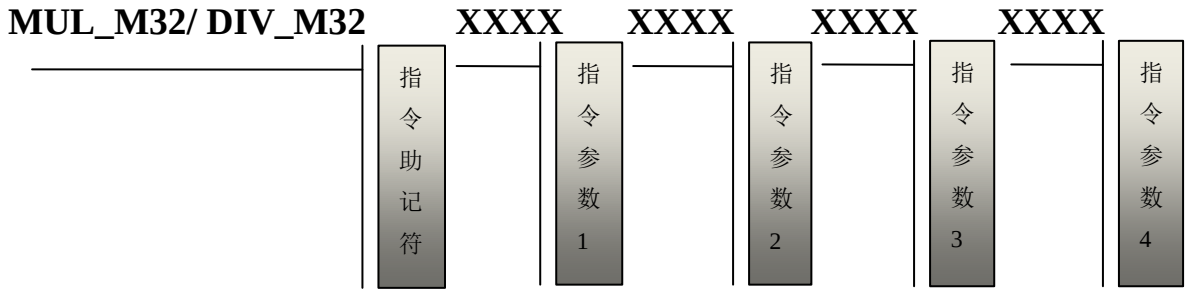
注意：如果驱动器的运算结果超出范围，那么目标寄存器里面的数据取自超出数据，而且运算结果有三个状态输出，分别是结果等于零、超出最大值、低于最小值，可以是辅助继电器 R，也可是输出继电器 Y。

MUL_M32/ DIV_M32：32 位数据乘法操作/32 位数据除法操作

MUL_D32：对一个数据寄存器内的数据进行乘法操作，相乘的对象可以是 32 位数据常数或者 D 寄存器内部的数据，如果使用寄存器，只能是 D 寄存器。

DIV_D32：对一个数据寄存器内的数据进行除法操作，相除的对象可以是 32 位数据常数或者 D 寄存器内部的数据，如果使用寄存器，只能是 D 寄存器。

MUL _M32/DIV _M32 指令的助记符语句结构为



指令参数 1：指令参数一表示有符号或无符号整型数据：

指令参数 1 表示第一个乘数/被除数的低 16 位 D 寄存器地址：

指令参数 2 表示第二个乘数/除数的低 16 位寄 D 存器地址或者常数：

指令参数 3 表示存放运算结果低 16 位的 D 寄存器地址：

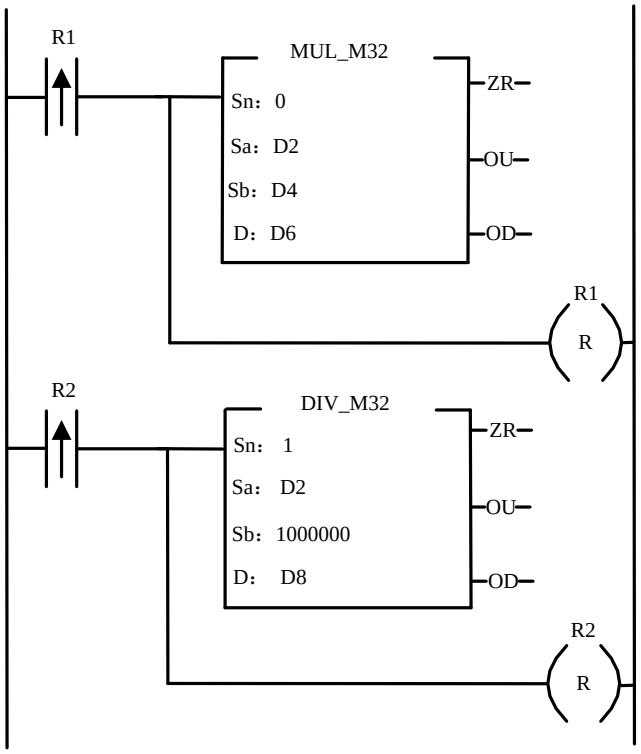
具体见下表

指令参数 1	指令参数 2	指令参数 3	指令参数 4
有符号/无符号	普通数据寄存器地址（Dn）	32 位常数(k)	输出寄存器地址(Da)
	普通数据寄存器地址（Dn）	普通数据寄存器地址(Dm)	

目标寄存器存放运算结果的低 16 位数据，目标寄存器地址加一的寄存器存放运算结果高 16 位的数据，并且在程序运算过程中是以 32 位有符号的整型数据来运算的，有符号取值范围是-2147483648～2147483647，无符号取值范围为 0~4294967295，运算结果超出该最大范围时，数值取自超出部分，而且运算结果有三个状态输出，分别是运算结果等于零、超出最大值、低于最小值，可以是辅助继电器 R，也可是输出继电器 Y，可以用该输出继电器来判断数据运算的是否溢出。请用户注意目标寄存器只能是普通数据寄存器 D，如果目标寄存器的地址超出范围，会产生错误。

例如

梯形图



助记符

LD_R	R1
MUL_M32	FUNC
	1
	0
	D2
	D4
	D6
RESET	R1
LD_R	R2
DIV_M32	FUNC
	0
	0
	D2
	1000000
	D8
RESE	R2

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测辅助继电器 R1 的上升沿
MUL_M32	FUNC	将寄存器 D2 和 D3 内的 32 位数据和寄存器
	1	D4 和 D5 内的 32 位数据相乘，并且将结果存
	0	入寄存器 D6 和 D7 中
	D2	
	D4	
	D6	
RESET	R1	清除继电器 R1
LD_R	R2	检测辅助继电器 R2 的上升沿
DIV_M32	FUNC	将寄存器 D2 和 D3 内的 32 位数据和常数
	0	100000 相除，并且将结果存入寄存器 D8 和
	0	D9 中
	D2	
	1000000	
	D8	
RESET	R2	清除继电器 R2

程序说明：如果检测到 R1 的上升沿，假设 D2 和 D3 内存放的 32 位数据是 50000，D4 存放的数据是 6000，

50000 × 6000 = 30000000；

那么 D6 中存放 30000000 的低 16 位，0x A300，D7 中存放 30000000 的高 16 位，0x11E1，

如果检测到 R2 的上升沿，假设 D2 和 D3 内存放的数据是 5000000，

5000000 /1000000 =5；

那么 D8 中存放 5 的低 16 位，0x0005，D9 中存放 5 的高 16 位，0x0000，

使用限制：MUL_M32/ DIV_M32 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且 ADD_M32/ SUB_M32 的操作对象只能是 D 寄存器，不能是 P 寄存器。

注意：如果驱动器的运算结果超出范围，那么目标寄存器里面的数据取自超出数据，而且运算结果有三个状态输出，分别是结果等于零、超出最大值、低于最小值，可以是辅助继电器 R，也可是输出继电器 Y。

注意：在除法操作中，如果出现了除数为 0 的情况，除法操作将不能被执行，并且将三个状态输出全部置为 OFF 状态。

3.5 整型数据数值运算

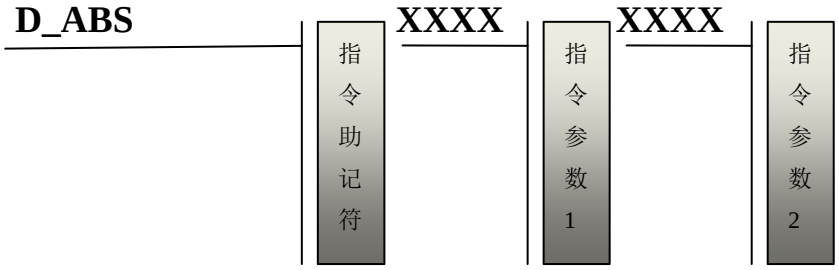
MOTEC 智能驱动器内置可编程控制器内部整型数据分为 16 位整型数据和 32 为整型数据，都可以进行数值运算。该运算指令只能操作 D 寄存器。

3.5.1 16 位整型数据数值运算

D_ABS 16 位数据求绝对值

D_ABS：该指令可以将一个普通数据寄存器内的数据求绝对值，并且存储到指定的普通数据寄存器。

D_ABS 的指令的助记符语句结构为



指令参数 1 表示 16 位整型数据 D 寄存器地址；

指令参数 2 表示目标寄存器地址。

具体详见下表

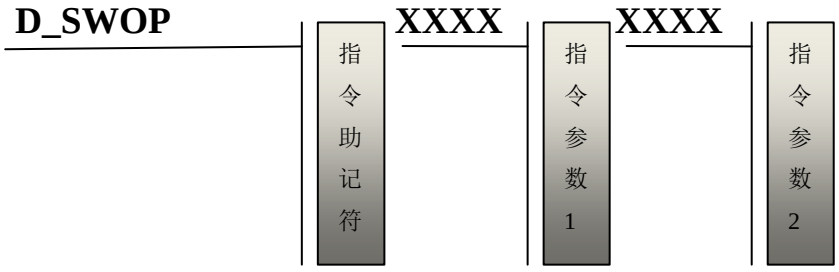
参数 1	参数 2
16 位普通数据寄存器（Dn）	输出普通寄存器地址（Da）

目标寄存器只能存放 16 位数据，并且在程序运算过程中是以 16 位有符号整型数据来运算的，所以该指令的最大取值范围是-32768～32767。

D_SWOP 16 位数据高 8 位与低 8 位交换

D_SWOP：该指令可以将一个 16 位普通数据寄存器内的数据的高 8 位与低 8 位交换，交换的结果存储到指定的普通数据寄存器中。

D_SWOP 的指令的助记符语句结构为



指令参数 1 表示 16 位整型数据 D 寄存器地址；

指令参数 2 表示目标寄存器地址。

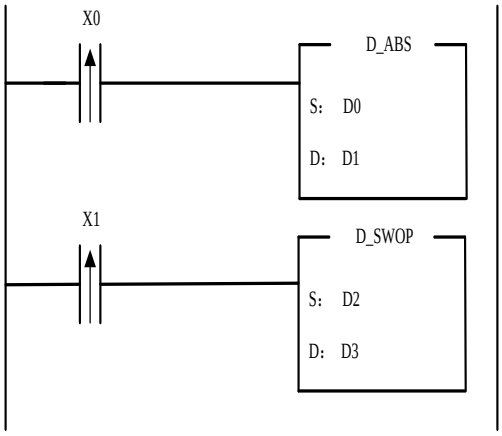
具体详见下表

参数 1	参数 2
16 位普通数据寄存器（Dn）	输出普通寄存器地址（Da）

目标寄存器只能存放 16 位数据，并且在程序运算过程中是以 16 位有符号整型数据来运算的，所以该指令的最大取值范围是-32768～32767。

例如

梯形图



助记符

```
LD_R      X0
D_ABS     FUNC
          D0
          D1
LD_R      X1
D_SWOP    FUNC
          D2
          D3
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 的上升沿
D_ABS	FUNC	将寄存器 D0 中的数据求绝对值，运算结果
	D0	
	D1	存入普通寄存器 D1 中
LD_R	X1	检测继电器 X1 的上升沿
D_SWOP	FUNC	将寄存器 D2 中的数据高 8 位与低 8 位交换，
	D2	
	D3	运算结果存入普通寄存器 D3 中

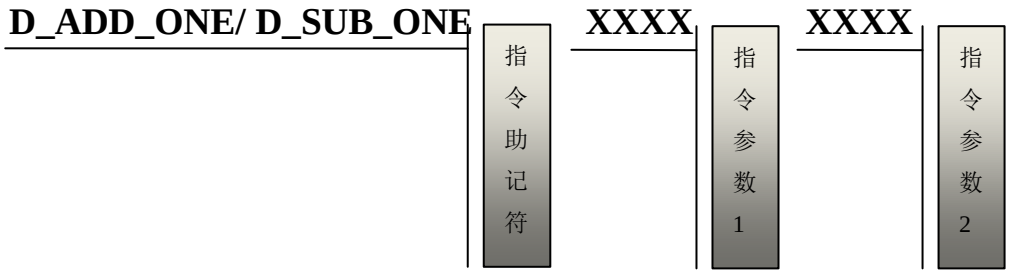
程序说明：当检测到 X0 的上升沿，假设 D0 中存放的数据为-1234，那么 D1 中存入的数据为 1234。当检测到 X1 的上升沿，假设 D2 中存放的数据为 100，那么 D3 中存入的数据为 25600。

使用限制：D_ABS/D_SWOP 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且 D_ABS/D_SWOP 的操作对象只能是 D 寄存器，不能是 P 寄存器。

D_ADD_ONE/ D_SUB_ONE 16 位数据加 1/16 位数据减 1

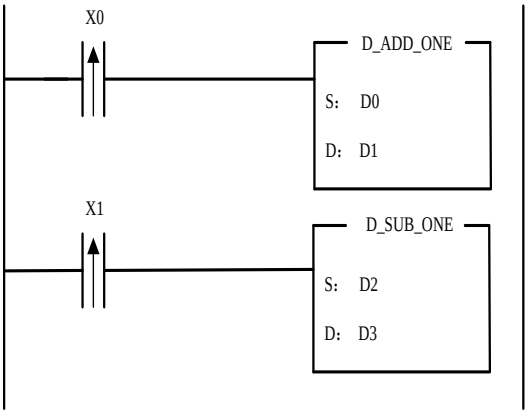
D_ADD_ONE：将一个普通数据寄存器中的数据加 1，运算结果存入指定的普通寄存器中。
D_SUB_ONE：将一个普通数据寄存器中的数据减 1，运算结果存入指定的普通寄存器中。

D_ADD_ONE/ D_SUB_ONE 的指令的助记符语句结构为



指令参数 1 表示 16 位整型数据 D 寄存器地址；
指令参数 2 表示目标寄存器地址。
例如

梯形图



助记符

LD_R	X0
D_ADD_ONE	FUNC
	D0
	D1
LD_R	X1
D_SUB_ONE	FUNC
	D2
	D3

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 的上升沿
D_ADD_ONE	FUNC	普通数据寄存器 D0 中的数据加 1，将运
	D0	结果存入普通数据寄存器 D1 中
	D1	
LD_R	X1	检测继电器 X1 的上升沿
D_SUB_ONE	FUNC	普通数据寄存器 D2 中的数据减 1，并将运
	D2	算结果存入寄存器 D3 中
	D3	

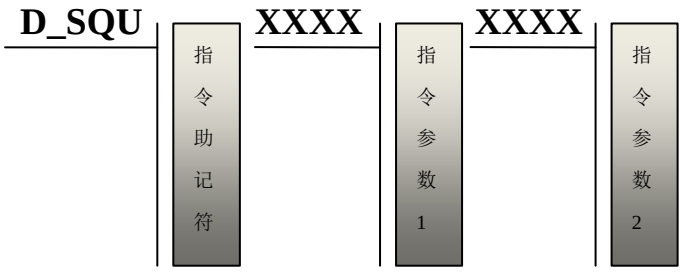
程序说明：当检测继电器 X0 上升沿，寄存器 D0 中的数据加 1，假设 D0 中存放的数据为 3，那么目标寄存器 D1 中存入的数据为 4。当检测到继电器 X1 上升沿，寄存器 D2 中的数据减 1，假设寄存器 D2 中存放的数据为 12，那么目标寄存器 D3 中存入的数据为 11。

使用限制：D_ADD_ONE/ D_SUB_ONE 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且 D_ADD_ONE/ D_SUB_ONE 的操作对象只能是 D 寄存器。

D_SQU 16 位数据开平方

D_SQU：将一个普通数据寄存器中的数据开平方，运算结果存入指定的普通寄存器中。

D_SQU 的指令的助记符语句结构为

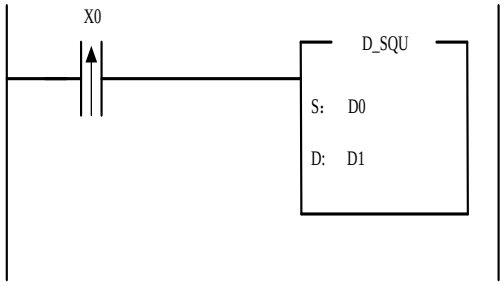


指令参数 1 表示 16 位整型数据 D 寄存器地址；

指令参数 2 表示目标寄存器地址。

例如

梯形图



助记符

```
LD_R      X0
D_SQU     FUNC
          D0
          D1
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
D_SQU	FUNC	将普通寄存器 D0 中的数据开平方，运算结果存入普通数据寄存器 D1 中
	D0	
	D1	

程序说明：当检测继电器 X0 上升沿时，普通数据寄存器 D0 中的数据开平方，假设寄存器 D0 中存放的数据为 16，那么目标寄存器 D1 中存入的数据为 4。

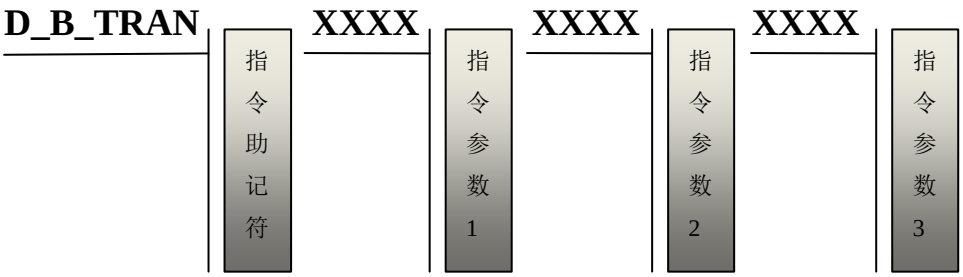
使用限制：D_SQU 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且 D_SQU 的操作对象只能是 D 寄存器。

注意：D_SQU 该指令运算结果存入的目标寄存器的数据为整数，由于是整型数据开方，将直接舍弃小数点以后的数，例如 64 的开平方为 8，65~80 的开平方也为 8，81 的开平方为 9。

D_B_TRAN 16 位数据块传输

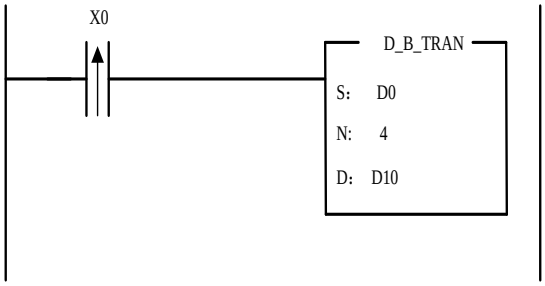
D_B_TRAN：该指令将 16 位普通数据寄存器中数据块传输到目标寄存器中。

D_B_TRAN 的指令的助记符语句结构为



指令参数 1 表示传输的 16 位寄存器首地址；
指令参数 2 表示传输 16 位寄存器的个数；
指令参数 3 表示传输目标 16 位普通数据寄存器首地址。
例如

梯形图



助记符

LD_R	X0
D_B_TRAN	FUNC
	D0
	4
	D10

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
D_B_TRAN	FUNC	将普通数据寄存器 D0 为首的寄存器中的数据传输到寄存器 D10 为首的寄存器中，传输的寄存器数据的个数为 4
	D0	
	4	
	D10	

程序说明：当检测继电器 X0 上升沿，普通数据寄存器 D0、D1、D2、D3 中的数据分别传输到寄存器 D10、D11、D12、D13 中。假设寄存器 D0 中存放的数据为 1 寄存器 D1 中存放的数据为 2，寄存器 D2 中存放的数据为 3，寄存器 D3 中存放的数据为 4，那么寄存器 D10、D11、D12、D13 中存入的数据分别为 1、2、3、4。

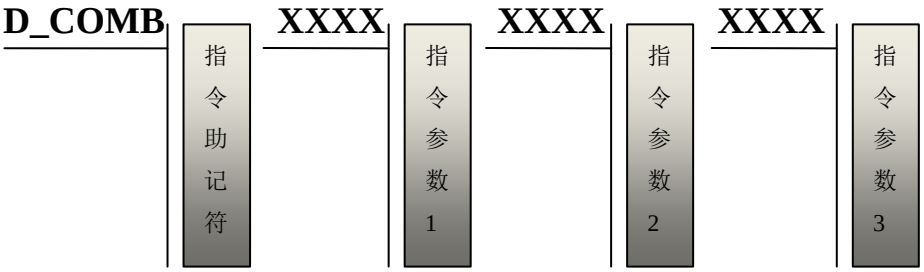
使用限制：D_B_TRAN 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

注意：16 位的块传输最多同时传输连续地址的 32 个 16 位数据寄存器内的数值。

D_COMB 两个 16 位数据组成一个 32 位数据

D_COMB：该指令是将两个 16 位寄存器中的数据组成一个 32 位数据，并存入指定的两个连续普通寄存器组成 32 位数据寄存器中。

D_COMB 的指令的助记符语句结构为



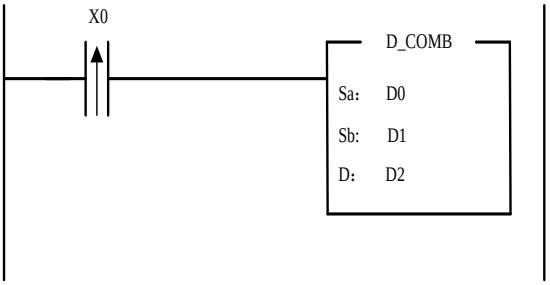
指令参数 1 表示一个 16 位普通数据寄存器地址；
指令参数 2 表示一个 16 位普通数据寄存器地址；
指令参数 3 表示合并组成的 32 位数据目标寄存器低 16 地址。
具体详见下表

指令参数 1	指令参数 2	指令参数 3
普通数据寄存器地址（Dn）	普通数据寄存器地址(Dm)	输出寄存器地址(Da)

目标寄存器存放运算结果的低 16 位数据，目标寄存器地址加一的寄存器存放运算结果高 16 位的数据，并且在程序运算过程中是以 32 位有符号的整型数据来运算的，所以该指令的最大取值范围是-2147483648～2147483647，当运算结果超出该最大范围时，会出现未知的错误，请用户注意。并且目标寄存器只能是普通数据寄存器 D，如果目标寄存器的地址超出范围，同样会产生错误。

例如

梯形图



助记符

```
LD_R        X0
D_COMB      FUNC
            D0
            D1
            D2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
D_COMB	FUNC	将普通数据寄存器 D0 和 D1 的数据合并,组成一个 32 位数据存入 D2，D3 中
	D0	
	D1	
	D2	

程序说明：当检测继电器 X0 的上升沿，将寄存器 D0 和 D1 组成一个 32 位数据存入 D2，D3 中，假设 D0 存放的数据为 123，D1 存放的数据为 456，那么寄存器 D2，D3 中存入的数据为 29884539。

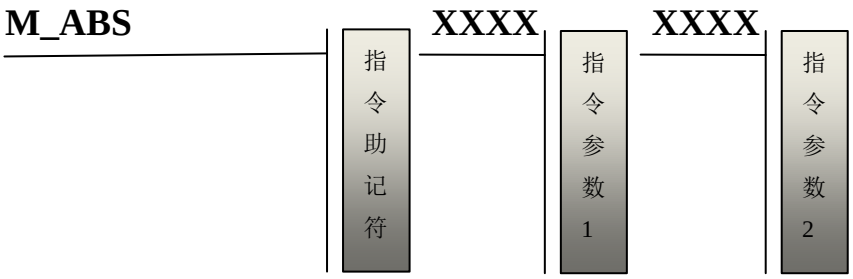
使用限制：D_COMB 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

3.5.2 32 位整型数据数值运算

M_ABS 32 位数据求绝对值

M_ABS：该指令将由两个连续 16 位普通寄存器组成的 32 位数据寄存器中的数据求绝对值，运算结果存入两个连续 16 位普通寄存器中。

M_ABS 的指令的助记符语句结构为



指令参数 1 表示 32 位数据寄存器低 16 位的地址

指令参数 2 表示目标存放 32 位数据的低 16 位寄存器地址。

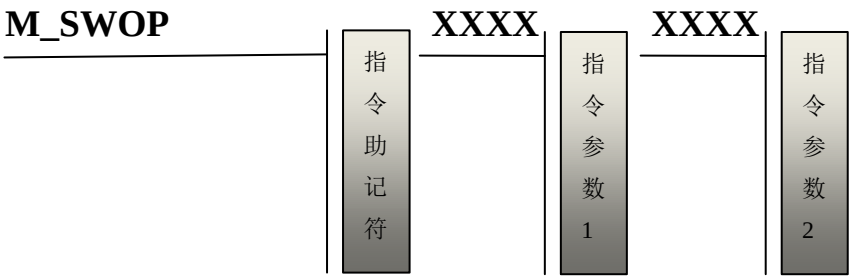
具体详见下表

参数 1	参数 2
32 位普通数据寄存器低 16 位地址(Dn)	输出 32 位普通寄存器低 16 位地址(Da)

M_SWOP 32 32 位数据高 16 位与低 16 位交换

M_SWOP：该指令将由两个连续的 16 位普通寄存器组成的 32 位数据寄存器中的数据高 16 位与低 16 位交换。交换的结果存入两个连续的 16 位普通寄存器中。

M_SWOP 的指令的助记符语句结构为



指令参数 1 表示 32 位数据寄存器低 16 位的地址

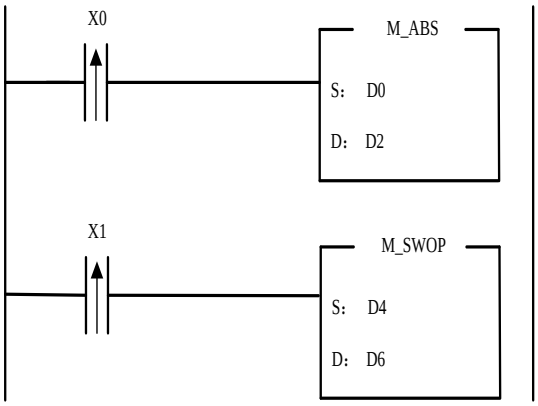
指令参数 2 表示目标存放 32 位数据的低 16 位寄存器地址。

具体详见下表

参数 1	参数 2
32 位普通数据寄存器低 16 位地址(Dn)	输出 32 位普通寄存器低 16 位地址(Da)

例如

梯形图 助记符




```
LD_R      X0
M_ABS     FUNC
          D0
          D2
LD_R      X1
M_SWOP    FUNC
          D4
          D6
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_ABS	FUNC	将 32 位普通数据寄存器 D0，D1 中的数据求
	D0	绝对值，运算结果存入 32 位普通数据寄存
	D2	器 D2，D3 中
LD_R	X1	检测继电器 X1 上升沿
M_SWOP	FUNC	将 32 位普通数据寄存器 D4，D5 中的数据高
	D4	16 位与低 16 位交换，交换的结果存入 32 位
	D6	普通数据寄存器 D6，D7 中

程序说明：当检测到继电器 X0 上升沿，假设 32 位普通数据寄存器 D0，D1 中存放的数据为 123456，那么 32 位普通数据寄存器 D2，D3 存入的数据为 123456。

当检测到继电器 X1 上升沿，假设 32 位普通数据寄存器 D4，D5 中存放的数据为 12345678，那么 32 位普通数据寄存器 1632501948。

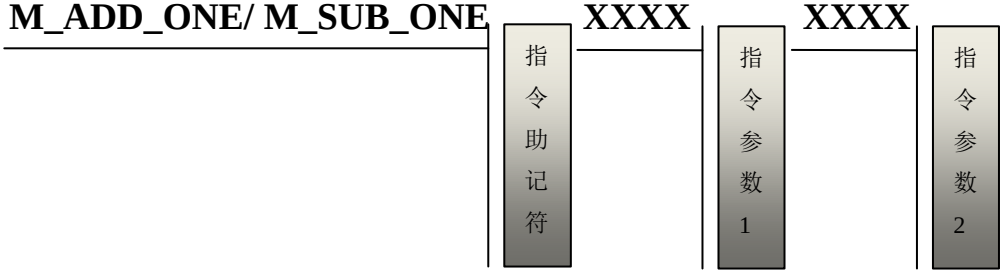
使用限制：M_ABS/ M_SWOP 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

M_ADD_ONE/ M_SUB_ONE 32 位数据加 1/32 位数据减 1

M_ADD_ONE：将由两个连续 16 位普通寄存器组成的 32 位数据寄存器中的数据加 1，运算结果存入指定的由两个连续 16 位普通寄存器组成的 32 位数据寄存器中。

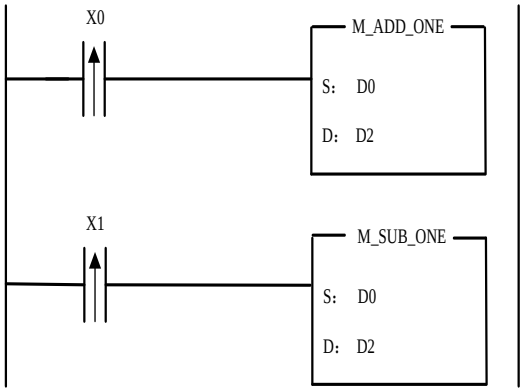
M_SUB_ONE：将由两个连续 16 位普通寄存器组成的 32 位数据寄存器中的数据减 1，运算结果存入指定的由两个连续 16 位普通寄存器组成的 32 位数据寄存器中。

M_ADD_ONE/ M_SUB_ONE 的指令的助记符语句结构为



指令参数 1 表示 32 位数据寄存器低 16 位的地址；
指令参数 2 表示 32 位目标数据寄存器的低 16 位地址
例如

梯形图



助记符

```
LD_R          X0
M_ADD_ONE     FUNC
              D0
              D2
LD_R          X1
M_SUB_ONE     FUNC
              D0
              D2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_ADD_ONE	FUNC	将 32 位普通数据寄存器 D0，D1 中的数据加 1，并将运算结果存入 32 位普通数据寄存器 D2，D3 中
	D0	
	D2	
LD_R	X1	检测继电器 X1 上升沿
M_SUB_ONE	FUNC	将 32 位普通数据寄存器 D0，D1 中的数据减 1，并将运算结果存入 32 位普通数据寄存器 D2，D3 中
	D0	
	D2	

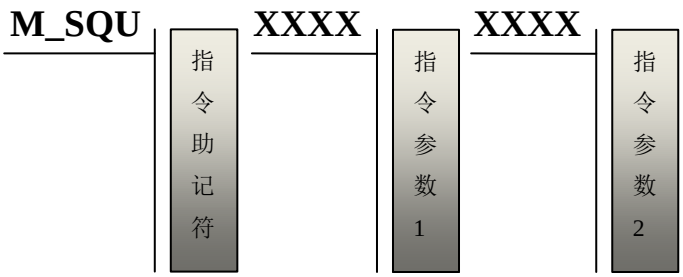
程序说明：当检测继电器 X0 上升沿，假设寄存器 D0，D1 中存放的数据为 123456，那么寄存器 D2，D3 中存入的数据为 123457。当检测继电器 X1 上升沿，那么寄存器 D2，D3 中存入的数据为 123455。

使用限制：M_ADD_ONE/ M_SUB_ONE 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

M_SQU 32 位数据开方

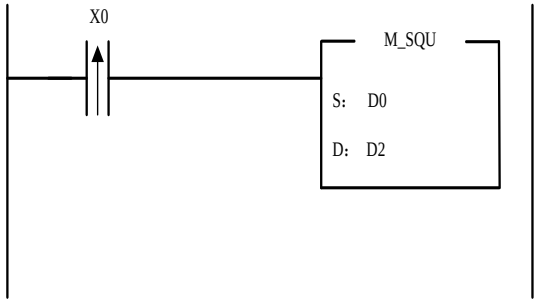
M_SQU：将由两个连续 16 位普通寄存器组成的 32 位数据寄存器中的数据开平方，运算结果存入指定的由两个连续 16 位普通寄存器组成的 32 位数据寄存器中。

M_SQU 的指令的助记符语句结构为



指令参数 1 表示 32 位数据寄存器低 16 位的地址；
指令参数 2 表示 32 位目标数据寄存器的低 16 位地址
例如

梯形图



助记符

```
LD_R    X0
M_SQU   FUNC
        D0
        D2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_SQU	FUNC	将寄存器 D0，D1 中存放的数据开平方，运算结果存入寄存器 D2，D3 中
	D0	
	D2	

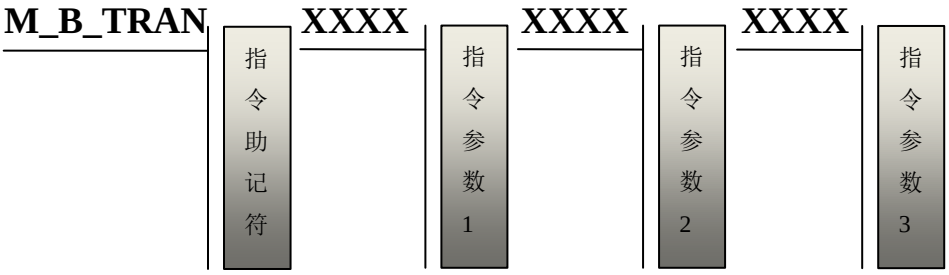
程序说明：当检测继电器 X0 上升沿，假设寄存器 D0，D1 中存放的数据为 152399025，那么寄存器 D2，D3 中存入的数据为 12345。

注意：M_SQU 该指令运算结果存入的目标寄存器的数据为整数，由于是整型数据开方，将直接舍弃小数点以后的数。

M_B_TRAN 32 位数据块传输

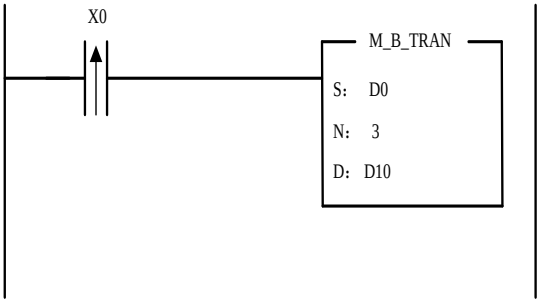
M_B_TRAN：该指令将 32 位普通数据寄存器中数据块传输到目标寄存器中。

M_B_TRAN 的指令的助记符语句结构为



指令参数 1 表示传输的 32 位寄存器低 16 位首地址；
指令参数 2 表示传输 32 位寄存器的个数；
指令参数 3 表示传输目标 32 位普通数据寄存器低 16 位首地址。
例如

梯形图



助记符

LD_R	X0
M_B_TRAN	FUNC
	D0
	3
	D10

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_B_TRAN	FUNC	将 32 位普通数据寄存器 D0，D1 为首的寄存器中的数据传输到 32 位寄存器 D10，D11 为首的寄存器中，传输的寄存器数据的个数为 3。
	D0	
	3	
	D10	

程序说明：当检测继电器 X0 上升沿，将 32 位普通数据寄存器 D0，D1、D2，D3、D4，D5 中的数据分别传输到寄存器 D10，D11、D12，D13、D14，D15 中。假设寄存器 D0，D1 中存放的数据为 11111111，寄存器 D2，D3 中存放的数据为 22222222，寄存器 D4，D5 中存放的数据为 33333333，那么 32 位寄存器 D10，D11、D12，D13、D14，D15 中存入的数据分别为 11111111、22222222、33333333。

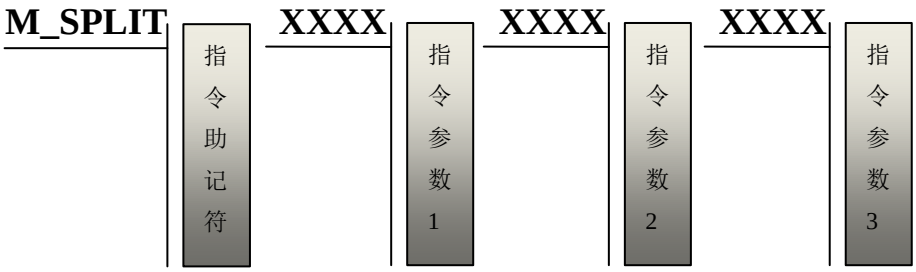
使用限制：M_B_TRAN 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

注意：32 位的块传输最多同时传输地址连续的 16 个 32 位数据。

M_SPLIT 一个 32 位数据拆分成两个 16 位数据

M_SPLIT：该指令将一个 32 位普通数据寄存器中的数据拆分成了两个 16 位数据，拆分后的两个 16 位数据存入两个普通数据寄存器中，目标寄存器可以不是连续的。

M_SPLIT 的指令的助记符语句结构为



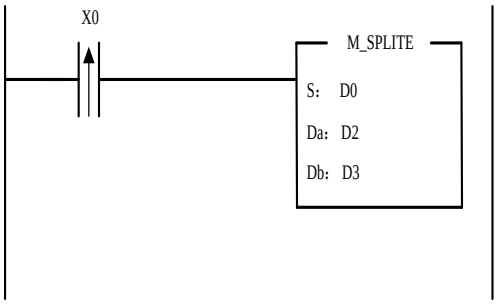
指令参数 1 表示 32 位普通数据寄存器的低 16 位地址；
指令参数 2 表示目标 16 位普通数据寄存器的地址，存入的数据是 32 位数据的低 16 位；
指令参数 3 表示目标 16 位普通数据寄存器的地址，存入的数据是 32 位数据的高 16 位。
具体详见下表

指令参数 1	指令参数 2	指令参数 3
32 位普通数据寄存器低 16 位地址（Dn）	输出寄存器地址(Dm)	输出寄存器地址(Da)

目标寄存器为两个 16 位普通数据寄存器，指令参数 2 中的寄存器存入的是 32 位数据的低 16 位数据，指令参数 3 中的寄存器存入的是 32 位数据的高 16 位数据。

例如

梯形图



助记符

```
LD_R      X0
M_SPLIT   FUNC
          D0
          D2
          D3
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_SPLIT	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数据拆分，拆分后的低 16 位数据存入寄存器 D2 中，高 16 位数据存入寄存器 D3 中
	D0	
	D2	
	D3	

程序说明：当检测继电器 X0 上升沿，假设 32 位寄存器 D0，D1 中存放的数据为 123456，那么 16 位寄存器 D2 中存入的数据为 57920，16 位寄存器 D3 中存入的数据为 1。

使用限制：M_SPLIT 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

注意：拆分后的两个 16 位普通数据寄存器地址不能一样。否则会引起数据出错。

3.6 整型数据位逻辑运算操作指令

MOTEC 智能驱动器内置可编程控制器支持 16 位和 32 位数据位逻辑运算操作指令，该指令操作 D 寄存器和常数。

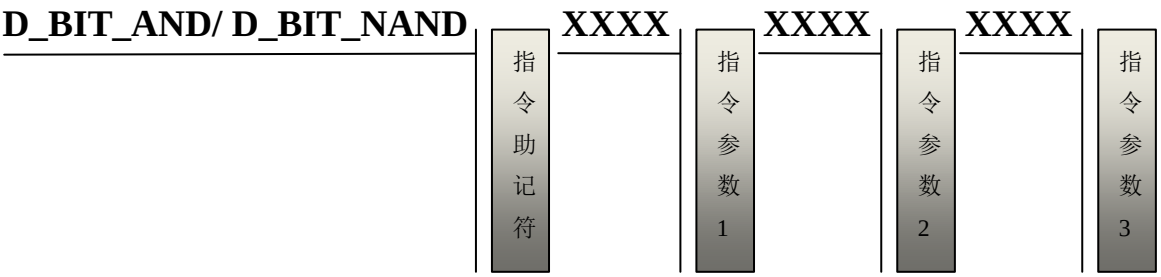
3.6.1 16 位整型数据位逻辑运算操作指令

D_BIT_AND/ D_BIT_NAND 16 位整型数据按位相与/16 位整型数据按位相与非

D_BIT_AND：两个 16 位整型数据相与，将逻辑运算结果存入指定的普通寄存器中。

D_BIT_NAND：两个 16 位整型数据相与非，将逻辑运算结果存入指定的普通寄存器中。

D_BIT_AND/ D_BIT_NAND 的指令的助记符语句结构为



指令参数 1 表示 16 位整型数据 D 寄存器地址；

指令参数 2 表示 16 位整型数据 D 寄存器地址或常数；

指令参数 3 表示逻辑运算结果存入的目标 D 寄存器地址。

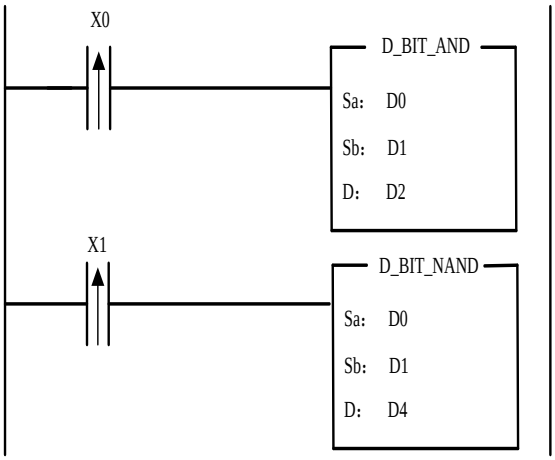
具体详见下表

指令参数 1	指令参数 2	指令参数 3
普通数据寄存器地址（Dn）	16 位常数(k)	输出寄存器地址
普通数据寄存器地址（Dn）	普通数据寄存器地址(Dm)	(Da)

目标寄存器只能存放 16 位数据，并且在程序运算过程中是以 16 位数据来运算的，所以该指令的最大取值范围是-32768～32767。

例如

梯形图 助记符




```
LD_R          X0
D_BIT_AND     FUNC
              1
              D0
              D1
              D2
LD_R          X1
D_BIT_NAND    FUNC
              1
              D0
              D1
              D4
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
D_BIT_AND	FUNC	将 16 位普通数据寄存器 D0 中存放的数据与 16 位普通数据寄存器 D1 中的数据相与，运算结果存入普通数据寄存器 D2 中
	1	
	D0	
	D1	
	D2	
LD_R	X1	检测继电器 X1 上升沿
D_BIT_NAND	FUNC	将 16 位普通数据寄存器 D0 中存放的数据与寄存器 D1 存放的数据相与非，运算结果存入 16 位普通数据寄存器 D4 中
	1	
	D0	
	D1	
	D4	

程序说明：该程序当检测继电器 X0 上升沿，将寄存器 D0 中的数据与寄存器 D1 中的数据相与，运算结果存入寄存器 D2 中，当检测继电器 X1 上升沿，将寄存器 D0 的数据与寄存器 D1 中的数据相与非，运算结果存入寄存器 D4 中。

例如下表

假设 D0 中存放的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	1	0	0	1	0	1	0	1

D1 中存放的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	0	1	0	1	1	0	0	1	1	1	0

对于上面程序中 16 位数据相与，经过运算以后，将结果存放到 D2 寄存器内，那么 D2 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0

对于上面程序中 16 位数据相与非，经过运算以后，将结果存放到 D4 寄存器内，那么 D4 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1

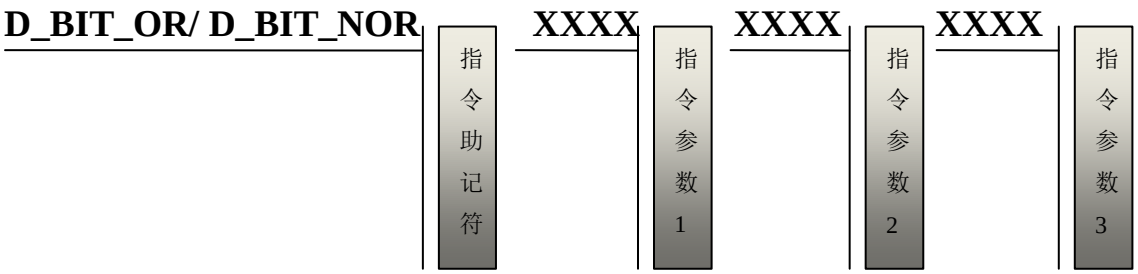
使用限制：D_BIT_AND/ D_BIT_NAND 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器或常数。

D_BIT_OR/ D_BIT_NOR 16 位整型数据按位相或/16 位整型数据按位相或非

D_BIT_OR：对一个数据寄存器内的数据进行相或操作，相或的对象可以是 16 位数据常数或者 D 寄存器内部的数据，将运算结果存入指定的普通寄存器中。

D_BIT_NOR：对一个数据寄存器内的数据进行相或非操作，相或非的对象可以是 16 位数据常数或者 D 寄存器内部的数据，将运算结果存入指定的普通寄存器中。

D_BIT_OR/ D_BIT_NOR 的指令的助记符语句结构为



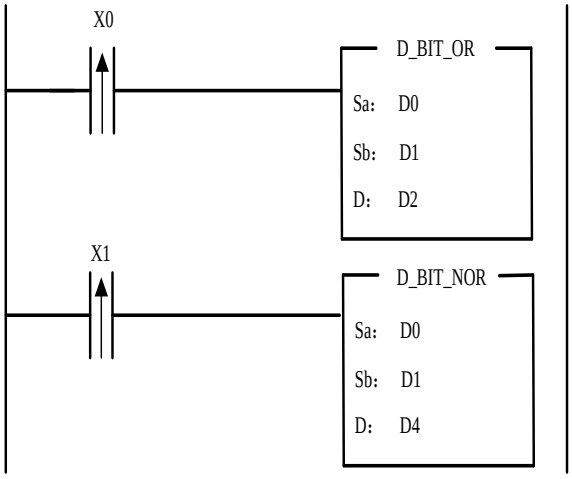
指令参数 1 表示 16 位数据 D 寄存器地址；

指令参数 2 表示 16 位数据 D 寄存器地址或常数；

指令参数 3 表示逻辑运算结果存入的目标 D 寄存器地址。

例如

梯形图



助记符

```
LD_R          X0
D_BIT_OR      FUNC
              1
              D0
              D1
              D2
LD_R          X1
D_BIT_NOR     FUNC
              1
              D0
              D1
              D4
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
D_BIT_OR	FUNC	将 16 位普通数据寄存器 D0 中存放的数据与
	1	16 位普通数据寄存器 D1 中的数据相或，运
	D0	算结果存入普通数据寄存器 D2 中
	D1	
	D2	
LD_R	X1	检测继电器 X1 上升沿
D_BIT_NOR	FUNC	将 16 位普通数据寄存器 D0 中存放的数据与
	1	寄存器 D1 存放的数据相或非，运算结果存
	D0	入 16 位普通数据寄存器 D4 中
	D1	
	D4	

程序说明：该程序当检测继电器 X0 上升沿，将寄存器 D0 中的数据与寄存器 D1 中的数据相或，运算结果存入寄存器 D2 中，当检测继电器 X1 上升沿，将寄存器 D0 的数据与寄存器 D1 中的数据相或非，运算结果存入寄存器 D4 中。

例如下表

假设 D0 中存放的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0

D1 中存放的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	1	1	0	0	0	1	0	0

对于上面程序中 16 位数据相或，经过运算以后，将结果存放到 D2 寄存器内，那么 D2 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	0	1	1	0	1	0	1	0	0

对于上面程序中 16 位数据相或非，经过运算以后，将结果存放到 D4 寄存器内，那么 D4 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	1	0	0	1	0	1	0	1	1

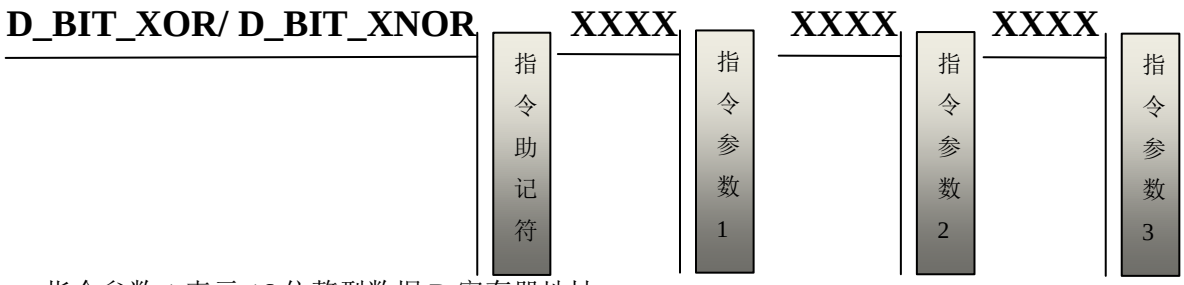
使用限制：D_BIT_OR/ D_BIT_NOR 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器或常数。

D_BIT_XOR/ D_BIT_XNOR 16 位整型数据按位相异或/16 位整型数据按位相同或

D_BIT_XOR：两个 16 位整型数据相异或，将逻辑运算结果存入指定的普通寄存器中。

D_BIT_XNOR：两个 16 位整型数据相同或，将逻辑运算结果存入指定的普通寄存器中。

D_BIT_XOR/ D_BIT_XNOR 的指令的助记符语句结构为



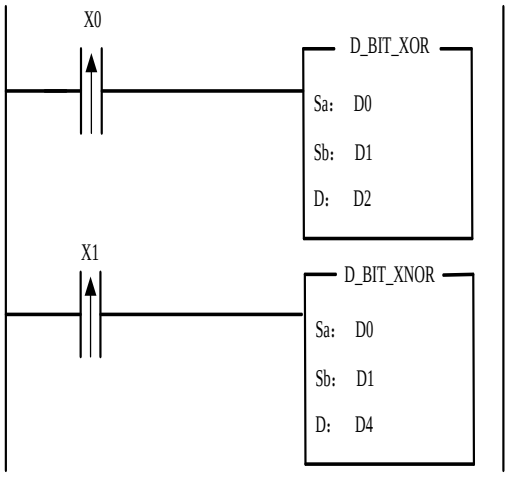
指令参数 1 表示 16 位整型数据 D 寄存器地址；

指令参数 2 表示 16 位整型数据 D 寄存器地址或常数；

指令参数 3 表示逻辑运算结果存入的目标 D 寄存器地址。

例如

梯形图



助记符

```
LD_R          X0
D_BIT_XOR     FUNC
              1
              D0
              D1
              D2
LD_R          X1
D_BIT_XNOR    FUNC
              1
              D0
              D1
              D4
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
D_BIT_XOR	FUNC	将 16 位普通数据寄存器 D0 中存放的数据与 16 位普通数据寄存器 D1 中的数据相异或，运算结果存入普通数据寄存器 D2 中
	1	
	D0	
	D1	
	D2	
LD_R	X1	检测继电器 X1 上升沿
D_BIT_XNOR	FUNC	将 16 位普通数据寄存器 D0 中存放的数据与寄存器 D1 存放的数据相同或，运算结果存入 16 位普通数据寄存器 D4 中
	1	
	D0	
	D1	
	D4	

程序说明：该程序当检测继电器 X0 上升沿，将寄存器 D0 中的数据与寄存器 D1 中的数据相异或，运算结果存入寄存器 D2 中，当检测继电器 X1 上升沿，将寄存器 D0 的数据与寄存器 D1 中的数据相同或，运算结果存入寄存器 D4 中。

例如下表

假设 D0 中存放的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0	1	1	0	1	0	1	0	1

D1 中存放的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	1

对于上面程序中 16 位数据相异或，经过运算以后，将结果存放到 D2 寄存器内，那么 D2 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0

对于上面程序中 16 位数据相同或，经过运算以后，将结果存放到 D4 寄存器内，那么 D4 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	1	0	1	1	1	1	1	1

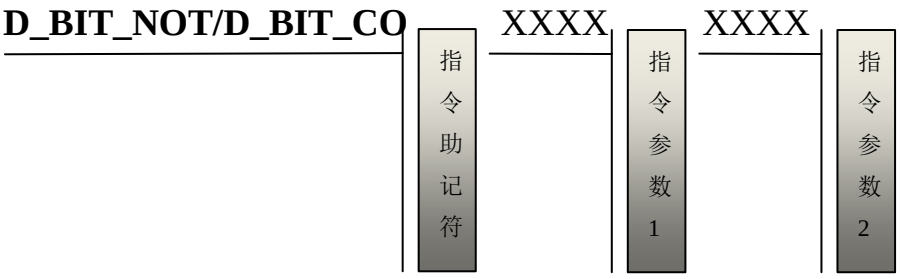
使用限制：D_BIT_XOR/ D_BIT_XNOR 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器或常数。

D_BIT_NOT/D_BIT_CO 16 位整型数据按位非/16 位整型数据按位求补

D_BIT_NOT：将一个 16 位普通寄存器中的数据按位非，运算结果存入指定的普通寄存器中。

D_BIT_CO：将一个 16 位普通寄存器中的数据按位求补码，运算结果存入指定的普通寄存器中。

D_BIT_NOT 的指令的助记符语句结构为

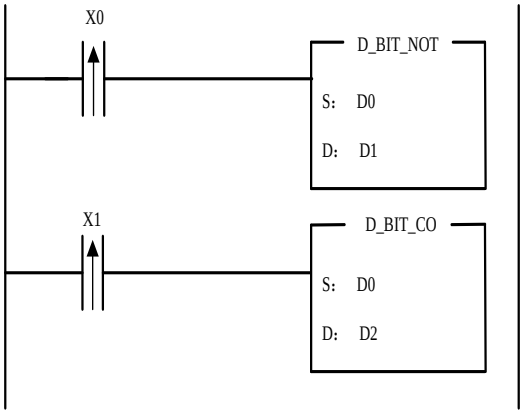


指令参数 1 表示 16 位数据 D 寄存器地址；

指令参数 2 表示 16 位数据运算后存入的指定目标 D 寄存器地址。

例如

梯形图



助记符

```
LD_R        X0
D_BIT_NOT   FUNC
            D0
            D1
LD_R        X1
D_BIT_CO    FUNC
            D0
            D2
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
D_BIT_NOT	FUNC	将 16 位普通数据寄存器 D0 中存放的数据按
	D0	位非，运算结果存入普通数据寄存器 D1 中
	D1	
LD_R	X1	检测继电器 X1 上升沿
D_BIT_CO	FUNC	将 16 位普通数据寄存器 D0 中存放的数据按
	D0	位求补码，运算结果存入 16 位寄存器 D2 中
	D2	

程序说明：该程序当检测继电器 X0 上升沿，将寄存器 D0 中存放的数据按位非，运算结果存入寄存器 D1 中，当检测继电器 X1 上升沿，将寄存器 D0 中的数据按位求补码，运算结果存入寄存器 D2 中。

例如下表

假设 D0 中存放的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0

对于上面程序中 16 位数据按位非，经过运算以后，将结果存放到 D1 寄存器内，那么 D1 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	1	0	0	1	1	0	1	1

对于上面程序中 16 位数据按位求补，经过运算以后，将结果存放到 D2 寄存器内，那么 D2 寄存器内的数据为：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0

使用限制：D_BIT_NOT/D_BIT_CO 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

注意：按位求补码，正数求补码与原码相同，负数求补码，符号位为 1，其余位为该数的绝对值的原码按位取反，然后整个数加 1。

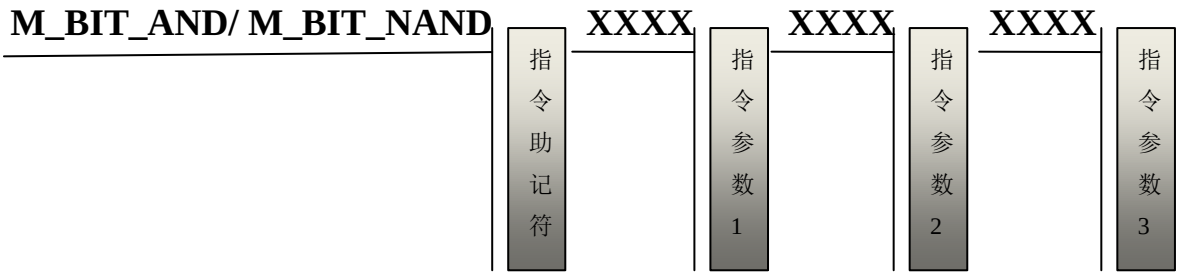
3.6.2 32 位整型数据位逻辑运算操作指令

M_BIT_AND/ M_BIT_NAND 32 位整型数据按位相与/32 位整型数据按位相与非

M_BIT_AND：两个 32 位数据相与，将逻辑运算结果存入指定的普通寄存器中。

M_BIT_NAND：两个 32 位数据相与非，将逻辑运算结果存入指定的普通寄存器中。

M_BIT_AND/ M_BIT_NAND 的指令的助记符语句结构为



指令参数 1 表示 32 位数据寄存器低 16 位的地址 Dn；

指令参数 2 表示 32 位数据寄存器低 16 位的地址 Dm 或常数 K；

指令参数 3 表示逻辑运算结果存入的 32 位目标数据寄存器的低 16 位地址 Da。

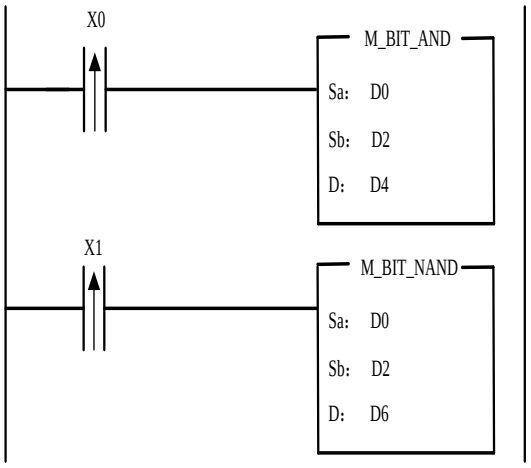
具体详见下表

指令参数 1	指令参数 2	指令参数 3
普通数据寄存器地址（Dn）	32 位常数(k)	输出寄存器地址
普通数据寄存器地址（Dn）	普通数据寄存器地址(Dm)	(Da)

目标寄存器存放运算结果的低 16 位数据，目标寄存器地址加一的寄存器存放运算结果高 16 位的数据，并且在程序运算过程中是以 32 位数据来运算的，所以该指令的最大取值范围是-2147483648～2147483647。

例如

梯形图



助记符

```
LD_R          X0
M_BIT_AND     FUNC
              1
              D0
              D2
              D4
LD_R          X1
M_BIT_NAND    FUNC
              1
              D0
              D2
              D6
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_BIT_AND	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数
	1	据与 32 位普通数据寄存器 D2，D3 存放的数
	D0	据相与，运算结果存入 32 位普通数据寄存器
	D2	D4，D5 中
	D4	
LD_R	X1	检测继电器 X1 上升沿
M_BIT_NAND	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数
	1	据与 32 位普通数据寄存器 D2，D3 存放的数
	D0	据相与非，运算结果存入 32 位寄存器 D6，
	D2	D7 中
	D6	

程序说明：该程序检测继电器 X0 上升沿时，将 32 位寄存器 D0，D1 中的数据与 32 位寄存器 D2，D3 中的数据相与，结果存入寄存器 D4，D5 中；当检测继电器 X1 上升沿时，将 32 位寄存器 D0，D1 中的数据与 32 位寄存器 D2，D3 中的数据相与非，结果存入寄存器 D6，D7 中。

例如下表

假设 D0 和 D1 存放的数据为

D1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1

D0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0	0	1	0	1	0	1	0	1

——D0 作为 32 位数据的低 16 位，D1 作为 32 位数据的高 16 位

寄存器 D2 和 D3 存放的数据为

D3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	0	0	0	1	0	1	1	0

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	0	1	0	1	0	1	0	0

——D2 作为 32 位数据的低 16 位，D3 作为 32 位数据的高 16 位

对于上面程序中 32 位数据相与，经过运算以后，将结果存放到 D4，D5 寄存器内，那么 D4，D5 寄存器内的数据为：

D5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0

——D4 作为 32 位数据的低 16 位，D5 作为 32 位数据的高 16 位

对于上面程序中 32 位数据相与非，经过运算以后，将结果存放到 D6，D7 寄存器内，那么 D6，D7 寄存器内的数据为：

D7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1

D6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	1	0	1	0	1	0	1	1

——D6 作为 32 位数据的低 16 位，D7 作为 32 位数据的高 16 位

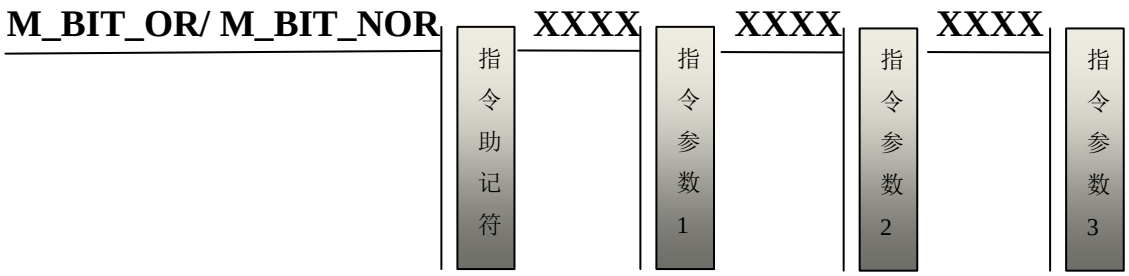
使用限制：M_BIT_AND/ M_BIT_NAND 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器或常数。

M_BIT_OR/ M_BIT_NOR 32 位整型数据按位相或/32 位整型数据按位相或非

M_BIT_OR：两个 32 位整型数据相或，将逻辑运算结果存入指定的普通寄存器中。

M_BIT_NOR：两个 32 位整型数据相或非，将逻辑运算结果存入指定的普通寄存器中。

M_BIT_OR/ M_BIT_NOR 的指令的助记符语句结构为



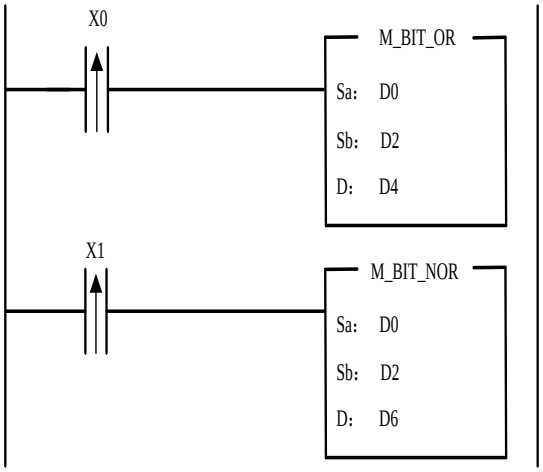
指令参数 1 表示 32 位数据寄存器低 16 位的地址；

指令参数 2 表示 32 位数据寄存器低 16 位的地址或常数；

指令参数 3 表示逻辑运算结果存入的 32 位目标数据寄存器的低 16 位地址。

例如

梯形图



助记符

```
LD_R        X0
M_BIT_OR    FUNC
            1
            D0
            D2
            D4
LD_R        X1
M_BIT_NOR   FUNC
            1
            D0
            D2
            D6
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_BIT_OR	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数据与 32 位普通数据寄存器 D2，D3 存放的数据相或，运算结果存入 32 位普通数据寄存器 D4，D5 中
	1	
	D0	
	D2	
	D4	
LD_R	X1	检测继电器 X1 上升沿
M_BIT_NOR	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数据与 32 位普通数据寄存器 D2，D3 存放的数据相或非，运算结果存入 32 位寄存器 D6，D7 中
	1	
	D0	
	D2	
	D6	

程序说明：该程序检测继电器 X0 上升沿时，将 32 位寄存器 D0，D1 中的数据与 32 位寄存器 D2，D3 中的数据相或，结果存入寄存器 D4，D5 中；当检测继电器 X1 上升沿时，将 32 位寄存器 D0，D1 中的数据与 32 位寄存器 D2，D3 中的数据相或非，结果存入寄存器 D6，D7 中。

例如下表

假设 D0 和 D1 存放的数据为

D1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	1	1	0	0	0	0	1	0	1

D0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0	0	1	1	1	0	0	1	0

——D0 作为 32 位数据的低 16 位，D1 作为 32 位数据的高 16 位

假设 D2 和 D3 存放的数据为

D3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	0	0	1	0	1	1	0	1

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	1	0	0	1	0	0	0	1	0

——D2 作为 32 位数据的低 16 位，D3 作为 32 位数据的高 16 位

对于上面程序中的 32 位数据相或，经过运算以后，将结果存放到 D4 和 D5 寄存器内，那么 D4 和 D5 寄存器内数据为

D5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	1	0	1	0	1	1	0	1

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	1	0	1	1	1	0	0	1	0

——D4 作为 32 位数据的低 16 位，D5 作为 32 位数据的高 16 位

对于上面程序中的 32 位数据相或非，经过运算以后，将结果存放到 D6 和 D7 寄存器内，那么 D6 和 D7 寄存器内数据为

D7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0

D6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1

——D6 作为 32 位数据的低 16 位，D7 作为 32 位数据的高 16 位

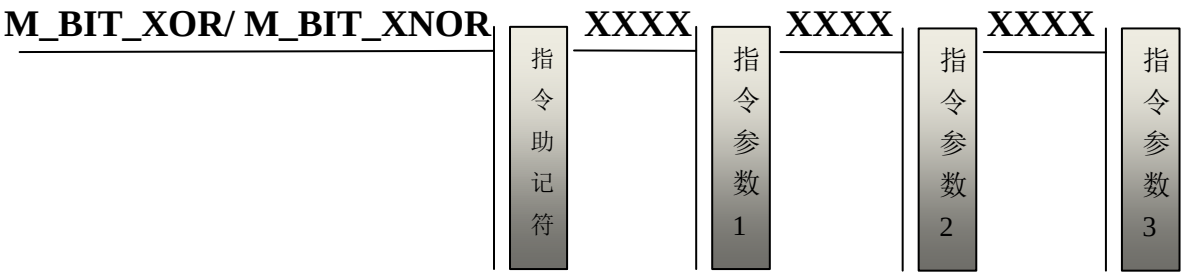
使用限制：M_BIT_OR/ M_BIT_NOR 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器或常数。

M_BIT_XOR/ M_BIT_XNOR 32 位整型数据按位相异或/32 位整型数据按位相同或

M_BIT_XOR：两个 32 位整型数据相异或，将逻辑运算结果存入指定的普通寄存器中。

M_BIT_XNOR：两个 32 位整型数据相同或，将逻辑运算结果存入指定的普通寄存器中。

M_BIT_XOR/ M_BIT_XNOR 的指令的助记符语句结构为



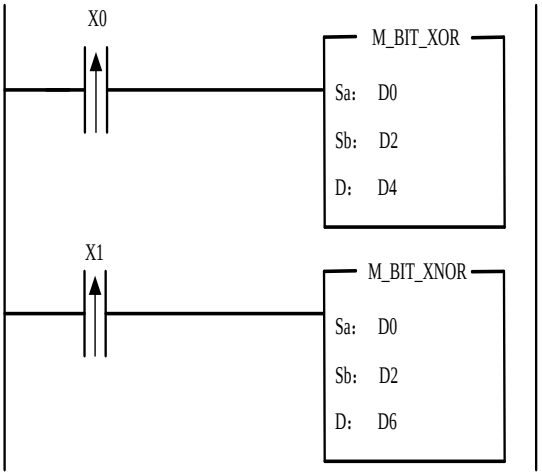
指令参数 1 表示 32 位数据寄存器低 16 位的地址；

指令参数 2 表示 32 位数据寄存器低 16 位的地址或常数；

指令参数 3 表示逻辑运算结果存入的 32 位目标数据寄存器的低 16 位地址。

例如

梯形图



助记符

```
LD_R          X0
M_BIT_XOR     FUNC
              1
              D0
              D2
              D4
LD_R          X1
M_BIT_XNOR    FUNC
              1
              D0
              D2
              D6
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_BIT_XOR	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数据与 32 位普通数据寄存器 D2，D3 存放的数据相异或，运算结果存入 32 位普通数据寄存器 D4，D5 中
	1	
	D0	
	D2	
	D4	
LD_R	X1	检测继电器 X1 上升沿
M_BIT_XNOR	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数据与 32 位普通数据寄存器 D2，D3 存放的数据相同或，运算结果存入 32 位寄存器 D6，D7 中
	1	
	D0	
	D2	
	D6	

程序说明：该程序检测继电器 X0 上升沿时，将 32 位寄存器 D0，D1 中的数据与 32 位寄存器 D2，D3 中的数据相异或，运算结果存入寄存器 D4，D5 中；当检测继电器 X1 的上升沿时，将 32 位寄存器 D0，D1 中的数据与 32 位寄存器 D2，D3 中的数据相同或，运算结果存入寄存器 D6，D7 中。

例如下表

假设 D0 和 D1 存放的数据为

D1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0	1	0	0	1	0	1	1	1

D0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0	1	0	1	0	0	1	1	0

——D0 作为 32 位数据的低 16 位，D1 作为 32 位数据的高 16 位

假设 D2 和 D3 存放的数据为

D3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	0	1	0	0	1	0	0	1

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	1	0	1	0	1	0	1	0

——D2 作为 32 位数据的低 16 位，D3 作为 32 位数据的高 16 位

对于上面程序中的 32 位数据相异或，经过运算以后，将结果存放到 D4 和 D5 寄存器内，那么 D4 和 D5 寄存器内数据为

D5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	0	1	1	0	1	1	1	1	0

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0

——D4 作为 32 位数据的低 16 位，D5 作为 32 位数据的高 16 位

对于上面程序中的 32 位数据相同或，经过运算以后，将结果存放到 D6 和 D7 寄存器内，那么 D6 和 D7 寄存器内数据为

D7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	0	0	1	0	0	0	0	1

D6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	1	1	1	1	0	0	1	1

——D6 作为 32 位数据的低 16 位，D7 作为 32 位数据的高 16 位

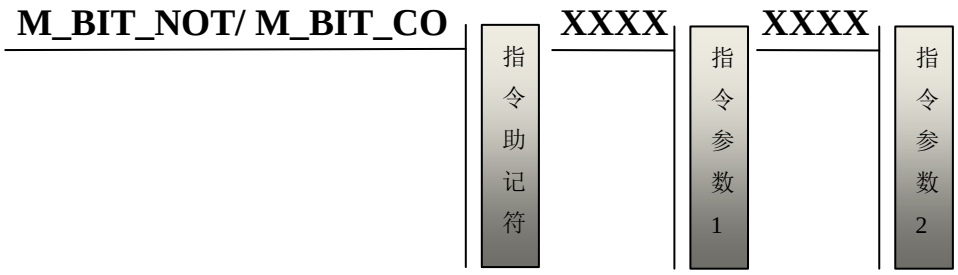
使用限制：M_BIT_XOR/ M_BIT_XNOR 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器或常数。

M_BIT_NOT/ M_BIT_CO 32 位整型数据按位非/32 位整型数据按位求补

M_BIT_NOT：将一个 32 位普通寄存器中的数据按位非，运算结果存入指定的普通寄存器中。

M_BIT_CO：将一个 32 位普通寄存器中的数据按位求补，运算结果存入指定的普通寄存器中。

M_BIT_NOT/ M_BIT_CO 的指令的助记符语句结构为

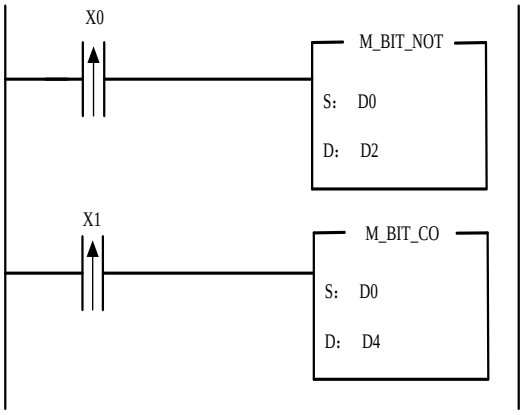


指令参数 1 表示 32 位数据寄存器的低 16 位地址；

指令参数 2 表示 32 位数据运算后存入的指定目标寄存器的低 16 位地址。

例如

梯形图



助记符

```
LD_R          X0
M_BIT_NOT     FUNC
              D0
              D2
LD_R          X1
M_BIT_CO      FUNC
              D0
              D4
```

指令解释（助记符部分）

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
M_BIT_NOT	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数
	D0	据按位非，运算结果存入 32 位普通数据寄存
	D2	器 D2，D3 中
LD_R	X1	检测继电器 X1 上升沿
M_BIT_CO	FUNC	将 32 位普通数据寄存器 D0，D1 中存放的数
	D0	据按位求补码，运算结果存入 32 位寄存器
	D4	D4，D5 中

程序说明：该程序检测继电器 X0 上升沿，将寄存器 D0，D1 中存放的数据按位非，结果存入寄存器 D2，D3 中，当检测继电器 X1 上升沿，将寄存器 D0，D1 中的数据按位求补码，结果存入寄存器 D4，D5 中。

例如下表

假设 D0 和 D1 存放的数据为：

D1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	0	0	0	1	1	0	1	1	0	1	0

D0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	1	0	0	0	0	1	0	0	0

——D0 作为 32 位数据的低 16 位，D1 作为 32 位数据的高 16 位

对于上面程序中的 32 位数据按位求非，经过运算以后，将结果存放到 D2 和 D3 寄存器内，那么 D2 和 D3 寄存器内数据为

D3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	1	1	0	0	1	0	0	1	0	1

D2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	1	1	1	1	0	1	1	1

——D2 作为 32 位数据的低 16 位，D3 作为 32 位数据的高 16 位

对于上面程序中的 32 位数据按位求补码，经过运算以后，将结果存放到 D4 和 D5 寄存器内，那么 D4 和 D5 寄存器内数据为

D5

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	0	0	1	0	0	1	0	1

D4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	1	1	1	1	1	0	0	0

——D4 作为 32 位数据的低 16 位，D5 作为 32 位数据的高 16 位

使用限制：M_BIT_NOT/ M_BIT_CO 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且该指令操作对象只能是 D 寄存器。

注意：按位求补码，正数求补码与原码相同，负数求补码，符号位为 1，其余位为该数的绝对值的原码按位取反，然后整个数加 1。可以通过该指令得出一个有符号的负数在控制器内部的保存形式。

3.7 浮点数操作指令

MOTEC 智能驱动器内置可编程控制器支持单精度浮点数操作，可以对浮点数进行逻辑运算和算术运算，并且可以将浮点数与整型等数据相互转换，

MOTEC 智能驱动器内置可编程控制器关于浮点数的运算都是以二进制浮点数来实现的，如果计算中有整型或者十进制浮点数进行操作，必须先转换成二进制浮点数来进行操作。

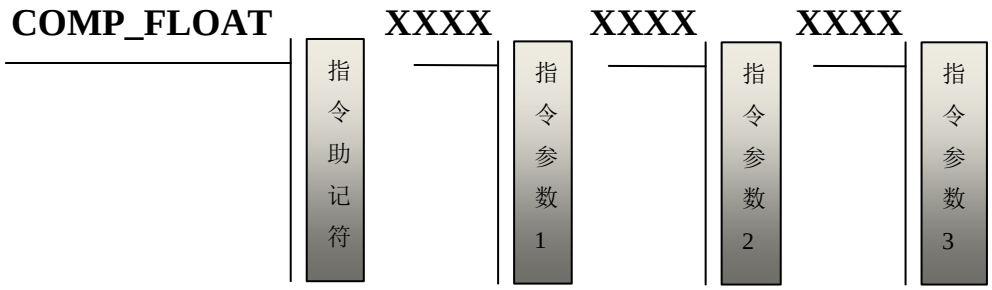
3.7.1 浮点数数据比较指令

浮点数比较操作指令

COMP_FLOAT: 将两个浮点数据进行比较

COMP_FLOAT：该指令将两个浮点数进行比较，可判断两个浮点数相等操作/大于等于操作/小于等于操作/大于操作/小于操作/不等于操作。

COMP_FLOAT 的指令的助记符语句结构为



指令参数 1 表示参与逻辑运算的浮点数存放的低 16 位寄存器的地址；

指令参数 2 表示参与逻辑运算的第二个参数，浮点数常数的大小或存放浮点数的低 16 位寄存器地址。

指令参数 3 表示输出辅助继电器。

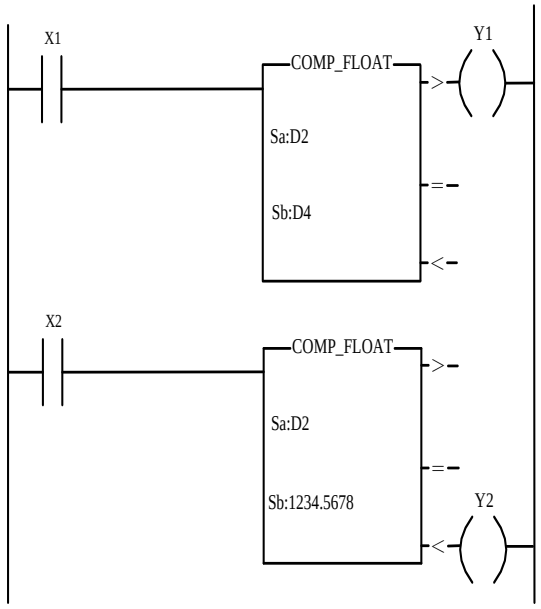
以浮点数是否大于等于逻辑运算为例，具体见下表

指令参数 1	指令参数 2	运算结果	
		结果	输出
普通数据寄存器地址（Dn）组成浮点数 Fn	浮点数参数常数 Fk	Fn ＞ Fk	“＞”端置 ON
		Fn ＝ Fk	“＝”端置 ON
		Fn ＜ Fk	“＜”端置 ON
普通数据寄存器地址（Dn）组成浮点数 Fn	普通数据寄存器地址（Dm）组成浮点数 Fm	Fn ＞ Fm	“＞”端置 ON
		Fn ＝ Fm	“＝”端置 ON
		Fn ＜ Fm	“＜”端置 ON

单精度浮点数在驱动器内部全部都是以 32 位数据格式存储的，指令参数 1 和指令参数 2 内的寄存器地址是 32 位数据的低 16 位存放的地址，由该地址存放的数据和该地址加 1 存放的数据组合成一个 32 位数据，用来表示一个浮点数的值。

例如

梯形图



助记符

```
LD                  X1
FLOAT_COMPARE   FUNC
                    1
                    D2
                    D4
                    Y1
LD                  X2
FLOAT_COMPARE   FUNC
                    0
                    D2
                 1234.5678
                    Y2
```

指令解释(助记符部分)

指令内容		指令说明
LD	X1	装载输入继电器 X1 的状态
FLOAT_COMPARE	FUNC	普通数据寄存器 D2,D3 存放的浮点数和普通
	1	数据寄存器 D4,D5 存放的浮点数做大于比较，
	D2	逻辑运算比较结果输出到输出继电器 Y1
	D4	
	Y1	
LD	X2	装载输入继电器 X2 的状态
FLOAT_COMPARE	FUNC	普通数据寄存器 D2,D3 存放的浮点数和浮点
	0	数常数 1234.5678 做小于的比较，逻辑运算比
	D2	较结果输出到输出继电器 Y2
	1234.5678	
	Y2	

程序说明：该程序当 X1 为 ON 时，就会判断 D2，D3 存放的浮点数是不是大于 D4，D5 存放的浮点数，并输出到 Y1；当 X2 为 ON 时，就会判断 D2，D3 存放的浮点数是不是小于 1234.5678，并输出到 Y2。

假设 D2,D3 组成的浮点数为 4321.8765，D4，D5 组成的浮点数为 1234.8765，那么 X1 和 X2 都为 ON 时，

4321.8765>=1234.8765 ; Y1 输出 ON；

4321.8765 > 1234.5678 ; Y2 输出 OFF；

使用限制：COMP_FLOAT 指令不能放在母线 的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器。

3.7.2 浮点数算术运算指令

ADD _FLOAT / SUB _FLOAT / MUL _FLOAT/ DIV_ FLOAT:

浮点数据加法操作/浮点数据减法操作/浮点数据乘法操作/浮点数据除法操作

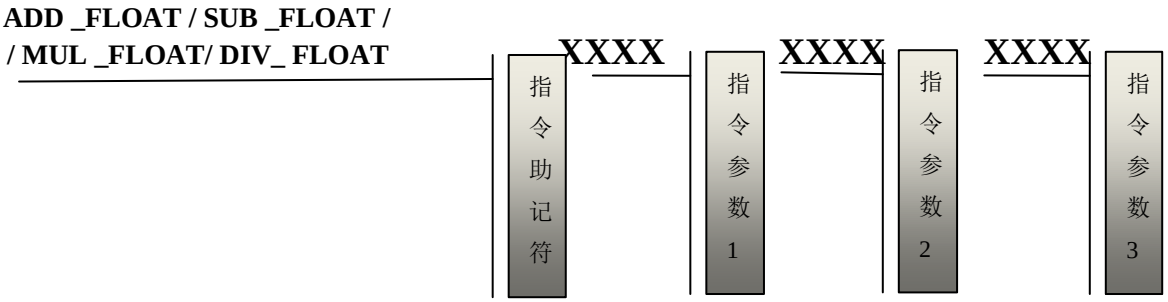
ADD _FLOAT:两个浮点数相加，并将结果存入指定的普通数据寄存器中；

SUB _FLOAT:两个浮点数相减，并将结果存入指定的普通数据寄存器中；

MUL _FLOAT:两个浮点数相乘，并将结果存入指定的普通数据寄存器中；

DIV_ FLOAT:两个浮点数相除，并将结果存入指定的普通数据寄存器中；

ADD _FLOAT / SUB _FLOAT / MUL _FLOAT/ DIV_ FLOAT 的指令的助记符语句结构为



指令参数 1 表示参与算术运算的浮点数存放的低 16 位寄存器的地址；

指令参数 2 表示参与算术运算的第二个参数,浮点数常数的大小或存放浮点数的低 16 位寄存器地址

指令参数 3 表示算术运算结果的存放的目标寄存器地址。

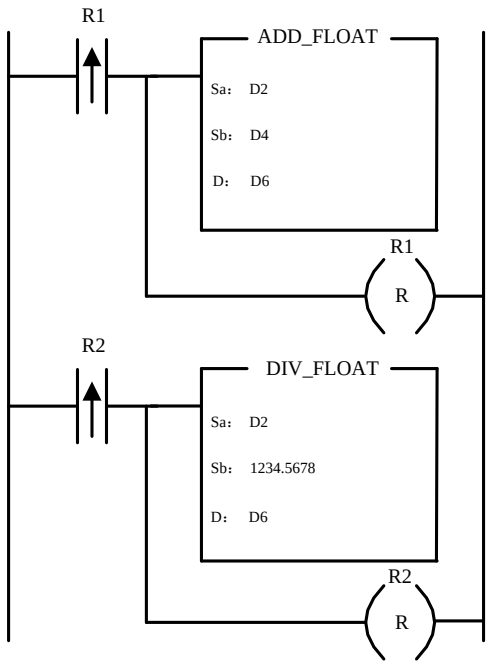
具体详见下表

指令参数 1	指令参数 2	指令参数 3	运算结果			
			加法操作	减法操作	乘法操作	除法操作
普通数据寄存器地址（Dn） 组成浮点数 Fn	浮点数参数 Fk	输出寄存器地址(Da) 组成浮点数 Fa	Fa=Fn+Fk	Fa=Fn-Fk	Fa=Fn*Fk	Fa=Fn/Fk
普通数据寄存器地址（Dn） 组成浮点数 Fn	普通数据寄存器地址（Dm） 组成浮点数 Fm		Fa=Fn+Fm	Fa=Fn-Fm	Fa=Fn*Fm	Fa=Fn/Fm

单精度浮点数在驱动器内部全部都是以 32 位数据格式存储的，指令参数 1 和指令参数 2 内的寄存器地址是 32 位数据的低 16 位存放的地址，由该地址存放的数据和该地址加 1 存放的数据组合成一个 32 位数据，用来表示一个浮点数的值。

例如

梯形图



助记符

```
LD_R          R1
ADD_FLOAT     FUNC
              1
              D2
              D4
              D6
RESET         R1
LD_R          R2
DIV_FLOAT     FUNC
              0
              D2
              1234.5678
              D6
RESET         R2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测辅助继电器 R1 的上升沿
ADD_FLOAT	FUNC	将 D2、D3 寄存器内的浮点数与寄存器
	1	D4、D5 内的浮点数相加，结果存入寄存
	D2	器 D6、D7 中
	D4	
	D6	
RESET	R1	清除继电器 R1
LD_R	R2	检测辅助继电器 R2 的上升沿
DIV_FLOAT	FUNC	将 D2、D3 寄存器内的浮点数与浮点数常
	0	数 1234.5678 相除，结果存入 D6、D7 中
	D2	
	1234.5678	
	D6	
RESET	R2	清除继电器 R2

程序说明：检测到 R1 的上升沿时，会触发 D2、D3 寄存器内的浮点数与 D4、D5 内的浮点数的相加操作，假设 D2,D3 内的浮点数为 1234.5678，D4、D5 内的浮点数为 8765.4321，

1234.5678 + 8765.4321 = 9999.9999

那么 D6 和 D7 组成的浮点数就为 9999.9999

检测到 R2 的上升沿，会触发 D2、D3 寄存器内的浮点数与浮点数常数 1234.5678 的相除操作，

1234.5678/1234.5678 = 1;

那么 D6 和 D7 组成的浮点数就为 1，注意，即使是一个正整数，在驱动器内部，也是以浮点数的形式来存储的。

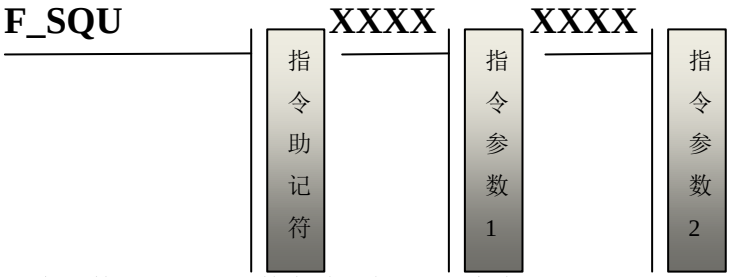
使用限制：ADD _FLOAT / SUB _FLOAT / MUL _FLOAT/ DIV_ FLOAT 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器。

3.7.3 浮点数数值运算

F_SQU 浮点数的开方

F_SQU：对一个普通寄存器中的浮点数进行开平方，运算的结果存入指定的普通寄存器中。

F_SQU 的指令的助记符语句结构为



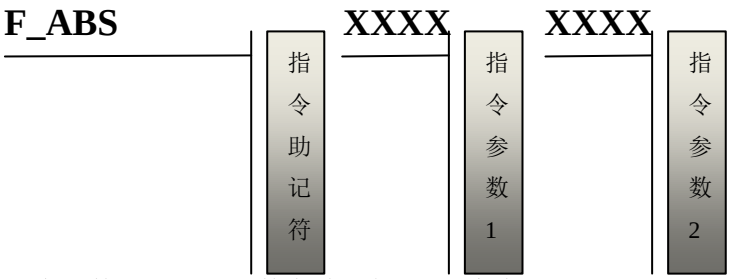
指令参数 1 表示浮点数存放的低 16 位寄存器地址；

指令参数 2 表示目标寄存器的低 16 位地址。

F_ABS 浮点数求绝对值

F_ABS：对一个普通寄存器中的浮点数求绝对值，运算的结果存入指定的普通寄存器中。

F_ABS 的指令的助记符语句结构为

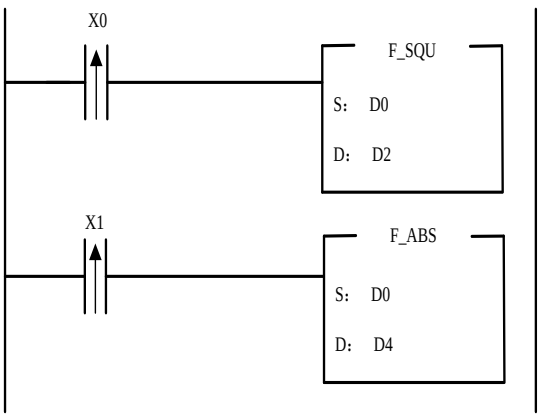


指令参数 1 表示浮点数存放的低 16 位寄存器地址；

指令参数 2 表示目标寄存器的低 16 位地址。

例如

梯形图 助记符




```
LD_R     X0
F_SQU    FUNC
         D0
         D2
LD_R     X1
F_ABS    FUNC
         D0
         D4
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X0	检测继电器 X0 上升沿
F_SQU	FUNC	将寄存器 D0，D1 存放的浮点数据开平方，
	D0	运算结果存入普通数据寄存器 D2，D3 中
	D2	
LD_R	X1	检测继电器 X1 上升沿
F_ABS	FUNC	将寄存器 D0，D1 中存放的浮点数求绝对值，
	D0	运算结果存入普通数据寄存器 D4，D5 中
	D4	

程序说明：当检测继电器 X0 上升沿，假设寄存器 D0，D1 中存放的数据为 987654.32125，那么目标寄存器 D2，D3 存入的数据是 993.8079833984，当检测继电器 X1 上升沿，那么目标寄存器 D4，D5 中存入的数据为 987654.32125。

使用限制：F_SQU/F_ABS 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器。

注意：浮点数开方只能为正数，如果为负数，会引起数据错误。

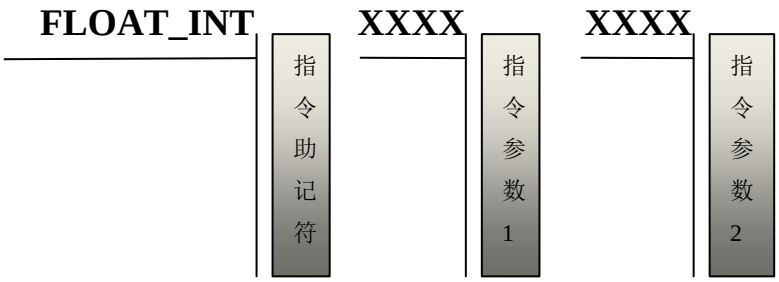
3.7.4 浮点数与整型转换存储操作指令

MOTEC 智能驱动器内置可编程控制器的浮点数逻辑运算和算术运算都是以二进制形式存在的。可以使用浮点数转换的指令将浮点数与整型数据或者十进制浮点数之间相互转换。

FLOAT_INT：浮点数转换成整数形式

FLOAT_INT：按照四舍五入的原则，将寄存器内的浮点数转换成整型数据，如果需要转换 16 位整型数据，那么只存储在一个普通数据寄存器中；如果需要转换成 32 整型数据，那么存放在连续两个的普通数据寄存器中。

FLOAT_INT 指令的助记符语句结构为

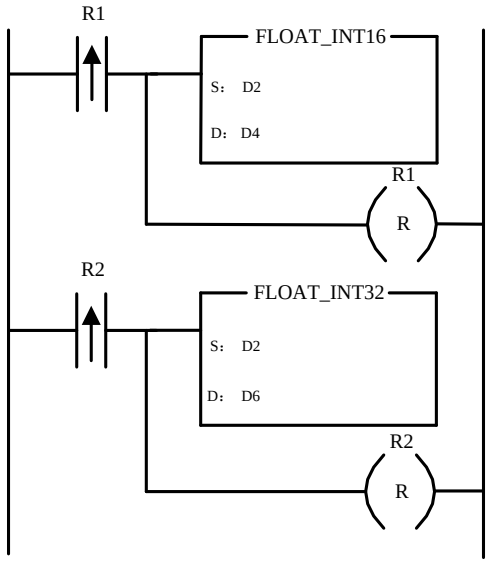


指令参数 1 表示需要转换的浮点数的寄存器地址；

指令参数 2 表示转换之后整型数据存放的地址，如果转化的结果是 16 位数据，那么结果就存放在该地址中，如果转换的结果是 32 位数据，那么该地址存放低 16 位数据，该地址加一的寄存器存放高 16 位数据；

例如

梯形图



指令解释(助记符部分)

指令内容

LD_R	R1
FLOAT_INT	FUNC
	0
	D2
	D4
RESET	R1
LD_R	R2
FLOAT_INT	FUNC
	1
	D2
	D6
RESET	R2

助记符

		指令说明
LD_R	R1	检测 R1 的上升沿
FLOAT_INT	FUNC	把 D2、D3 内存放的浮点数转换成 16 位整数
	0	并存入寄存器 D4 中
	D2	
	D4	
RESET	R1	清除继电器 R1
LD_R	R2	检测 R2 的上升沿
FLOAT_INT	FUNC	把 D2、D3 内存放的浮点数转换成 32 位整数
	1	并存入寄存器 D6、D7 中
	D2	
	D6	
RESET	R2	清除继电 R2

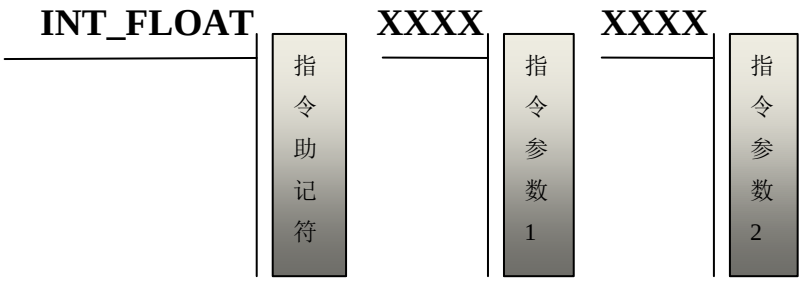
程序说明：检测到 R1 的上升沿，那么就将 D2,D3 存储的浮点数转换成 16 位整型数据存放在寄存器 D4 中，假设 D2、D3 内的数据为-1234.5678，那么 D4 = 0x84D3; 检测到 R2 的上升沿，那么就将那么就将 D2,D3 存储的浮点数转换成 32 位整型数据存放在 D6 和 D7 中，那么 D6 = 0x04D3，D7 = 0x8000；

使用限制：FLOAT_INT 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器，转换出来的整型数据是有符号数据。

INT_FLOAT：整数形式转换成浮点数

INT _FLOAT：可以将寄存器内的 16 位有符号整型数据或者 32 位有符号整型数据转换成浮点数，并且存入指定的普通数据寄存器中；

INT _FLOAT 指令的助记符语句结构为

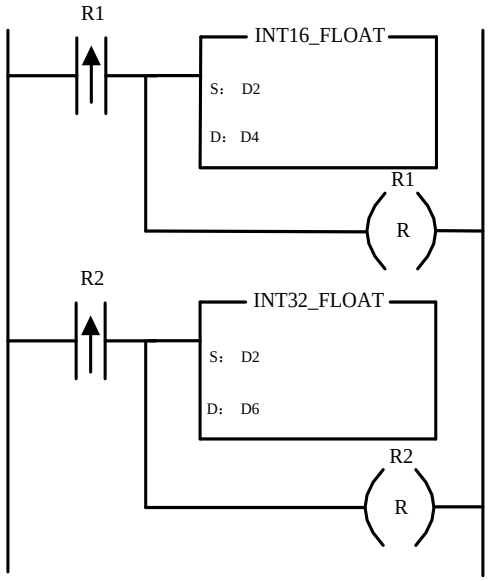


指令参数 1 表示需要转换的整型数据寄存器地址；

指令参数 2 表示转换之后的浮点数结果存放的普通数据寄存器地址。

例如

梯形图



助记符

LD_R	R1
INT_FLOAT	FUNC
	0
	D2
	D4
RESET	R1
LD_R	R2
INT_FLOAT	FUNC
	1
	D2
	D6
RESET	R2

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测 R1 的上升沿
INT_FLOAT	FUNC	把 D2 内存放的 16 位整型数转换成浮点数并存入寄存器 D4、D5 中
	0	
	D2	
	D4	
RESET	R1	清除继电器 R1
LD_R	R2	检测 R2 的上升沿
INT_FLOAT	FUNC	把 D2、D3 内存放的 32 位整型数转换成浮点数并存入寄存器 D6、D7 中
	1	
	D2	
	D6	
RESET	R2	清除继电器 R2

程序说明：检测到 R1 的上升沿，那么就将 D2 存储的 16 位整型数据转换成浮点数存放在寄存器 D4、D5 中，检测到 R2 的上升沿，那么就将 D2,D3 存储的 32 位整型数据转换成浮点数存放在 D6 和 D7 中；

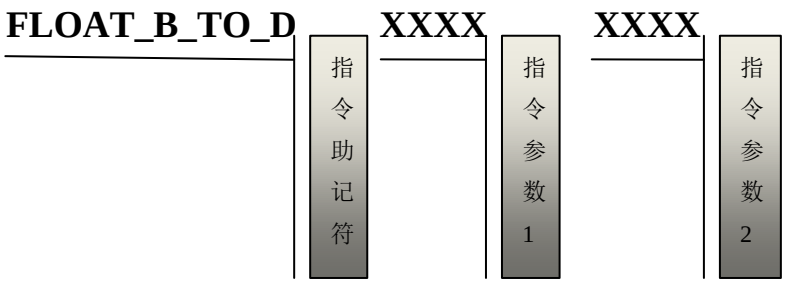
使用限制：INT_FLOAT 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器。并且指令会默认需要转换的整型数据是有符号的。

3.7.5 二进制浮点数与十进制浮点数相互转换

FLOAT_B_TO_D：二进制浮点数转换成十进制浮点数

FLOAT_B_TO_D：该指令可以将二进制浮点数转换成十进制浮点数，转换的对象是寄存器存储的浮点数，转换的结果是将十进制浮点数的指数部分存入目标寄存器，尾数部分存入目标地址的下一个寄存器。

FLOAT_B_TO_D 指令的助记符语句结构为

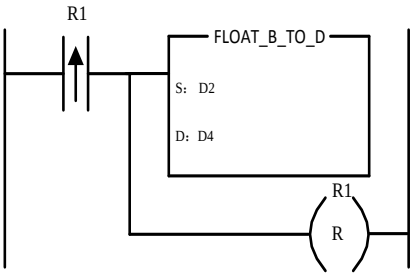


指令参数 1 表示需要转换的二进制浮点数存放的寄存器地址。

指令参数 2 表示转换之后的十进制浮点数结果存放的普通数据寄存器地址。

例如

梯形图



助记符

LD_R	R1
FLOAT_B_TO_D	FUNC
	D2
	D4
RESET	R1

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测 R1 的上升沿
FLOAT_B_TO_D	FUNC	将 D2、D3 存放的二进制浮点数转换成十进制浮点数存放在 D4 和 D5 中。
	D2	
	D4	
RESET	R1	清除继电器 R1

程序说明：检测到 R1 的上升沿后，会触发二进制浮点数向十进制浮点数的转换，假设 D2、D3 存放的浮点数是-123456.7890，那么在转换之后，由于有效数字的限制，那么 D4 = -1234，

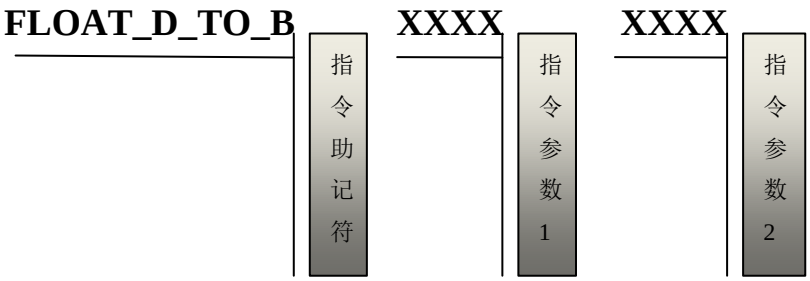
D5 = 2；

使用限制： FLOAT_B_TO_D 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器，转换出来的整型数据是有符号数据，并且十进制浮点数不能作为计算使用，只能用在给客户显示使用。

FLOAT_D_TO_B：十进制浮点数转换成二进制浮点数

FLOAT_D_TO_B：该指令可以将十进制浮点数转换成二进制浮点数，转换的对象是寄存 器存储的浮点数。

FLOAT_D_TO_B 指令的助记符语句结构为

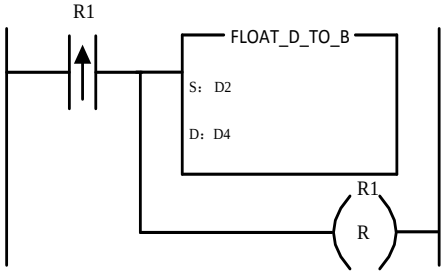


指令参数 1 表示需要转换的十进制浮点数存放的寄存器地址。

指令参数 2 表示转换之后的二进制浮点数结果存放的普通数据寄存器地址。

例如

梯形图



助记符

LD_R	R1
FLOAT_D_TO_B	FUNC
	1
	D2
	D4
RESET	R1

指令解释(助记符部分)

指令内容		指令说明
LD_R	R1	检测 R1 的上升沿
FLOAT_D_TO_B	FUNC	将 D2、D3 存放的十进制浮点数转换成二进制浮点数存放在 D4 和 D5 中。
	1	
	D2	
	D4	
RESET	R1	清除继电器 R1

程序说明：检测到 R1 的上升沿后，会触发十进制浮点数向二进制浮点数的转换，假设 D2=1234，D3 = -5，那么 D2 和 D3 存放的十进制浮点数是 0.01234；寄存器 D4，D5 中存入的二进制浮点数为 0.01234。

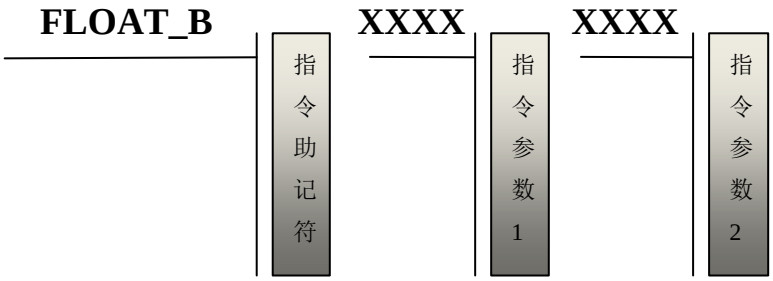
使用限制： FLOAT_D_TO_B 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器，转换出来的整型数据是有符号数据，并且十进制浮点数不能作为计算使用，只能用在给客户显示使用。

3.7.6 二进制浮点数的存储

FLOAT_B：将二进制浮点数存储到驱动器内部

FLOAT_B：该指令可以将一个浮点数常数存储到驱动器内部，存放的形式是 32 位数据。

FLOAT_B 指令的助记符语句结构为

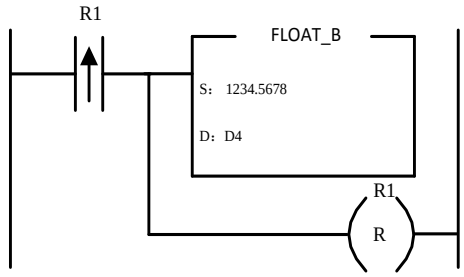


指令参数 1 表示需要存储的二进制浮点数常数；

指令参数 2 表示存放二进制浮点数的寄存器地址。

例如

梯形图



助记符

LD_R	R1
FLOAT_B	FUNC
	1234.5678
	D2
RESET	R1

指令解释(助记符部分)

指令内容		
LD_R	R1	指令说明
FLOAT_B	FUNC	检测 R1 的上升沿
	1234.5678	将浮点数常数 1234，5678 存放在驱动器内部
	D2	的 D 寄存器 D4、D5 中。
RESET	R1	清除继电器 R1

程序说明：检测到 R1 的上升沿后，会触发将浮点数常数 1234.5678 存放在 D4 和 D5 中。

使用限制：FLOAT_B 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行，并且浮点数指令的操作对象只能是 D 寄存器，不能是 P 寄存器，转换出来的整型数据是有符号数据。

3.8 数据存储和读取指令

MOTEC 智能驱动器内置可编程控制器支持对于参数的永久保存，可以使用数据的存储和读取指令，将程序使用过程中的数据保存在驱动器内部的 FLASH 中，保证数据在驱动器掉电以后不会丢失。

SAVE_DATA/GET_DATA:

保存寄存器的值到 FLASH/读取 FLASH 里面的值到寄存器

SAVE_DATA：该指令可以将当前寄存器的值保存到驱动器内部 FLASH，保存后的值掉电之后不会被消除。

GET_DATA：该指令可以将驱动器内部 FLASH 里面的数据读取到寄存器内，用来恢复最后一次保存的值。

SAVE_DATA/ GET_DATA 指令的助记符语句结构为

指令参数 1 表示需要存储/读取的寄存器数据,

0——存储/读取 P 寄存器值

3——存储/读取 R 寄存器值

7——存储/读取 TLD 寄存器值

8——存储/读取 CLD 寄存器值

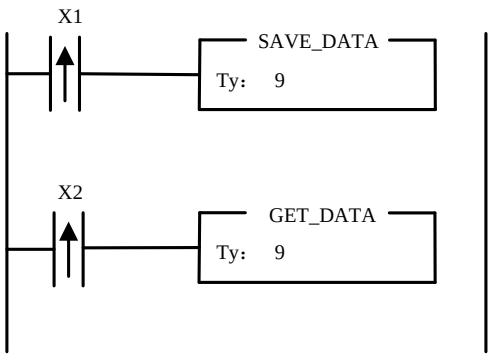
9——存储/读取 D 寄存器值

10——存储/读取所有参数

注意：由于输入继电器状态 X，输出继电器状态 Y，电机的当前状态继电器 S，定时器状态继电器 T，计数器状态继电器 C 都是随着程序的启动实时更新的状态，不能存储和读取到驱动器内部。

例如

梯形图 助记符




```
LD_R              X1
PLC_SAVE_DATA     FUNC
                  9
LD_R              X2
PLC_GET_DATA      FUNC
                  9
```

指令解释（助记符部分）

助记符内容		指令说明
LD_R	X1	检测 X1 的上升沿
PLC_SAVE_DATA	FUNC 9	将 D 寄存器的内容存放在内部 FLASH 中
LD_R	X2	检测 X2 的上升沿
PLC_GET_DATA	FUNC 9	将内部 FLASH 中存储的 D 寄存器的内容读出

程序说明：使用 X1 和 X2 可以完成 D 寄存器的存储和读取，防止掉电消失。为了方便使用，用户可以使用特殊辅助继电器 R2018，在程序开始运行的第一个周期读取上次保存的值。

注意：每次由内部 FLASH 读取的寄存器的值，只能读取最后一个保存在 FLASH 中的值。

使用限制：SAVE_DATA/ GET_DATA 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

注意：由于 flash 的存储次数有限制，在 10 万次以内，所以 SAVE_DATA 指令不能使用电平操作，不能每个周期都保存到 flah 否则会导致该寄存器的 flash 出错，所以 SAVE_DATA 指令只能通过沿操作。

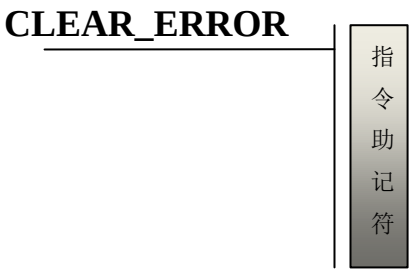
3.9 错误代码清除指令

MOTEC 智能驱动器内置可编程控制器可以保存内部的报警信号和错误代码用来方便判断，错误代码存储在驱动器内部寄存器 P12 中。在运行在可编程控制器方式时，可以使用指令来清除该错误代码。

CLEAR_ERROR：清除当前错误代码

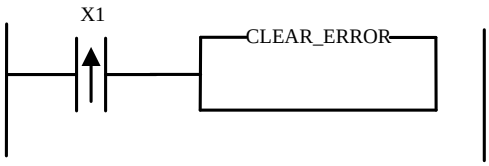
CLEAR_ERROR：用来清除驱动器内部寄存器 P12 内存储的错误代码。

CLEAR_ERROR 指令的助记符语句结构为



例如

梯形图



助记符

LD_R	X1
CLEAR_ERROR	FUNC

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
CLEAR_ERROR	FUNC	清除当前错误代码

程序说明：使用 X1 的上升沿可以清除当前错误代码。

注意：MOTEC 智能驱动器当遇到报警的时候，会自动将电机的使能去掉，使用该指令需要在排除故障之后，只能清除当前错误代码，并不能使能电机，需要使用运动控制指令的使能电机才可以。

使用限制：CLEAR_ERROR 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

第 4 章 运动控制指令

MOTEC 智能驱动器内置可编程控制器相对于普通的电机驱动器或者可编程控制器，拥有最大的特点就是驱动器和可编程控制器两者的有效结合，可以使用可编程控制器的强大逻辑运算和算术运算的能力，和驱动器的驱动能力，对电机进行各种操作。

MOTEC 智能驱动器内置可编程控制器集成了很多关于电机的运动控制指令，可以进行位置控制，速度控制，模拟量控制，回零操作，点动操作等，并且这些操作使用简单的可编程控制器指令来实现。

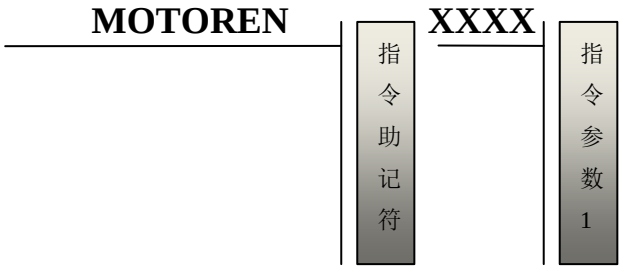
4.1 运动控制状态相关指令

运动控制状态的相关指令，可以操作驱动器的基本状态指令。

MOTOREN：使能释放电机指令

MOTOREN：该指令可以使能或者释放电机，在使能状态，电机励磁电流不为 0，电机锁轴；释放状态，电机励磁电流为 0，电机轴释放锁轴。

MOTOREN 指令的助记符语句结构为



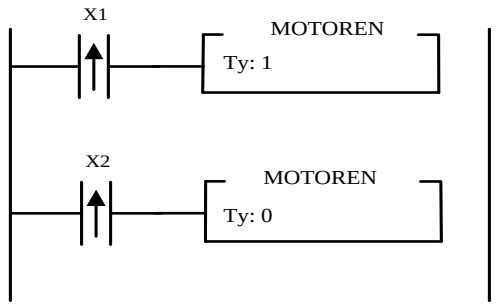
指令参数 1 表示具体的操作内容：

0 :释放电机

1:使能电机

例如

梯形图



助记符

LD_R	X1
MOTOR_ENABLE	FUNC 1
LD_R	X2
MOTOR_ENABLE	FUNC 0

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
MOTOR_ENABLE	FUNC 1	使能电机
LD_R	X2	检测 X2 的上升沿
MOTOR_ENABLE	FUNC 0	释放电机

程序说明：该程序可以使用 X1 的上升沿来使能电机，使用 X2 的上升沿来释放电机。

注意：使能电机和释放电机类似于继电器的置位和清除的操作，可以使用沿操作，这一个周期对进行使能/释放操作，下一个周期会继续保持，直到下一个使能/释放操作到来。

使用限制：MOTOREN 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

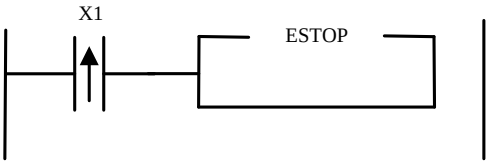
ESTOP：急停指令

ESTOP：紧急停止指令，在任何操作环境下，该指令都会紧急停止电机的转动。将当前的转速迅速降到 0，减速度可以通过设置 P 寄存器的急停减速度。

ESTOP 指令的助记符语句结构为

例如：

梯形图



助记符

LD_R X1

ESTOP FUNC

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
ESTOP	FUNC	电机急停

程序说明：该程序中，X1 可以作为电机的急停开关，X1 为 ON 时就停止。

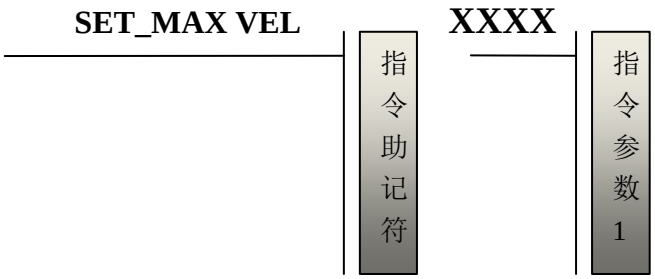
注意：急停指令可以可编程控制器程序中用于任何控制模式，位置模式、速度模式、或者模拟模式下，都可以使用，减速度可以通过设置 P 寄存器急停减速度来实现。

使用限制：ESTOP 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

SET_MAX VEL：设置位置模式下的最大转速

SET_MAX VEL：位置模式控制下，电机会根据轨迹规划做出相应的动作，最大转速代表在轨迹规划和位置控制实际运动过程中电机所能达到的最大转速。其中，驱动器内部的位置轨迹规划最大速度同样是该值。

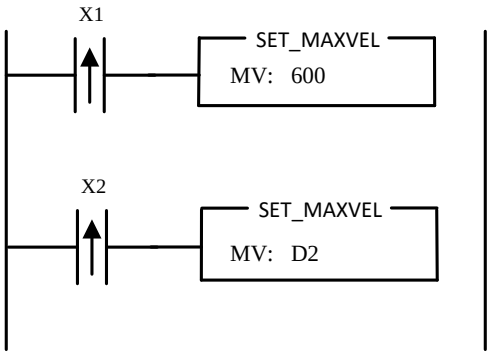
SET_MAX VEL 指令的助记符语句结构为



指令参数 1 表示最大转速的取值常数或者寄存器地址

例如：

梯形图



助记符

```
LD_R            X1
SET_VELOCITY    FUNC
                0
                600
LD_R            X2
SET_VELOCITY    FUNC
                1
                D2
```

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
SET_VELOCITY	FUNC	设置位置模式的最大转速是 600rpm
	0	
	600	
LD_R	X2	检测 X2 的上升沿
SET_VELOCITY	FUNC	设置位置模式的最大转速是寄存器 D2 中的数据
	1	
	D2	

程序说明：X1 有上升沿时，会设置位置模式最大转速为 600rpm，X2 有上升沿时，会设置位置模式最大转速为 D2 寄存器存储的值，假设 D2 = 800,那么就会设置最大转速为 800rpm。

注意：该指令设置的位置模式最大转速必须是有符号 16 位整型数据，单位是 1rpm。

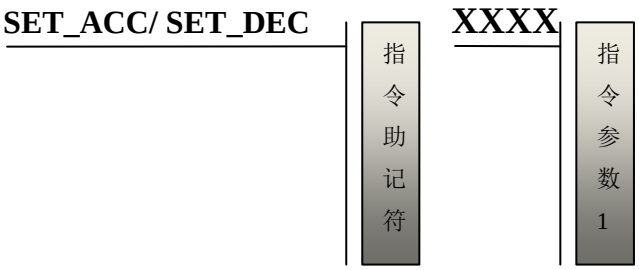
使用限制：SET_MAX VEL 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

SET_ACC/ SET_DEC：设置加速度/设置减速度

SET_ACC:设置驱动器运动控制的加速度，可以更改 T 曲线的加速度。

SET_DEC:设置驱动器运动控制的减速度，可以更改 T 曲线的减速度。

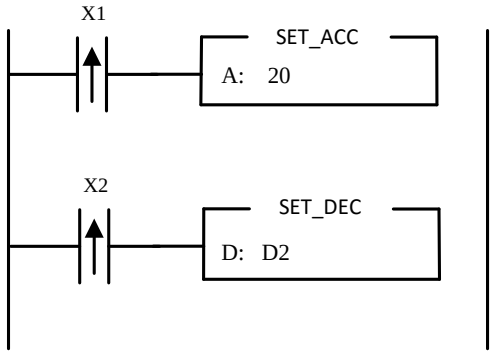
SET_ACC/ SET_DEC 指令的助记符语句结构为



指令参数 1 表示最大转速的取值常数或者寄存器地址

例如：

梯形图 助记符



LD_R	X1
SET_ACC	FUNC
	0
	20
LD_R	X2
SET_DEC	FUNC
	1
	D2

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	
SET_ACC	FUNC	检测 X1 的上升沿
	0	设置加速度是 20r/s2
	20	
LD_R	X2	
SET_DEC	FUNC	检测 X2 的上升沿
	1	设置减速度是寄存器 D2 中的内容
	D2	

程序说明：X1 有上升沿时，会设置加速度为 20r/s2，X2 有上升沿时，会设置加速度为 D2 寄存器内的数据，假设 D2=5，那么会设置减速度为 5r/s2。

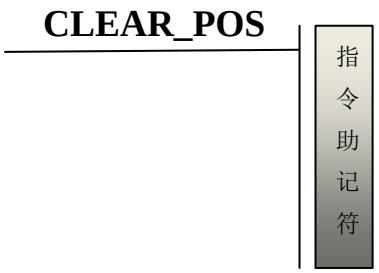
注意：该指令设置的加减速度必须是有符号 16 位整型数据，单位是 r/s2，驱动器内部轨迹规划时，T 曲线轨迹规划和速度模式下该加减速度。

使用限制：SET_ACC/ SET_DEC 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

CLEAR_POS：当前位置清零

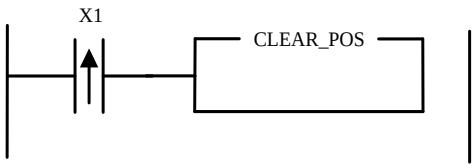
CLEAR_POS：该指令可以清零驱动器内部位置计数器计数，和编码器反馈的值。

CLEAR_POS 指令的助记符语句结构为



例如：

梯形图



助记符

LD_R	X1
CLEAR_POS	FUNC

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
CLEAR_POS	FUNC	当前位置计数清零

程序说明：X1 有上升沿时，会将当前驱动器内部的电机位置计数清零。使用该指令，可以配合电机回零操作，设置电机的绝对零点，在没有失步的情况下，该值是非常准确的。

注意：该指令只是对于当前位置的清零操作，在驱动器每次上电以后，当前位置与编码器的类型有关。

使用限制：CLEAR_POS 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

4.2 位置控制相关指令

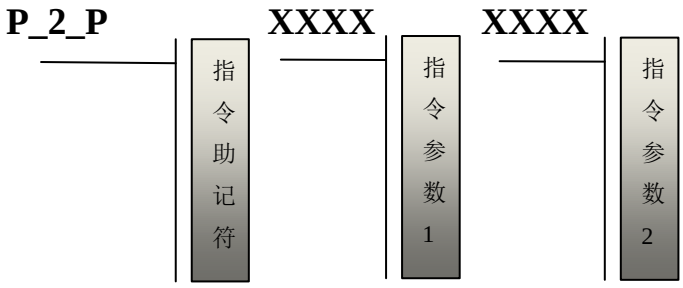
MOTEC 智能驱动器内置可编程控制器可以驱动电机进行精确的定位操作，使用相关的指令，就可以方便的驱动进行点到点控制，完全脱离传统的上位机控制器。

MOTEC 智能驱动器内置可编程控制器对于位置控制驱动需要使用两个指令。

P_2_P: 点到点设定指令

P_2_P:该指令可以设置一段点到点控制的各项参数，包括运动距离和运动方式。

P_2_P 指令的助记符语句结构为



指令参数 1 表示位置控制的运动模式：

0 :绝对运动模式

1:相对运动模式

指令参数 2 表示位置控制的运动距离或者寄存器地址。

具体见下表

指令参数 1	指令参数 2
0：绝对运动模式	32 位运动距离
1：相对运动模式	普通数据寄存器地址或者 32 为有符号常数

注意：在 MOTEC 智能驱动器内置可编程控制器中，位置控制的运动距离是一个 32 位有符号的整型数据，单位是脉冲个数。

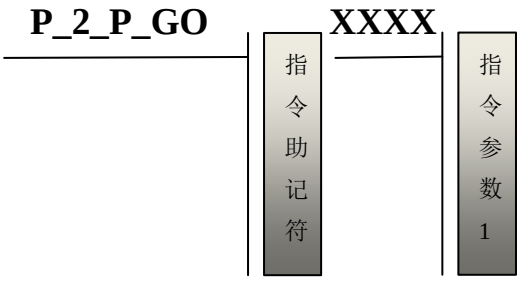
例如，假设驱动器的每转需要 10000 个脉冲，在相对运动模式下，要求电机正转五圈，运动距离应该设置成 50000，要求反转五圈，那么运动距离应该设置成-50000；

如果运动距离取自普通数据寄存器，那么指令参数 2 中的寄存器地址是 32 位有符号整型数据的低 16 位存放的地址。

P_2_P_GO：点到点启动指令

P_2_P_GO：该指令需要配合是在 P_2_P 指令下，启动一段点到点位置控制。

P_2_P_GO 指令的助记符语句结构为

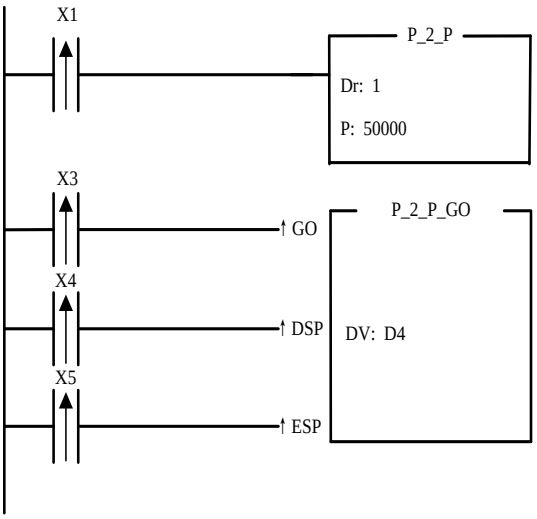


指令参数 1 表示减速速度：

注意: P_2_P_GO 并不是需要必须有 P_2_P 指令的参与，只要是驱动器内部寄存器设置的是相对位置运动，运动距离不为 0，或者是绝对运动模式，运动距离与当前位置不相等，都可以使用 P_2_P_GO 指令启动一段点到点的位置指令。

例如:

梯形图



助记符

LD_R	X1
P_2_P	FUNC
	1
	0
	50000
LD_R	X3
P_2_P_GO	FUNC
	1
	D4
FIN20	1
LD_R	X4
FIN30	1
LD_R	X5

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
P_2_P	FUNC	设置点到点位置控制的参数，运动距离取自
	1	32 位常数 50000 个脉冲，相对运动
	0	
	50000	
LD_R	X3	检测 X3 的上升沿
P_2_P_GO	FUNC	点到点位置控制运动启动，减速速度为寄存器
	1	D4 中的数值
	D4	
FIN20	1	
LD_R	X4	检测 X4 上升沿，电机以寄存器 D4 数据减速
FIN30	1	停止
LD_R	X5	检测 X5 上升沿，电机急停。

程序说明：该程序假设驱动器每转需要的脉冲数是 10000，如果 X1 有上升沿，则设置正转 5 圈，如果 X3 有上升沿，则电机开始运动。电机在运动过程中，当 X4 出现上升沿，电机以 D4 中的数据减速停止，当 X5 现上升沿，电机立刻停止

注意：P_2_P 不能启动电机运动，只是对下一段点到点位置控制做相应的参数设置，必须要通过 P_2_P_GO 才可以启动该段运动。P_2_P_GO 指令只会运动在此指令之前最后一个设置的点到点位置控制的参数，减速停止和急停可用可不用，根据客户需要。

使用限制：P_2_P/P_2_P_GO 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。如果只需要启动一个点到点运动时，那么只需要一个周期的 ON 状态，需要使用沿操作。

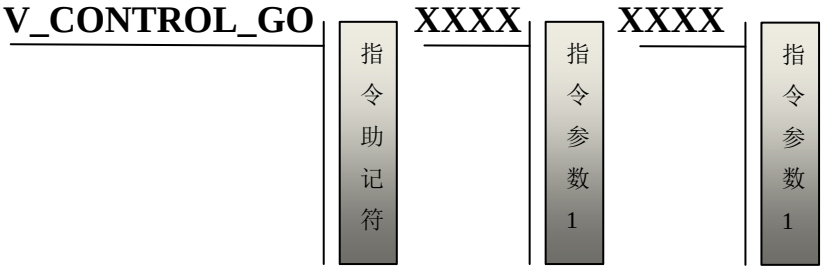
4.3 速度控制相关指令

MOTEC 智能驱动器内置可编程控制器可以使用可编程控制器指令直接控制电机以一定的转速转动，可以是正转或者反转。

V_CONTROL_GO：速度模式运行指令

V_CONTROL_GO：使用该指令，可以将驱动器设置成速度控制模式，使电机匀速运动，速度的给定值可以来自普通数据寄存器或者常数，电机处于使能状态时，该指令会直接启动电机转动。

V_CONTROL_GO 指令的助记符语句结构为



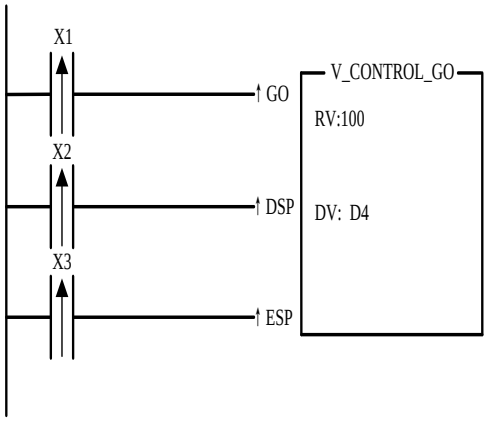
指令参数 1 表示速度的取值或者寄存器地址：

指令参数 2 表示减速速度取值或者寄存器地址；

注意：在 MOTEC 智能驱动器内置可编程控制器中，速度控制中设置的速度是一个 16 位有符号的整型数据，正数代表正转，负数代表反转，单位是 RPM。

例如

梯形图



助记符

LD_R	X1
V_CONTROL_GO	FUNC
	0
	100
	1
	D4
FIN20	1
LD_R	X2
FIN30	1
LD_R	X3

指令解释(助记符部分)

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
V_CONTROL_GO	FUNC	当前控制模式设置为速度模式，电机的速度设定值为 100rpm，减速速度取值寄存器 D4 中的数值
	0	
	100	
	1	
	D4	
FIN20	1	
LD_R	X2	检测 X2 的上升沿，电机以寄存器 D4 中的数据减速停止
FIN30	1	
LD_R	X3	检测 X3 上升沿，电机立刻停止

程序说明：检测到 X1 的上升沿时，会自动设置驱动器为速度模式，并且电机的设定转速取自常数 100，电机的设置转速为正转方向 100rpm，检测到 X2 的上升沿时，电机以寄存器 D4 中的数据减速停止，当检测到 X3 的上升沿，电机则立刻停止运动。

注意：如果电机当前是使能状态，当使用 V_CONROL_GO 时，会立即启动电机。急停与减速停止根据客户需要所使用。

使用限制：V_CONROL_GO 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。如果只需要启动一个速度控制运动时，那么只需要一个周期的 ON 状态，需要使用沿操作。

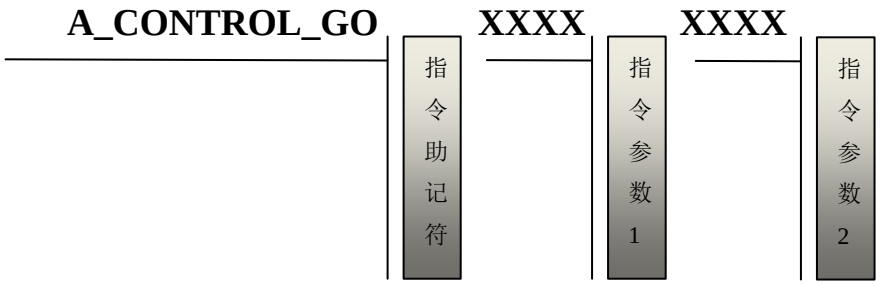
4.4 模拟量控制指令

MOTEC 智能驱动器内置可编程控制器在大部分型号驱动器内部集成了一或多路模拟量输入信号，可以使用该模拟量来调节电机的转速，具体的硬件信息请参照具体驱动器的使用手册。

A_CONTROL_GO：模拟量输入控制转速

A_CONTROL_GO：使用该指令，系统会直接读取 MOTEC 智能驱动器集成的模拟量输入口，并且将电机设置成速度模式，转速由模拟量值线性转化而成。

A_CONTROL_GO 指令的助记符语句结构为



指令参数 1 表示具体的操作内容：

0 :电机正向转动

1:电机反向转动

指令参数 2 表示减速速度取值

注意：该指令中的模拟量取自驱动器内部模拟量输入，参与运算的模拟量转速最大值，模拟量控制死区取自驱动器内部寄存器 P 中的定义。具体的 P 寄存器数据请查看相关的驱动器手册和参数表。电压信号发生突变时，为了不导致速度发生突变，MOTEC 智能驱动器设置了一个加速/减速过程，加/减速度值为 T 曲线轨迹规划设置的加/减速度值。

模拟量输入值具体转换为转速的过程如下

速度和电压的关系如下面的公式(1)所示V_{Max}

$$velocity = Dir \times V_{Max} \times V_{in} / 5 \quad (1)$$

其中，velocity 为实际速度，$V_{\max}$ 是设置的最大速度值，V_{in} 为电压输入值，Dir 为电机的运动方向，其取值为 1 或 0。当指令参数 1 为 0 时，$Dir = 1$，反之 $Dir = -1$。

电压和速度的关系如图 4.1 所示。

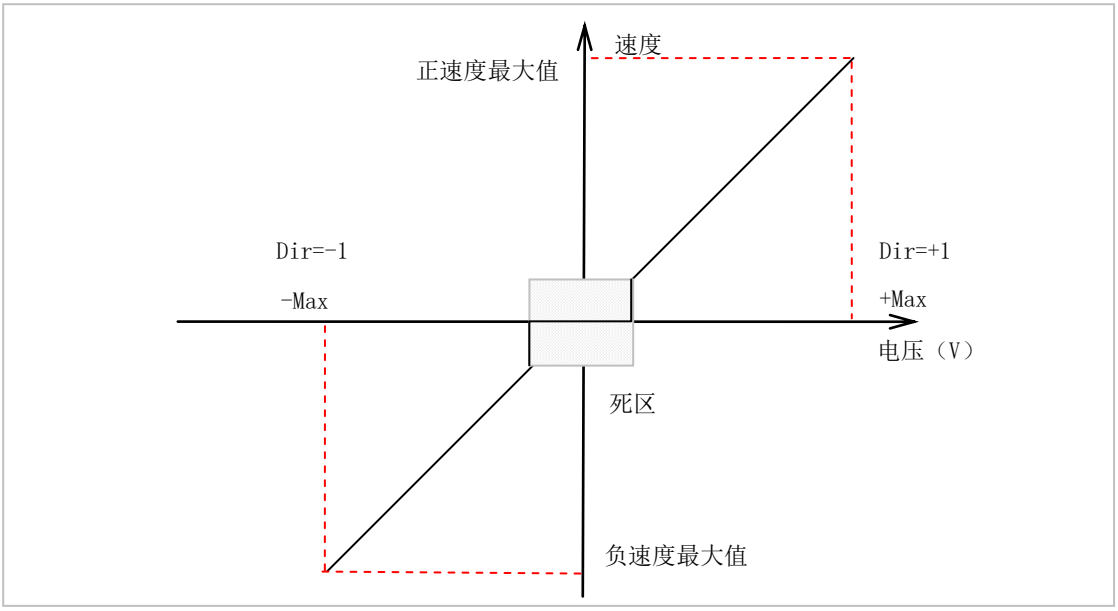
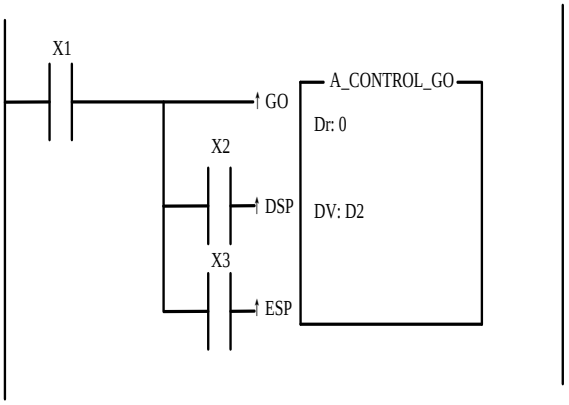


图 4.1 模拟电压信号和速度的关系

例如：

梯形图



助记符

```
LD              X1
MPS
A_CONTROL_GO    FUNC
                0
                X2
                X3
                1
                D2
```

指令解释(助记符部分)

指令内容		D2
LD	X1	指令说明
MPS		装载 X1 的状态
A_CONTROL_GO	FUNC	
	0	模拟模式匀速运行，正转，减速速度取值为寄
	X2	存器 D2 中
	X3	装载 X2 状态，电机以 D2 中的数据停止运行
	1	装载 X3 状态，电机立刻停止运行

程序说明：该程序可以控制电机以模拟输入的值来匀速运动，X1 导通的状态时，电机正转，假如模拟量最大转速设置为 1000rpm，当前模拟输入为+3V，那么电机的设定转速为

为相应比例的值，当 X2 导通后，电机以寄存器 D2 的数值减速停止，X3 导通，电机立刻停止。

注意：如果电机当前是使能状态，当使用 A_CONTROL_GO 指令时，如果当前模拟量输入大于控制死区，会立即启动电机。

使用限制： A_CONTROL_GO 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。该指令执行条件为 ON 时，才会更新模拟量设置转速，如果需要每个扫描周期根据模拟量输入更新转速，那么需要在每个周期 A_CONTROL_GO 指令的执行条件为 ON，而不能使用沿操作形式。

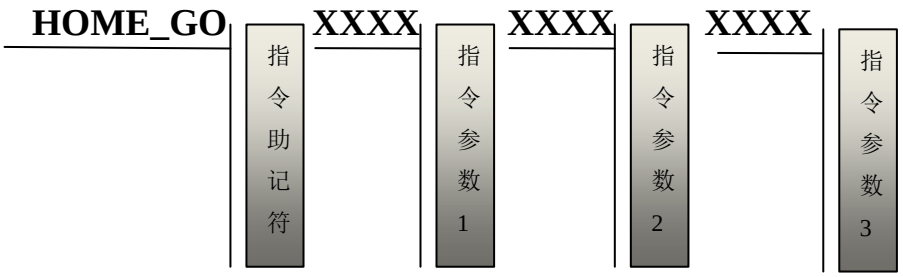
4.5 回零操作相关

MOTEC 智能驱动器内置可编程控制器可以在程序中直接控制电机回零操作。

HOME_GO：回零操作指令

HOME_GO：该指令可以操作驱动器回零，回零的方式和转速都可以设置。

HOME_GO 指令的助记符语句结构为



指令参数 1 表示回零转速的取值来源：

0——回零转速取自常数

1——回零转速取自寄存器

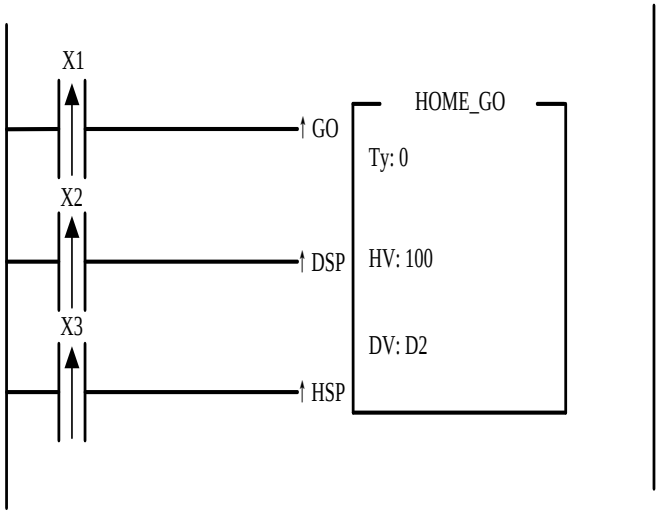
指令参数 2 表示回零转速的常数值或者取值的寄存器地址。

指令参数 3 表示减速速度取值

注意：MOTEC 智能驱动器可以通过上位机软件设置正负限位开关和零位开关，回零寻找 Z 脉冲个数等，在需要使用到某一个方向的零位开关时，需要将对应方向输入口设置为限位开关。并且，无论正负只是相对于电机的转动方向而定。下面说明电机寻找零位开关的相关操作。

例如

梯形图



助记符

LD_R	X1
HOME_GO	FUNC
	0
	0
	100
	1
	D2
FIN 20	1
LD_R	X2
FIN 30	1
LD_R	X3

指令解释（助记符部分）

指令内容		指令说明
LD_R	X1	检测 X1 的上升沿
HOME_GO	FUNC	回零开始，回零转速为 100rpm，减速速度取
	0	自寄存器 D2
	0	
	100	
	1	
	D2	
FIN 20	1	
LD_R	X2	装载 X2 的上升沿，电机以寄存器 D2 中的数
FIN 30	1	据减速停止
LD_R	X3	装载 X3 上升沿，立刻停止回零

程序说明：使用 X1 继电器的上升沿来启动回零，回零转速是 100rpm，当 X2 输入继电器（INPUT3）出现上升沿，回零慢慢减速停止运行，在回零过程中，当检测 X3 上升沿出现，回零运动立刻停止。

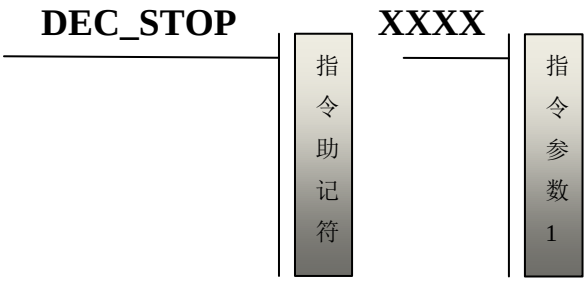
注意：在启动回零操作时，系统会自动将当前的控制模式设置为位置模式，如果用户需要回归电机内部计数的零位置，可以使用位置控制操作，运动方式设置为绝对位置模式，目标位置为 0，减速停止，根据客户需要所用。

使用限制：HOME_GO 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。该指令执行条件为 ON 时，才会继续回零。

DEC_STOP：减速停止

DEC_STOP：该指令用来使电机减速停止。

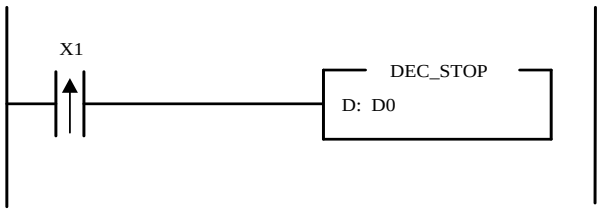
DEC_STOP 指令的助记符语句结构为



指令参数 1 表示减速速度可取自 D 寄存器或正整数

例如

梯形图



使用限制：DEC_STOP 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

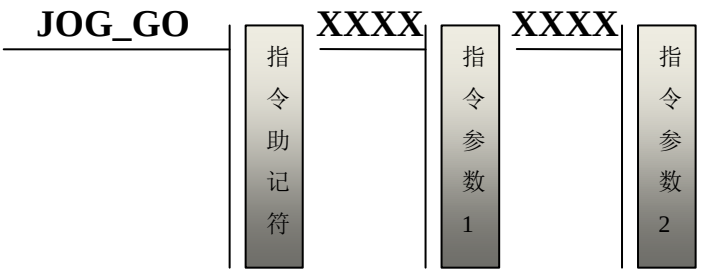
4.6 点动操作指令

MOTEC 智能驱动器内置可编程控制器可以在程序中直接控制电机进行点动操作，点动操作的方向可以是正向或者负向，并且点动转速可以设置。

JOG_GO：点动操作指令

JOG_GO：该指令可以该扫描周期执行条件有效时控制电机转动，执行条件无效时电机停止。

JOG_GO 指令的助记符语句结构为



指令参数 1 表示点动的运动方向：

0——正向点动

1——负向点动

指令参数 2 表示点动速度的取值来源：

0——点动速度取自常数

1——点动速度取自寄存器

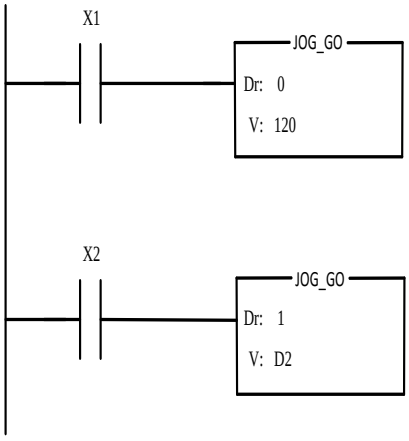
具体如下表

指令参数 1	指令参数 2
0：正向点动	0：点动速度取自常数
1：负向点动	1：点动速度取自寄存器

——点动转速没有方向，只可以为正数

例如

梯形图



助记符

```
LD        X1
JOG_GO    FUNC
          0
          0
          120
LD        X2
JOG_GO    FUNC
          1
          1
          D2
```

指令解释（助记符部分）

指令内容		指令说明
LD	X1	装载 X1 的状态
JOG_GO	FUNC	正向点动，点动速度为常数 120rpm
	0	
	0	
	120	
LD	X2	装载 X2 的状态
JOG_GO	FUNC	负向点动，点动速度取自寄存器 D2 的内
	1	容
	1	
	D 2	

程序说明：该程序是使用 X1 输入继电器的 ON 状态控制电机正向点动，点动速度是常数 120rpm，使用 X2 输入继电器的 ON 状态控制电机的负向点动，点动速度取自寄存器 D2 中的内容，如果 D2 中的数值为 60，那么负向点动的速度就是 60rpm。

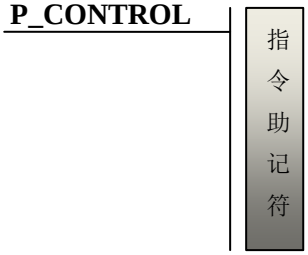
注意：点动转速没有正负之分，当使用 JOG_GO 指令时，系统会自动的将控制模式设置为位置控制模式；如果使用点动指令往同一方向点动，触发的继电器不同的时候，程序中需要加互锁功能。否则不同的启动条件有冲突，会导致不能做点动操作。

使用限制：JOG_GO 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。该指令执行条件为 OFF 时，电机会马上停止。

4.7 脉冲方向模式指令

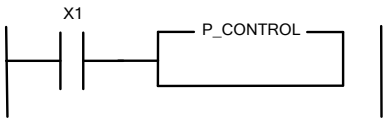
P_CONTROL：该指令将驱动器的操作模式改为脉冲方向模式，电机运动目标位置来自脉冲。

P_CONTROL 指令的助记符语句结构为



例如

梯形图



助记符

LD	X1
P_CONTROL	FUNC

指令解释（助记符部分）

指令内容		指令说明
LD	X1	检测 X1 上升沿
P_CONTROL	FUNC	脉冲模式

程序说明：该程序检测 X1 为 ON 信号，触发将操作模式改为脉冲模式，电机运动来自脉冲方向。

使用限制：P_CONTROL 指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。

第 5 章 编程时注意的事项

1、定时器操作指令、计数器操作指令、高级指令、运动控制指令不能放在母线的开始部分，只有在当前执行条件为 ON 的时候，该指令才会执行。也就是需要输入继电器或普通辅助继电器来配合执行。

2、T 继电器和 C 继电器在上电一刻会被初始化为 OFF 状态，只有在操作过一次该定时器或者计数器之后，T 继电器和 C 继电器的状态才取决于 TLD 寄存器，CLD 寄存器是否为 0 有关。所以定时器操作指令和计数器操作指令在定时器状态继电器或计数器状态继电器参与的逻辑运算中，需要使用一个辅助继电器参与才能保证在第一次上电以后定时器或计数器没有设置之前能够为 ON。

3、进行某些高级指令或某些运动控制指令时，必须使用沿操作，同样可以使用电平操作，但是需要在指令执行结束之后对触发电平做复位操作。为了保证数据的准确，时间速度快，普通辅助继电器最好使用沿操作。根据程序的需要，如果需要再次执行高级指令或运动控制指令，利用复位指令将普通辅助继电器的状态清除。例如：3.4.1 中 16 位数据算术运算指令。

4、32 位寄存器是由两个连续的 16 位寄存器连接起来，指令参数中为寄存器地址时，指定用于寄存器低 16 位地址，在 32 位操作时候，建议客户使用偶数寄存器地址，并且避开已经使用的寄存器的下一个寄存器。

5、P 寄存器会有读写限制，具体详见各驱动器的参数表说明。

联系方式：

MOTEC（中国）营业体系
北京诺信泰伺服科技有限公司
地址：北京市通州区环科中路17号11B（联东U谷西区）
电话：010-56298855-666
传真：010-65546721
邮编：100027
网址：http://www.motec365.com
http://www.nortiontech.com
eMail：motecSupport@sina.com