M218 编程指令

3

综述　　本章给出了本书中 M218 的编程指令介绍

本章内容　　本章包含一下章节内容：

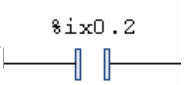
布尔逻辑指令（处理位）

布尔逻辑指令用于处理位格式数据（1/O 位、内部位等）

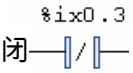
- 输入/输出　例如：N/O（常开/常闭）触点。
- 执行元件　例如：直接线圈（%Q、%M）等
- 上升、下降沿：用于检测 PLC I/O 位和内部位的上升、下降沿

装入指令

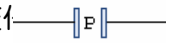
N/O(常开)触点：当控制这个触点的状态为 1 时，触点



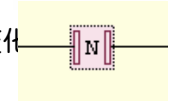
N/C(常闭)触点：当控制这个触点的状态为 0 时，触点闭



上升沿触点：检测控制位从 0 到 1 的变



下降沿触点：检测控制位从 1 到 0 的变



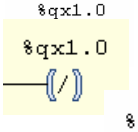
程序例：

分析：当输入点%IX0.2 的状态是 1，同时输入点%IX0.3 状态是 0 时，则输出点%QX1.0 的输出 1.

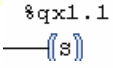
输出指令

直接输出线圈：相关的位实体取等式的直接结果

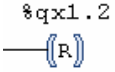
反向输出线圈：相关的位实体取等式的直接反值



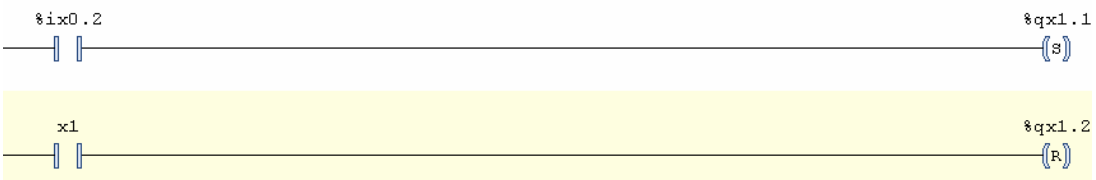
置位（SR）线圈：等式结果为 1 时，强制输出位置为 1



复位（RS）线圈：等式结果为 1 时，强制输出位置为

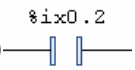


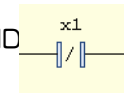
程序例：



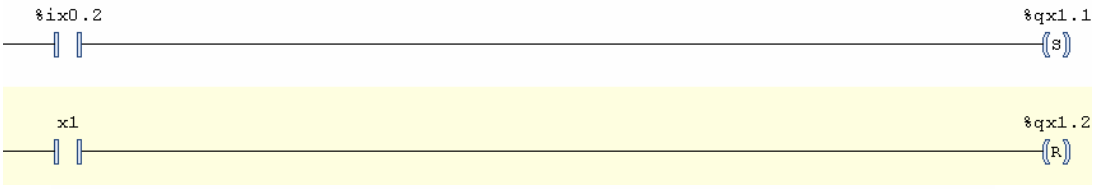
分析：当%IX0.2 状态是 1 时，将%QX1.1 置位为 1；当 X1 状态是 1 时，将%QX1.1 置位 0.

逻辑与（AND）指令

执行操作数与前一条指令的逻辑与（AND 

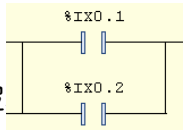
执行操作数与前一条指令的逻辑与反（AND 

程序例：

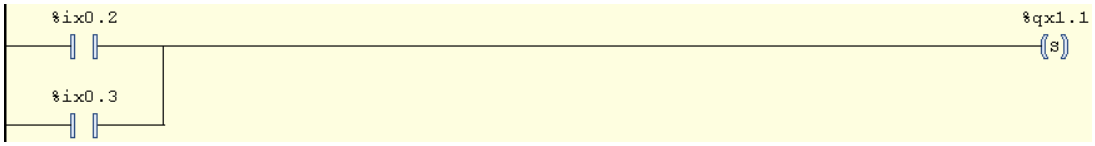


分析：当 IX0.7 状态是 1，同时 MX2.0 状态是 0 时，QX0.2 状态是 1.

逻辑或（OR）指令

OR 执行操作数与前一条指令的逻 

程序例：



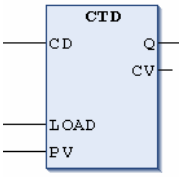
分析：当%IX0.2 状态是 1，或者%IX0.3 是状态 1，则%QX1.1 置为 1

标准库指令

多选一指令 MUX	
取极限指令 LIMIT 取最大值 MAX	
取最小值指令 MIN	
赋值指令 MOVE	
布尔类型转换指令	
字节类型转换指令	
日期转换指令	
实数/长实数类型转换	
字符串类型转换命令	

减计数器指令 CTD

指令块如下图



输入：

CD：布尔型（BOOL）；该输入端的上升沿触发 CV 的递减计数

LOAD：布尔型（BOOL）；当其为上升沿触发时，CV 被置为上限值 PV

PV：字型（WORD）；上限值，也就是 CV 开始递减时的初始值

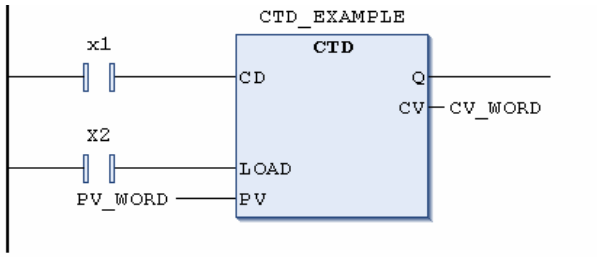
输出：

Q：布尔型（BOOL）；一旦 CV 达到 0 时，其值为 TRUE

CV：字型（WORD）；不断减 1 的值，从 PV 开始直至其达到 0

当 LOAD 为 TRUE 时，计数变量 CV 被初始化为上限值 PV。当 CD 端有一个从 FALSE 变为 TRUE 的上升沿时，若 CV 大于 0 时，它将减 1（也就是说，它不会输出小于 0 的值）。当 CV 等于 0 时，Q 返回 TRUE。

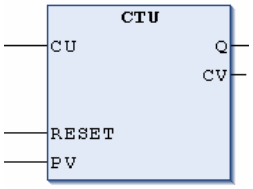
程序例：



分析：程序执行时，将 PV_WORD 设为 3，当 X2 由 FALSE 变为 TRUE 上升沿触发时，CV_WORD 也变为 3。此时输入端 X1 执行 FALSE 变为 TRUE 上升沿触发，则 CV_WORD 自动减计数 1；当 X1 第 3 次由 FALSE 变 TRUE 上升沿触发时，CV_WORD 递减到 0 时，此时 Q 输出为 1.

加计数指令 CTU

指令块如下图



输入：

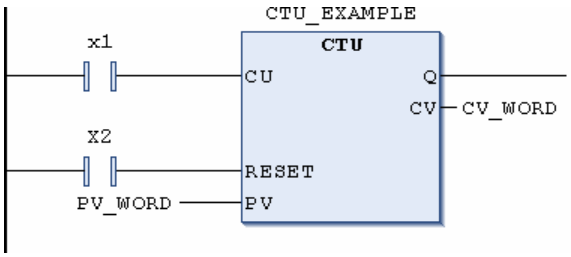
CU：布尔型（BOOL）；该输入端的上升沿触发 CV 的递增计数

RESET：布尔型（BOOL）；当其为 TRUE 时，CV 被复位为 0

PV：字型（WORD）；CV 计数的上限

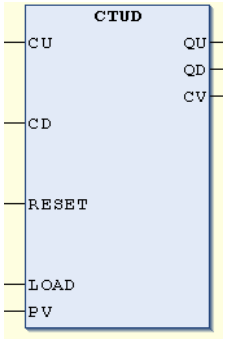
输出：

Q：布尔型（BOOL）；一旦 CV 达到其上限 PV 时，其值为 TRUE
CV：字型（WORD）；不断加 1 的值，直至其达到 PV
当 RESET 为 TRUE 时，计数变量 CV 被初始化为 0。当 CU 端有一个从 FALSE 变为 TRUE 的上升沿时，CV 将加 1。当 CV 大于或等于上限 PV 时，Q 返回 TRUE。
程序例：

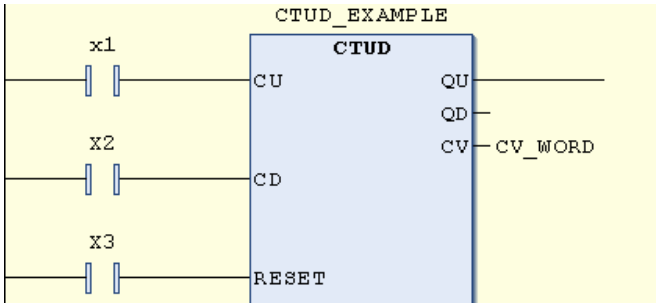


分析：程序执行时，将 PV_WORD 设为 3，此时 CV_WORD 是 0。此时输入端 X1 执行 FALSE 变为 TRUE 上升沿触发，则 CV_WORD 自动加计数 1；当 X1 第 3 次由 FALSE 变 TRUE 上升沿触发时，CV_WORD 递增到 3 时，此时 Q 输出为 1。此时如将 RESET 端 X2 执行 FALSE 变为 TRUE 上升沿触发，则指令重新复位数变量 CV 被初始化为 0。

加减双向计数指令 CTUD
指令块如下图



输入：
CU：布尔型（BOOL）；当 CU 端有上升沿时，触发 CV 的递增计数
CD：布尔型（BOOL）；当 CD 端有上升沿时，触发 CV 的递减计数
RESET：布尔型（BOOL）；当其为 TRUE 时，CV 被复位为 0
LOAD：布尔型（BOOL）；当其为 TRUE 时，CV 被置为 PV
PV：字型（WORD）；CV 递增时的上限值，或 CV 开始递减时的初始值
输出：
QU：布尔型（BOOL）；一旦 CV 达到 PV 时，其值为 TRUE
QD：布尔型（BOOL）；一旦 CV 达到 0 时，其值为 TRUE
CV：字型（WORD）；不断减 1 的值，从 PV 开始直至其达到 0
程序例：



分析：当 RESET 为 TRUE 时，计数变量 CV 被初始化为 0。当 LOAD 为 TRUE 时，计数变量 CV 被初始化为上限值 PV。当 CU 端有一个从 FALSE 变为 TRUE 的上升沿时，CV 将加 1。当 CD 端有一个从 FALSE 变为 TRUE 的上升沿时，若 CV 不会降到 0 以下时，它将减 1。当 CV 大于或等于上限 PV 时，QU 返回 TRUE。当 CV 等于 0 时，QD 返回 TRUE。

延时断开指令 TOF

定时器功能块，完成关延时的功能。当定时器的输入端由 TRUE 变为 FALSE 时（下降沿），等过了一段时间后，定时器的输出端才变为 FALSE。

指令块如下图

输入：

IN：布尔型（BOOL）；该输入端的下降沿触发 ET 端的计时

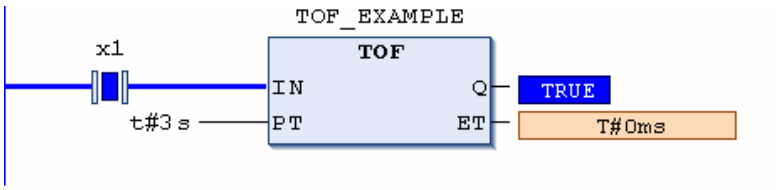
PT：时间型（TIME）；ET 计时时间的上限值（延时时间）

输出：

Q：布尔型（BOOL）；一旦 ET 端计时达到上限值 PT 时，输出一个下降沿（延时时间过去了）

ET：时间型（TIME）；时间的当前状态

程序例：



分析：当 IN 为 TRUE 时，Q 为 TRUE，ET 为 0。

一旦 IN 变为 FALSE，定时器的输出端 ET 以精确到毫秒级别开始计时，直到它等于 PT，随后它会维持不变。当 IN 变为 FALSE 且 ET 等于 PT 时，Q 为 FALSE。否则它为 TRUE。在本例中，PT 设为 3s，当 X1 由 TRUE 变为 FALSE 下降沿触发时，定时器输出端 ET

开始计时，定时到达 3s 后输出 Q 由 TRUE 变为 FALSE。

延时导通指令 TON

定时器功能块，完成开延时的功能。当定时器的输入端变为 TRUE 时，等过了一段时间后，定时器的输出端才变为 TRUE。

指令块如下图

输入：

IN：布尔型（BOOL）；该输入端的上升沿触发 ET 端的计时

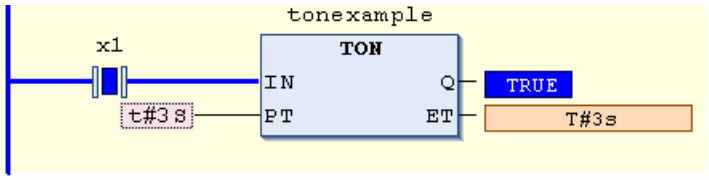
PT：时间型（TIME）；ET 计时时间的上限值（延时时间）

输出：

Q：布尔型（BOOL）；一旦 ET 端计时达到上限值 PT 时，输出一个上升沿（延时时间过去了）

ET：时间型（TIME）；时间的当前状态

程序例



分析：

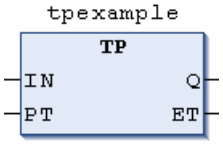
当 IN 为 FALSE 时,Q 为 FALSE，ET 为 0。一旦 IN 变为 TRUE，定时器的输出端 ET 以精确到毫秒级别开始计时，直到它等于 PT，随后它会维持不变。当 IN 变为 TRUE 且 ET 等于 PT 时，Q 为 TRUE。否则它为 FALSE。

在本例中，PT 设为 3s，当 X1 由 FALSE 变为 TRUE 上升沿触发时，定时器输出端 ET 开始计时，定时到达 3s 后输出 Q 由 FALSE 变为 TRUE。

触发定时器指令 TP

触发定时器功能块。定时器的输出值不断增加，直至其达到限值。在计时期间，“脉冲”变量为 TRUE，其他时候为 FALSE。

指令块如下图



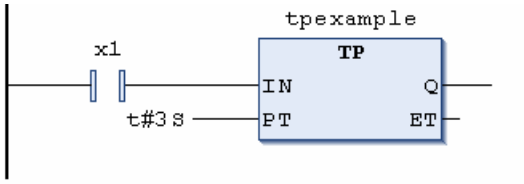
输入：

IN：布尔型（BOOL）；该输入端的上升沿触发 ET 端的计时

PT：时间型（TIME）；计时时间的上限值

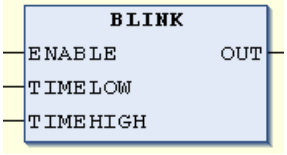
输出：

Q：布尔型（BOOL）；当 ET 端在计时的时候，其值为 TRUE
ET：时间型（TIME）；时间的当前状态
程序例：

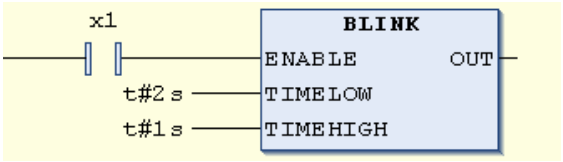


分析：
当 IN 为 FALSE 时,Q 为 FALSE，ET 为 0。一旦 IN 变为 TRUE，定时器的输出端 ET 以毫秒精度开始计时，直到它等于 PT，随后它会维持不变。当 IN 变为 TRUE 且 ET 小于或等于 PT 时，Q 为 TRUE。否则它为 FALSE。在由 PT 值指定的时间到达时，Q 返回了一个信号
在本例中，PT 设为 3 时，X1 为 FALSE，Q1 为 FALSE，当 X1 为 TRUE，Q 输出变为 TRUE，同时 ET 开始计数，当 ET=3S 时，Q 输出变回为 FALSE.

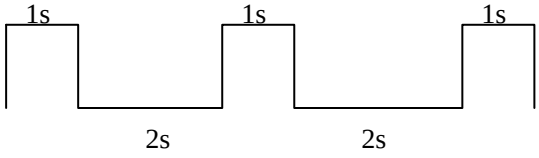
脉冲指令 BLINK
功能块 BLINK 产生脉冲信号。输入由 BOOL 类型 ENABLE，以及 TIME 类型 TIMELOW 和 TIMEHIGH 组成。输出 OUT 是 BOOL 类型。
指令块如下图



如果 ENABLE 为 TRUE，在时间周期 TIMEHIGH，BLINK 设置输出为 TRUE；然后在时间周期 TIMELOW，设置输出为 FALSE。
程序例

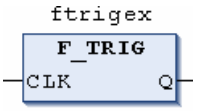


分析：在本例中，当 ENABLE X1 是 TRUE 时，BLINK 开始工作，输出低电平 2s 高电平 1s 的脉冲，如下图。



下降沿触发指令 F_TRIG

该功能块检测一个下降沿。指令块如下图



输入：

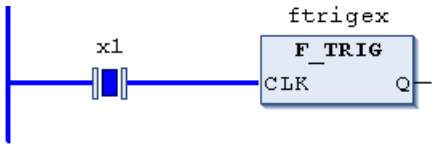
CLK：布尔型（BOOL）；被检测其下降沿的布尔型输入信号

输出：

Q：布尔型（BOOL）；当 CLK 上检测到一个下降沿时，其值为 TRUE

只要输入变量 CLK 为 TRUE，输出 Q 都保持为 FALSE。一旦 CLK 为 FALSE，Q 会先返回 TRUE，然后被置为 FALSE。这意味着每次调用这个功能块时，Q 会返回 FALSE 直到 CLK 在上升沿后有一个下降沿

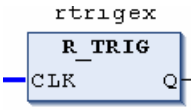
程序例：



分析：在本例中，当 X1 输入一个由 TRUE 变为 FALSE 的下降沿，则 F_TRIG 的输出 Q 也输出一个由 FALSE 变为 TRUE 的上升沿，然后再变为 FALSE。

上升沿触发指令 R_TRIG

该功能块检测一个上升沿。指令块如下图



输入：

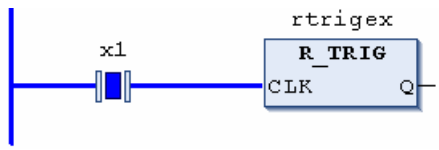
CLK：布尔型（BOOL）；被检测上升沿的布尔型输入信号

输出：

Q：布尔型（BOOL）；当 CLK 上检测到一个上升沿时，其值为 TRUE

只要输入变量 CLK 为 FALSE，输出 Q 保持为 FALSE。一旦 CLK 为 TRUE，Q 会先返回 TRUE，然后被置为 FALSE。这意味着每次调用这个功能块时，Q 会返回 FALSE 直到 CLK 在下降沿后有一个上升沿

程序例：



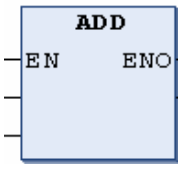
分析：在本例中，当 X1 输入一个由 FALSE 变为 TRUE 的上升沿，则 R_TRIG 的输出 Q

也输出一个由 FALSE 变为 TRUE 的上升沿，然后再变为 FALSE。

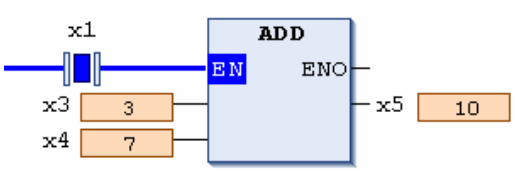
加运算指令 ADD

变量相加。允许的变量类型： BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。

指令块如下图



程序例：

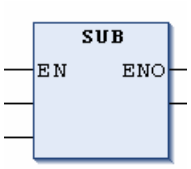


分析：在本例中，当 X1 为 TRUE 时，ADD 执行把操作数 X3 和 X4 相加的运算，并把结果输出到 X5 中，如 X3=3；X4=7；则 X5=10。

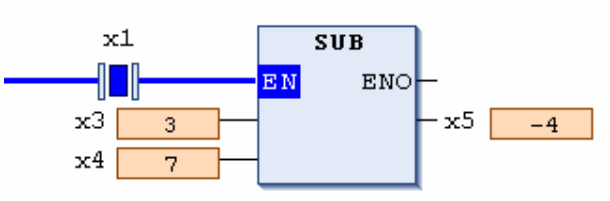
减运算指令 SUB

从某个变量中减去一个变量。允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。

指令如下图



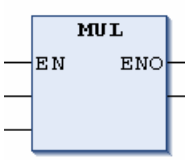
程序例：



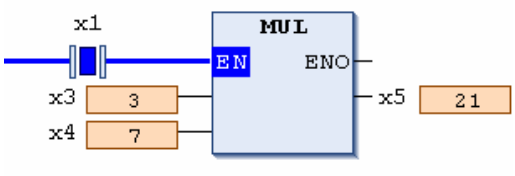
分析:在本例中，当 X1 为 TRUE 时，SUB 执行把操作数 X3 减去 X4 的运算，并把结果输出到 X5 中，如 X3=3；X4=7；则 X5=−4。

乘运算指令 MUL

变量相乘。允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。
指令如下图

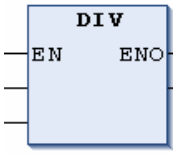


程序例：

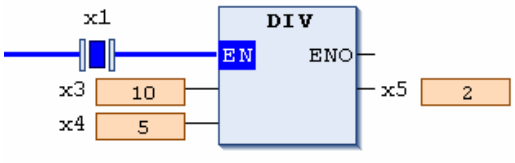


分析：在本例中，当 X1 为 TRUE 时，MUL 执行把操作数 X3 乘以 X4 的运算，并把结果输出到 X5 中，如 X3=3；X4=7；则 X5=21。

除运算指令 DIV
用一个变量除另一个变量。允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL 和 LREAL。
指令如下图



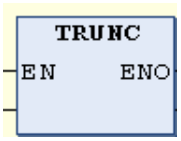
程序例：



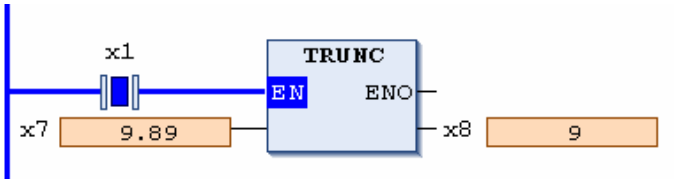
分析：在本例中，当 X1 为 TRUE 时，DIV 执行把操作数 X3 除 X4 的运算，并把结果输出到 X5 中，如 X3=10；X4=5；则 X5=2。

截尾取整指令 TRUNC

把实数类型（REAL）转换成DINT类型。取被转换值的整数部分。
指令如下图

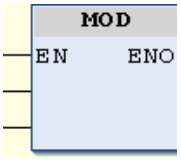


程序例：

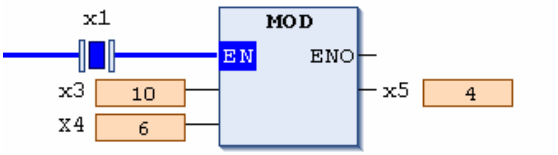


分析：在本例中，当 X1 为 TRUE 时，TRUNC 执行把操作数 X7 截尾取整的运算，即当 X7=9.89 时，取其整数部分，并把结果输出到 X8，即 X8=9。

取余指令 MOD
一个变量与另一个变量相除取余。允许的变量类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT。结果为除法运算的余数，是一个整数。指令如下图

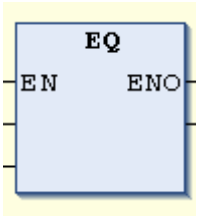


程序例：



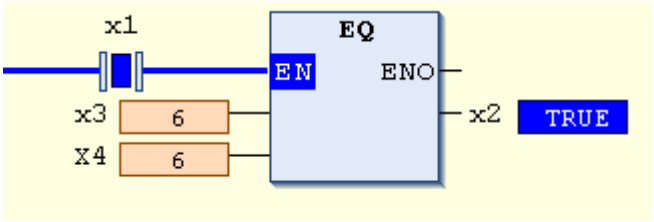
分析：在本例中，当 X1 为 TRUE 时，MOD 指令执行把 X3 除以 X4，并把余数输出到 X5 中；即当 X3=10，X4=6 时，余数 X5=4。

比较等于指令 EQ
指令如下图：



当两个操作数相等时，返回值为 TRUE。操作数可以为 BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、LREAL、TIME、DATE、TIME_OF_DAY、DATE_AND_TIME 和 STRING 类型。

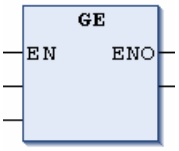
程序例：



分析：在本例中，当 X1 为 TRUE 时，EQ 指令执行，比较 X3 和 X4 的值，如 X3=X4，则输出比较结果 X2 为 TRUE；否则，X2 为 FALSE。所以，当 X3=X4=6 时，比较结果 X2=TRUE。

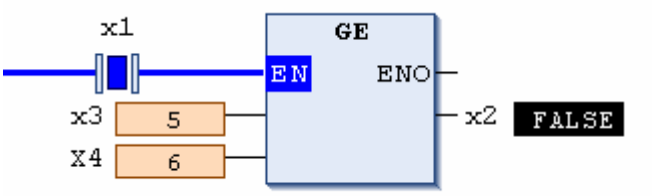
比较 大于等于指令 GE

大于或等于。指令如下图：



当第一个操作数大于或者等于第二个操作数时，返回值为 TRUE。操作数可以为 BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、LREAL、TIME、DATE、TIME_OF_DAY、DATE_AND_TIME 和 STRING 类型。

程序例：

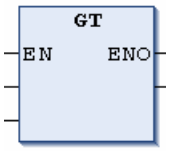


分析：在本例中，当 X1 为 TRUE 时，GE 指令执行，比较 X3 和 X4 的值，如 X3 大于等于 X4，则输出比较结果 X2 为 TRUE；否则，X2 为 FALSE。所以，当 X3=5；X4=6 时，比

较结果 X2=FALSE。

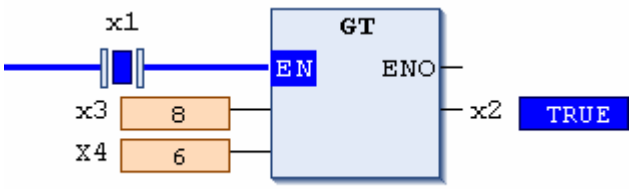
比较 大于指令 GT

大于。指令如下图：



当第一个操作数比第二个大时，返回值为 TRUE。操作数可以为 BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、LREAL、TIME、DATE、TIME_OF_DAY、DATE_AND_TIME 和 STRING 类型。

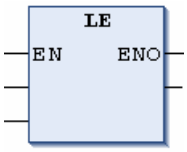
程序例：



分析：在本例中，当 X1 为 TRUE 时，GT 指令执行，比较 X3 和 X4 的值，如 X3 大于 X4，则输出比较结果 X2 为 TRUE；否则，X2 为 FALSE。所以，当 X3=8；X4=6 时，比较结果 X2=TRUE。

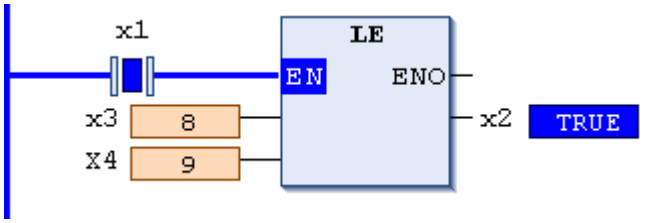
比较 小于等于指令 LE

小于等于。指令如下图：



第一个操作数小于或者等于第二个操作数时，返回值为 TRUE。操作数可以为 BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、LREAL、TIME、DATE、TIME_OF_DAY、DATE_AND_TIME 和 STRING 类型。

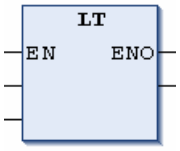
程序例：



分析：在本例中，当 X1 为 TRUE 时，LE 指令执行，比较 X3 和 X4 的值，如 X3 小于或者等于 X4，则输出比较结果 X2 为 TRUE；否则，X2 为 FALSE。所以，当 X3=8；X4=9 时，比较结果 X2=TRUE。

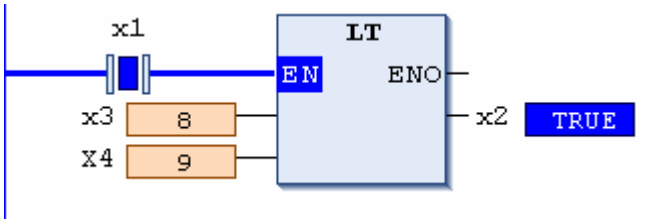
比较 小于指令 LT

小于。指令如下图：



当第一个操作数比第二个小时，返回值为 TRUE。操作数可以为 BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、LREAL、TIME、DATE、TIME_OF_DAY、DATE_AND_TIME 和 STRING 类型。

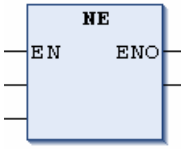
程序例：



分析：在本例中，当 X1 为 TRUE 时，LE 指令执行，比较 X3 和 X4 的值，如 X3 小于 X4，则输出比较结果 X2 为 TRUE；否则，X2 为 FALSE。所以，当 X3=8；X4=9 时，比较结果 X2=TRUE。

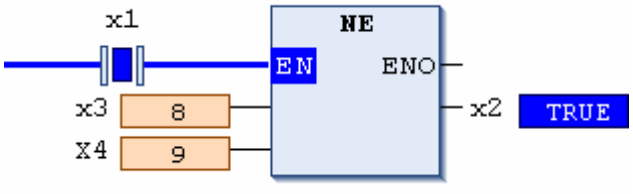
比较 不等于指令 NE

不等于。指令如下图：



当两个操作数不相等时，返回值为 TRUE。操作数可以为 BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、LREAL、TIME、DATE、TIME_OF_DAY、DATE_AND_TIME 和 STRING 类型。

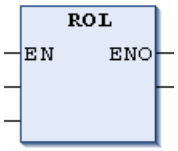
程序例：



分析：在本例中，当 X1 为 TRUE 时，LE 指令执行，比较 X3 和 X4 的值，如 X3 不等于 X4，则输出比较结果 X2 为 TRUE；否则，X2 为 FALSE。所以，当 X3=8；X4=9 时，比较结果 X2=TRUE。

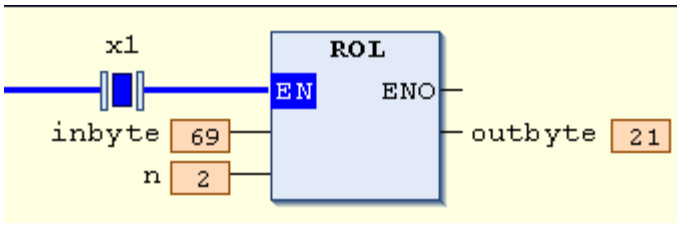
循环左移指令 ROL

将操作数按位循环左移。指令如下图：



ROL（in，n）。允许的数据类型：BYTE、WORD、DWORD。in 会左移二进制位 n 次，同时左边移出的位重新补充到右边。

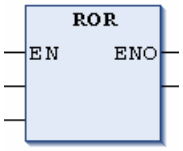
程序例：



分析：在本例中，inbyte 和 outbyte 分别设为 byte 数据类型，inbyte=10#69，n=2。当 X1 为 TRUE 时，ROL 执行循环左移位，此时 inbyte=10#69=2#01000101，循环左移 2 位后，outbyte=2#00010101=10#21。

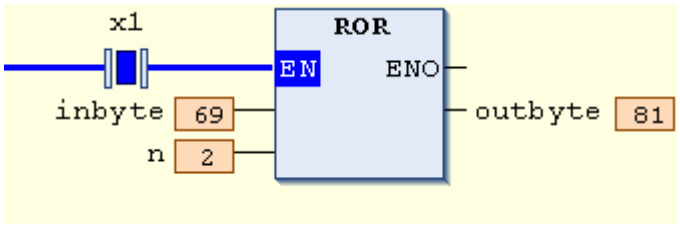
循环右移指令 ROR

将操作数按位循环右移。指令如下图：



ROR（in，n）。允许的数据类型：BYTE、WORD、DWORD。 in 将会右移二进制位 n 次，同时右边移出的位将会重新补充到左边。

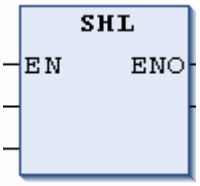
程序例：



分析：在本例中，inbyte 和 outbyte 分别设为 byte 数据类型，inbyte=10#69，n=2。当 X1 为 TRUE 时，ROR 执行循环右移位，此时 inbyte=10#69=2#01000101，循环右移 2 位后，outbyte=2#01010001=10#81。

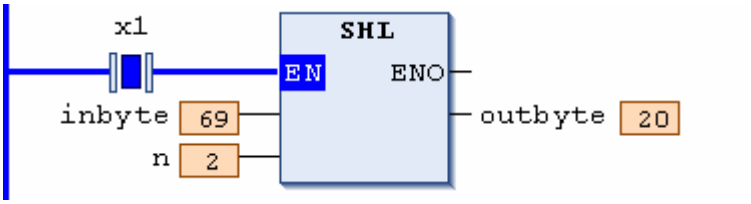
左移指令 SHL

将操作数按位左移。指令如下图：



SHL（in，n）。in：需要左移的操作数。n：操作数左移的位数。如果 n 超出了数据本身的位数，BYTE、WORD 和 DWORD 类型的操作数将会补 0，而有符号类型的操作数（例如 INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

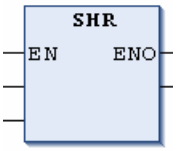
程序例：



分析：在本例中，inbyte 和 outbyte 分别设为 byte 数据类型，inbyte=10#69，n=2。当 X1 为 TRUE 时，ROL 执行左移位，此时 inbyte=10#69=2#01000101，左移 2 位后，outbyte=2#00010100=10#20。

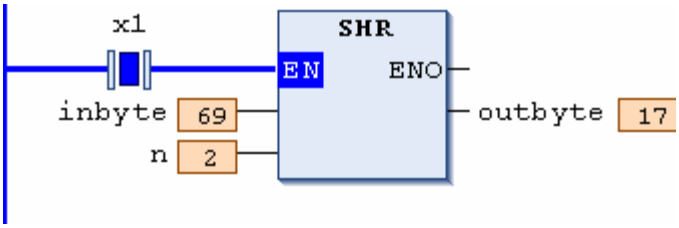
右移指令 SHR

将操作数按位右移。指令如下图：



SHR（in，n）。in：需要右移的操作数。n：操作数右移的位数。如果 n 超出了数据本身的位数，BYTE、WORD 和 DWORD 类型的操作数将会补 0，而有符号类型的操作数（例如 INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

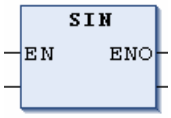
程序例：



分析：在本例中，inbyte 和 outbyte 分别设为 byte 数据类型，inbyte=10#69，n=2。当 X1 为 TRUE 时，ROR 执行右移位，此时 inbyte=10#69=2#01000101，循环右移 2 位后，outbyte=2#00010001=10#17。

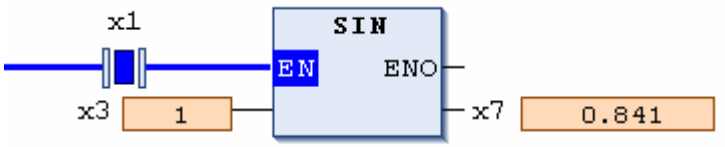
正弦函数 SIN

返回一个数的正弦值，数据以弧度计算。。指令如下图：



输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。

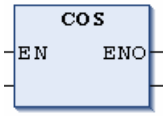
程序例：



分析：在本例中，X3=1 弧度=180/pi 度；当 X1 为 TRUE 时，SIN 执行正弦运算，将 X3 的正弦值输出到 X7 中，所以 X7=0.841。

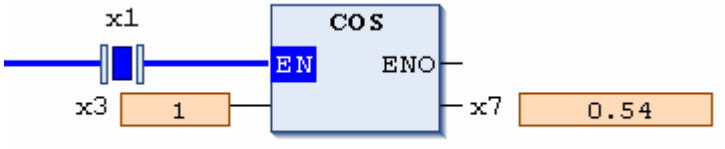
余弦函数 COS

返回一个数的余弦值，数据以弧度计算。指令如下图：



输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。

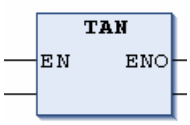
程序例：



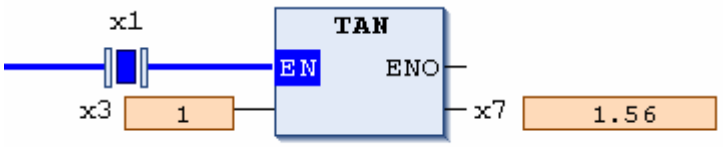
分析：在本例中，X3=1 弧度=180/pi 度；当 X1 为 TRUE 时，SIN 执行余弦运算，将 X3 的余弦值输出到 X7 中，所以 X7=0.54。

正切函数 TAN

返回一个数的正切值。数据以弧度计算。指令如下图：

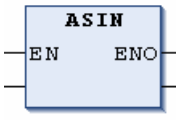


输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。
程序例：

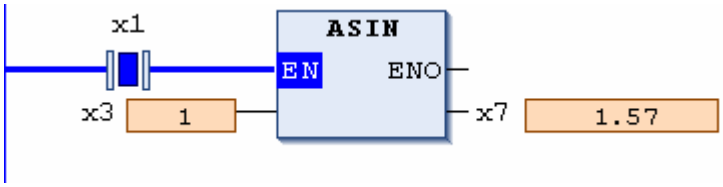


分析：在本例中，X3=1 弧度=180/pi 度；当 X1 为 TRUE 时，SIN 执行正切运算，将 X3 的正切值输出到 X7 中，所以 X7=1.56。

反正弦函数 ASIN
返回一个数的反正弦值，数据以弧度计算。指令如下图：

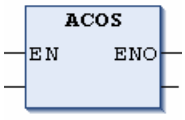


输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。
程序例：

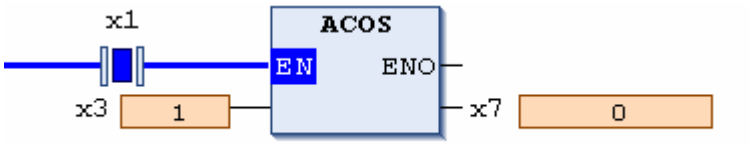


分析：在本例中，X3=1 弧度=180/pi 度；当 X1 为 TRUE 时，ASIN 执行反正弦运算，将 X3 的反正弦值输出到 X7 中，所以 X7=1.57。

反余弦函数 ACOS
返回一个数的反余弦值，数据以弧度计算。指令如下图：



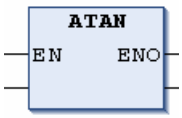
输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。
程序例：



分析：在本例中，X3=1 弧度=180/pi 度；当 X1 为 TRUE 时，ACOS 执行反余弦运算，将 X3 的反余弦输出到 X7 中，所以 X7=0。

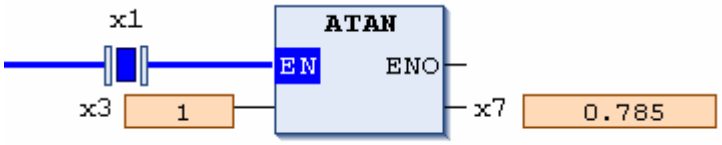
反正切函数 ATAN

返回一个数的反正切值。数据以弧度计算。指令如下图：



输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。

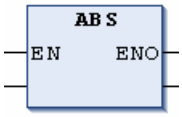
程序例：



分析：在本例中，X3=1 弧度=180/pi 度；当 X1 为 TRUE 时，ATAN 执行反正切运算，将 X3 的反正切值输出到 X7 中，所以 X7=0.785。

取绝对值函数 ABS

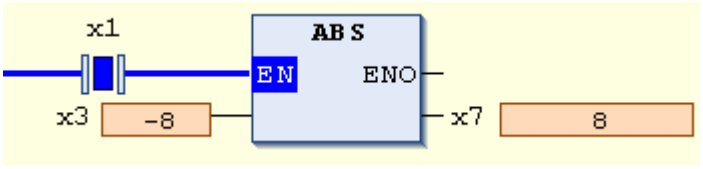
返回一个数的绝对值。指令如下图：



可以使用下列输入输出变量类型组合：

输入	输出
INT	INT，REAL，WORD，DWORD，DINT
REAL	REAL
BYTE	INT，REAL，BYTE，WORD，DWORD，DINT
WORD	INT，REAL，WORD，DWORD，DINT
DWORD	REAL，DWORD，DINT
SINT	REAL
USINT	REAL
UINT	INT，REAL，WORD，DWORD，DINT，UDINT，UINT

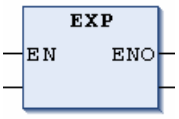
DINT　REAL，DWORD，DINT
UDINT　REAL，DWORD，DINT，UDINT
程序例：



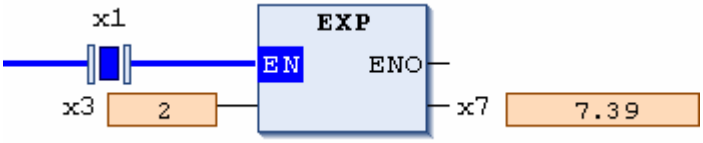
，

分析：在本例中，当 X1 为 TRUE 时，ABS 指令执行，将 X3 的绝对值输出到 X7 中；如 X3=−8，则输出 X7=8。

指数函数 EXP
返回指数函数。指令如下图：

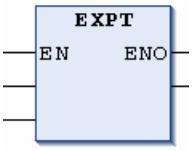


输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。
程序例：

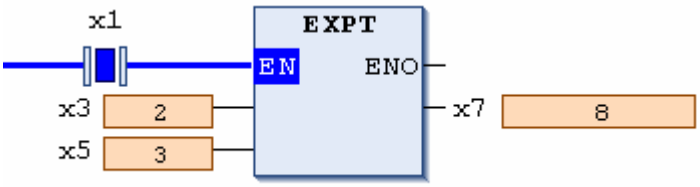


分析：在本例中，当 X1 为 TRUE 时，EXP 指令执行，将 X3 的指数输出到 X7 中；如 X3=2，则输出 X7=7.39

幂函数 EXPT
求一个变量关于另一个变量的幂。指令如下图：



两个操作数可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。
程序例：

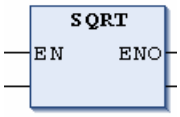


分析：在本例中，当X1 为TRUE时，EXPT指令执行，把X3 的X5 次幂输出到X7 中；如X3=2，

X5=3，则X7=2^3=8。

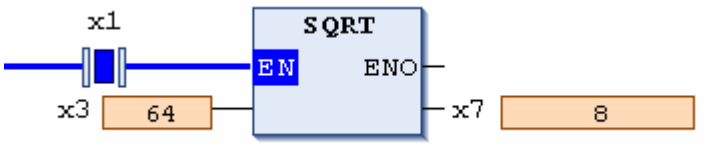
取平方根函数 SQRT

返回一个数的平方根。指令如下图：



输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能为 REAL 类型。

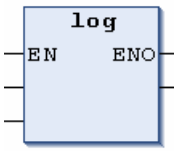
程序例：



分析：在本例中，当 X1 为 TRUE 时，SQRT 指令执行，把 X3 的平方根值输出到 X7 中；如 X3=64，则 X7=8。

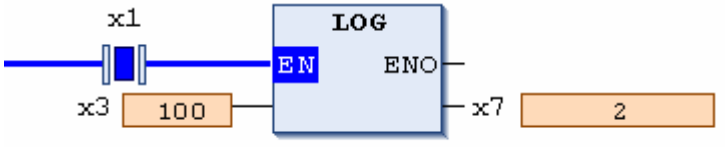
对数函数 LOG

返回值是以 10 为底的对数。指令如下图：



输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。

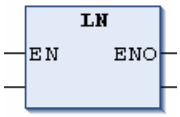
程序例：



分析：在本例中，当 X1 为 TRUE 时，LOG 指令执行，把 X3 的以 10 为底的对数结果输出到 X7 中；如 X3=100，则 X7=2。

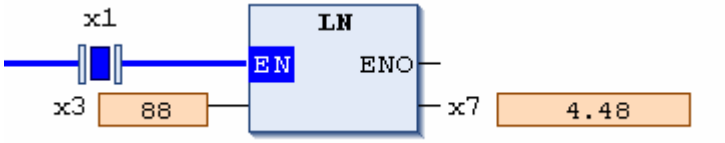
自然对数函数 LN

返回一个数的自然对数。指令如下图：



输入变量可以是 BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT 类型。输出变量只能是 REAL 类型。

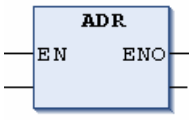
程序例：



分析：在本例中，当 X1 为 TRUE 时，LN 指令执行，把 X3 的自然对数结果输出到 X7 中；如 X3=88，则 X7=4.48。

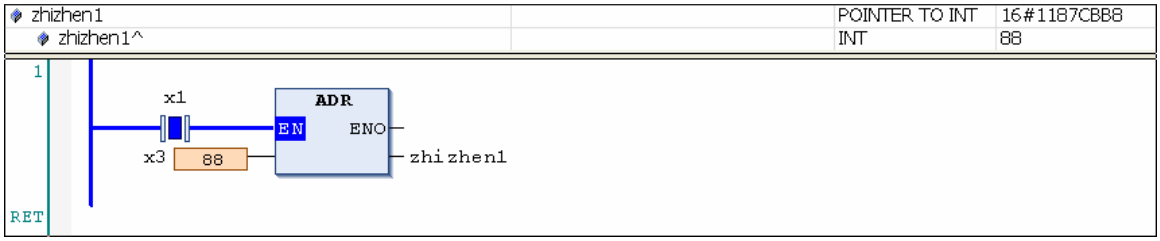
取地址指令 ADR

取地址指令。指令如下图：



ADR返回变量自身的地址，数据类型为DWORD。这个地址可以作为指针传递给操作函数，也可以赋给工程内的某个指针。

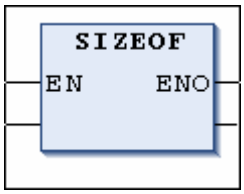
程序例：



分析：在本例中，当 X1 为 TRUE 时，ADR 指令执行，将 X3 的地址赋予指针变量 zhizhen1 上，即 zhizhen1 指向了 X1；如 X3=88，则指针变量 zhizhen1 的值 16#1187CBB8 即是 X1 的地址，同时指针的指向变量的值 zhizhen1^是 88。

字节长度指令 SIZEOF

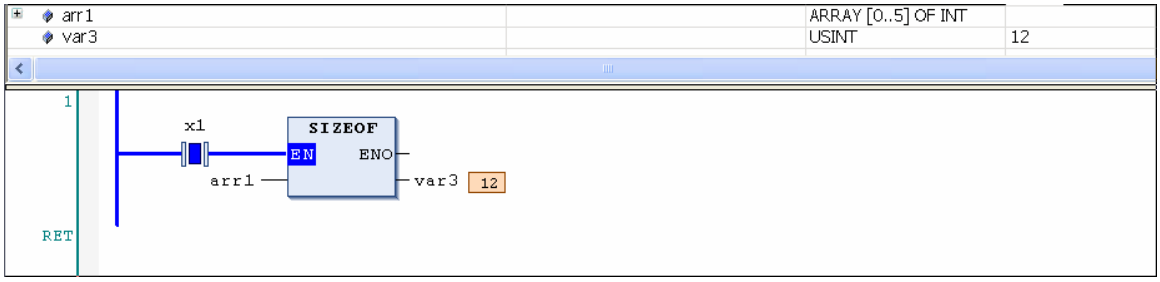
这个操作符用来确定给定变量 x 需要占用多少个字节。指令如下图：



SIZEOF 操作符通常返回一个无符号数。返回值的类型与变量 x 的大小相匹配。

SIZEOF(x)的返回值	返回值的类型
0 <= x 的值 < 256	USINT
256 <= x 的值 < 65536	UINT
65536 <= x 的值 < 4294967296	UDINT
4294967296 <= x 的值	ULINT

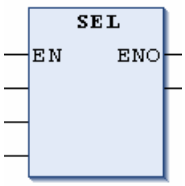
程序例：



分析：在本例中，当 X1 为 TRUE 时，SIZEOF 指令执行，来确定给定数组变量 arr1 需要占用多少个字节，并将结果输出到 var3（USINT 型）；由于 arr1 是 0 到 5 的 INT 型数组变量，因此 var3=12。

二选一指令 SEL

从两个操作数中选择一个。指令如下图：



由 G 决定 IN0 还是 IN1 为输出。

OUT := SEL(G, IN0, IN1) 的含义：

OUT := IN0; 若 G=FALSE

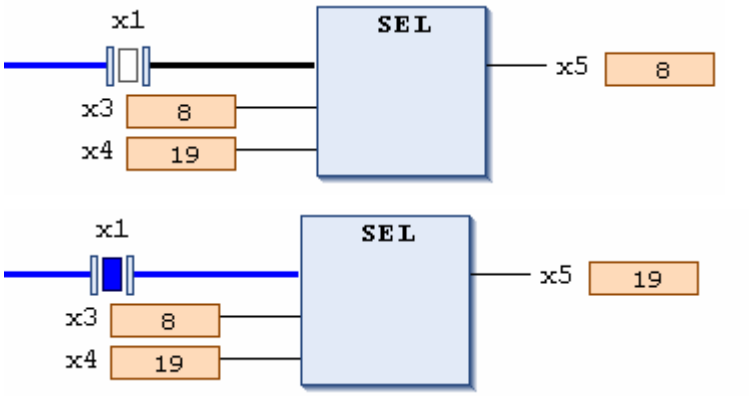
OUT := IN1; 若 G=TRUE.

允许的数据类型：

IN0，IN1，OUT：任意类型

G：BOOL.

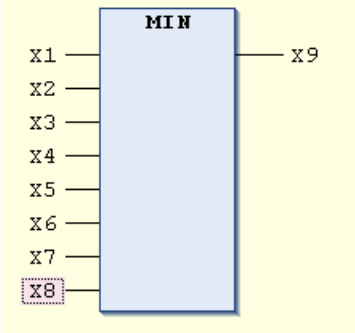
程序例：



分析：在本例中，当 X1 为 FALSE 时，SEL 指令选择 X3 输出到 X5 中，所以 X5=X3=8；当 X1 为 TRUE 时，SEL 指令选择 X4 输出到 X5 中，所以 X5=X3=19。

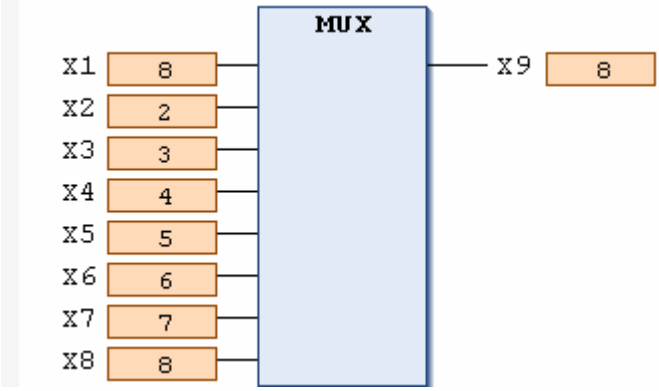
多选一指令 MUX

多项选择操作符。指令如下图：



IN0、. . . ,INn 以及 OUT 可以是任意类型的变量。X1 必须为 BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT 或 UDINT 类型。MUX 从这一组值中选择第 X1 个值。

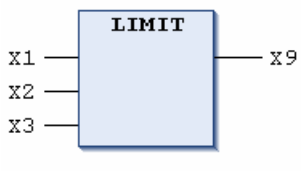
程序例：



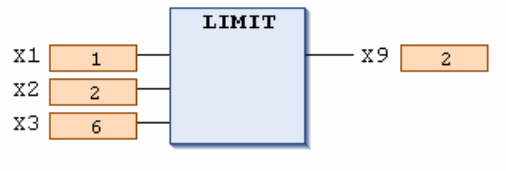
分析：在本例中，MUX 指令根据 X1 的值，来决定 X9 的输出值。当 X1=8 时，MUX 取功能块中的第 8 个值，即 X9=X8=8。

取极限指令 LIMIT

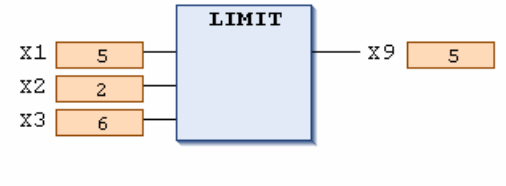
取极限。指令如下图：



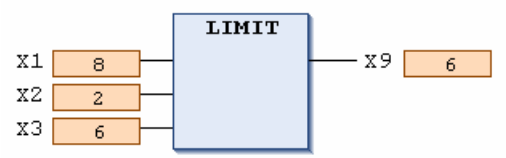
程序例：
当 X1<X2 时



X2<X1<X3 时



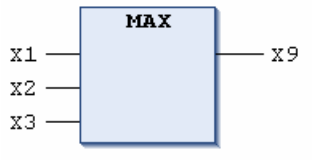
X1>X3 时



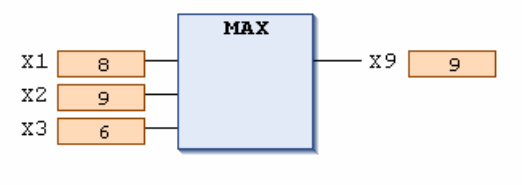
分析：Max 是结果的上限值，Min 是结果的下限值。如果 IN 值大于上限值 Max，LIMIT 将返回 Max，而如果 IN 小于 Min，那么结果为 Min。在本例中，当 X1<X2<X3 时，输出 X9=X2；当 X2<X1<X3 时，输出 X9=X1；当 X1>X3>X2 时，输出 X9=X3。

取最大值 MAX

取最大值函数。返回输入的值中最大的那一个。指令如下图：



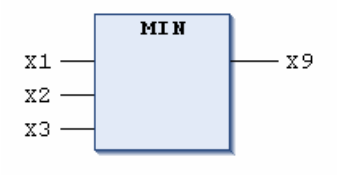
IN0，IN1,IN2 和 OUT 可以为任意类型的变量。
程序例：



分析：程序运行时，MAX 指令取输入的 X1,X2,X3 中最大的值，并将结果输出到 X9 中。在本例中 X2 最大，因此 X9=X2=9。

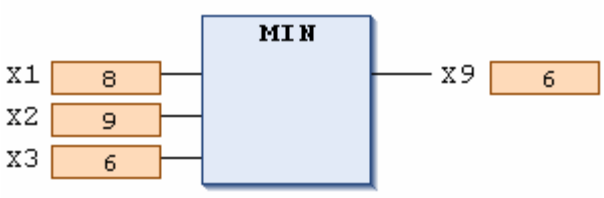
取最小值指令 MIN

取最小值函数。返回两个值中较小的那一个。指令如下图：



IN0，IN1,IN2 和 OUT 可以为任意类型的变量。

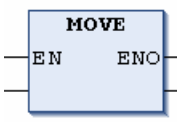
程序例：



分析：程序运行时，MIN 指令取输入的 X1,X2,X3 中最小的值，并将结果输出到 X9 中。在本例中 X3 最小，因此 X9=X3=6。

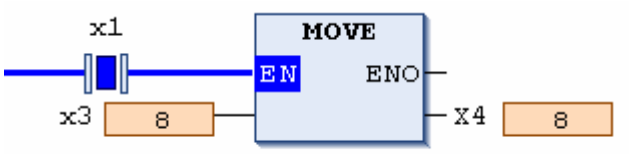
赋值指令 MOVE

将一个变量的值赋给另一个适当类型的变量。指令如下图：



在图形编辑器 FBD、LD、CFC 中，MOVE 是一个方框。在这个方框里（未锁定的）EN/ENO 功能也可以用于变量赋值。

程序例：



分析：在本例中，当 X1 为 TRUE 时，MOVE 指令执行，将 X3 的值赋给到 X4 中；即 X4=X3=8。

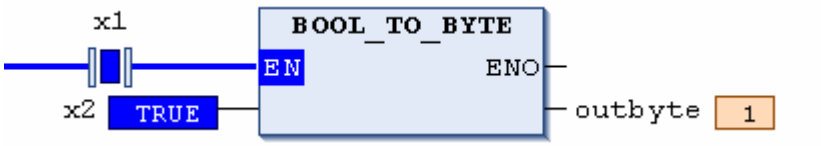
布尔类型转换指令

从布尔类型转换为其它任意类型。

BOOL_TO_<数据类型>转换为数字类型时，若操作数为 TRUE，结果为 1；若操作数为 FALSE，结果为 0。

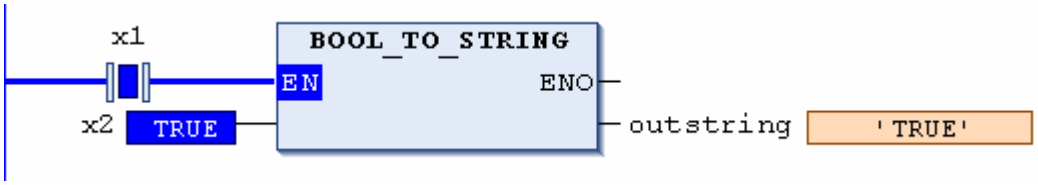
BOOL_TO_<数据类型>转换为字符串类型时，若操作数为 TRUE，结果为“TRUE”，若操作数为 FALSE，则结果为“FALSE”。

程序例 1：



分析：在本例中，当 X1 为 TRUE 时，BOOL_TO_BYTE 指令执行，输出结果 outbyte；由于 X2=TRUE，所以 outbyte=1 。

程序例 2：



分析：在本例中，当 X1 为 TRUE 时，BOOL_TO_STRING 指令执行，输出结果 outstring；由于 X2=TRUE，所以 outstring=TRUE。

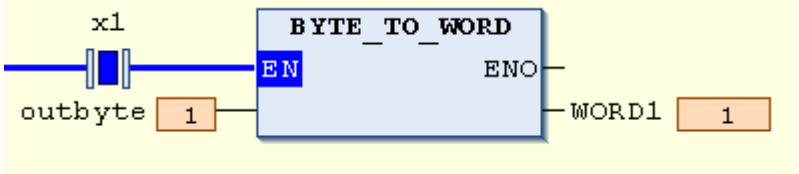
字节类型转换指令

BYTE_TO_<数据类型>转换为数字类型时，若操作数为 TRUE，结果为 1；若操作数为 FALSE，结果为 0。

BYTE_TO_<数据类型>转换为布尔类型时，若操作数为 TRUE，结果为 TRUE；若操作数为 FALSE，结果为 FALSE。

BYTE_TO_<数据类型>转换为字符串类型时，若操作数为 TRUE，结果为“TRUE”，若操作数为 FALSE，则结果为“FALSE”。

程序例：

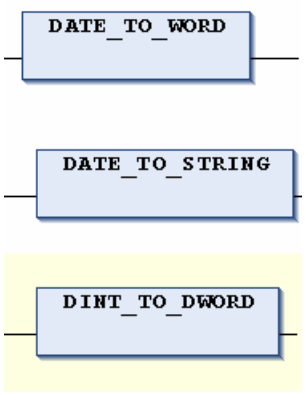


分析：在本例中，当 X1 为 TRUE 时，BYTE_TO_WORD 指令执行，输出结果 WORD1；由于 OUTBYTE=1，所以 WORD1=1。

日期转换指令

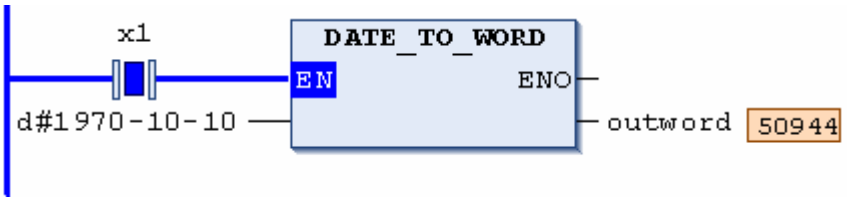
从日期和日期时间类型变量转换为其他类型变量

指令如下图：



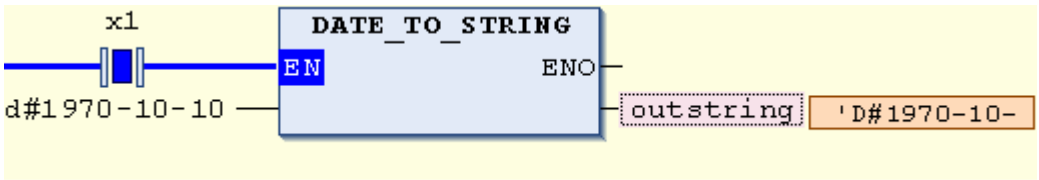
日期以秒为单位，用 DWORD 数据类型从 1970 年 1 月 1 日起存储在内部。然后再进行转化。从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。转换为字符串类型变量时，转换结果为日期常量。

程序例 1：



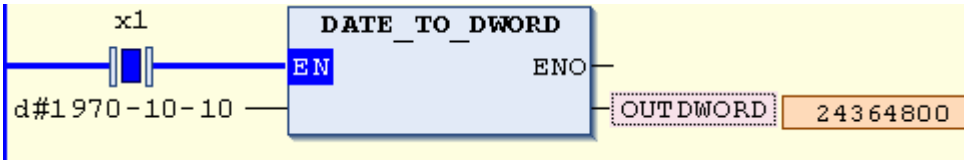
分析：在本例中，当 X1 为 TRUE 时，DATE_TO_WORD 指令执行，输出结果 OUTWORD；由于输入日期是 1970-10-10，所以 outword=50994。

程序例 2：



分析：在本例中，当 X1 为 TRUE 时，DATE_TO_STRING 指令执行，输出结果 OUTSTRING；由于输入日期是 1970-10-10，所以 OUTSTRING=D#1970-10-10。

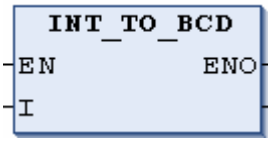
程序例 3：

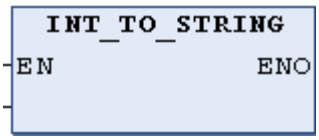


分析：在本例中，当 X1 为 TRUE 时，DATE_TO_DWORD 指令执行，输出结果 OUTDWORD；由于输入日期是 1970-10-10，所以 OUTDWORD=24364800。

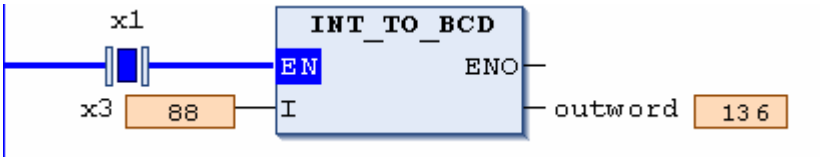
整数转换指令

INT_TO_数据类型，是将整型数据转成其他数据类型。指令如下图：

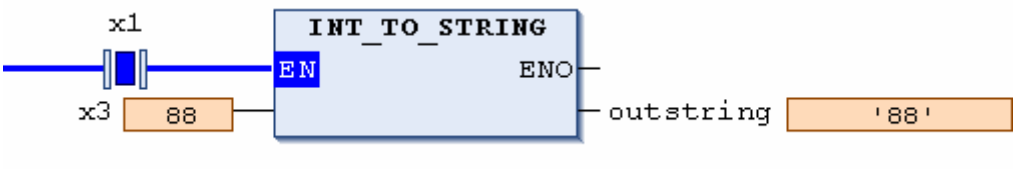




程序例 1：



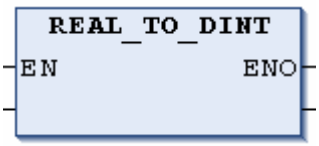
分析：在本例中，当 X1 为 TRUE 时，INT_TO_BCD 指令执行，输出结果 OUTWORD；由于输入 X3=88，所以 OUTWORD=136。



分析：在本例中，当 X1 为 TRUE 时，INT_TO_STRING 指令执行，输出结果 OUTSTRING；由于输入 X3=88，所以 OUTSTRING=’88’。

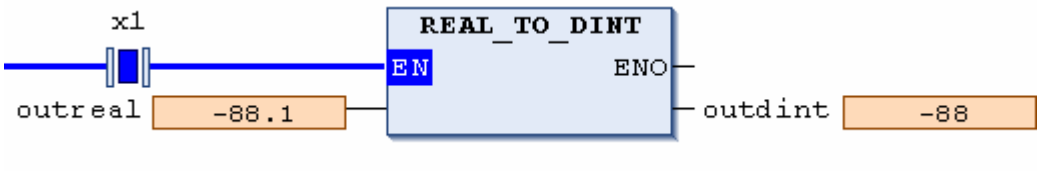
实数/长实数类型转换

从实数/长实数变量类型转换为其它类型。指令如下图：



数值将被四舍五入为近似的整数值，然后转换成新的变量类型。请注意转换为字符串类型时，（长）实数的位数不能超过 16 位。如果位数太多，那么第十六位将被四舍五入。如果定义的字符串长度比较短，那么数字的右侧部分将被截去。

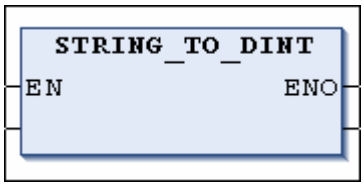
程序例：



分析：在本例中，当 X1 为 TRUE 时，REAL_TO_DINT 指令执行，输出结果 OUTDINT；由于输入 X3=-88.1，数值将被四舍五入，所以 OUTDWORD=-88。

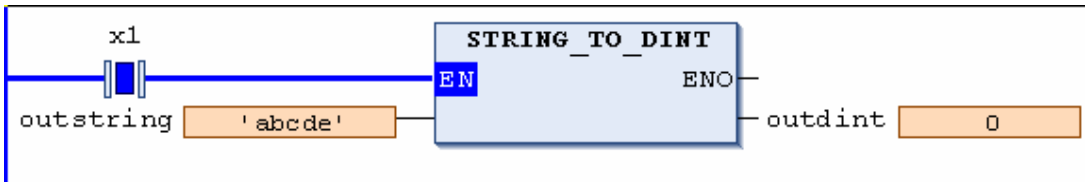
字符串类型转换命令

STRING_TO_<数据类型>。把字符串类型变量转换为其它类型。指令如下图：



先把 STRING 转换为 INT 类型变量，然后把 INT 转换为 BYTE 类型。由于高字节将被截去，因此结果将介于 0-255 之间。STRING 类型变量的操作数中必须包含一个在目标类型变量里有效的值，否则转换的结果为 0。

程序例：



分析：当 X1 为 TRUE 时，STRING_TO_DINT 指令执行，输出结果 OUTDINT；由于输入 OUTSTRING=’abcde’，不在 DINT 的数据类型中，所以 OUTDINT=0。