



EasyLad梯形图语言

编
程
手
册

(V 6 . 0)



前言

- 本资料包括 EasyLad 梯形图语言的各种指令的解释及编程方法。
- 在安装使用前，必须确认已仔细研究过本手册。
- 有关梯形图编辑环境的使用方法请参照《EasyLad 梯形图集成编辑器使用手册》（另外立册）。

声明

在不于通知的情况下保留对本手册进行修改的权利。

目 录

第一章 内部指令编程系统.....	1
1.1 指令的基本格式.....	1
1.2 编程语言.....	1
1.2.1 语句表.....	1
1.2.2 梯形图.....	2
1.2.3 画梯形图的几点注意事项.....	5
1.2.4 梯形图程序结构.....	6
1.3 内部软元件介绍.....	7
1.3.1 输入/输出继电器 X、Y.....	7
1.3.2 辅助继电器 M.....	8
1.3.3 常数 K/H/字符/字符串.....	10
1.3.4 内部定时器 T.....	11
1.3.5 IEC 定时器 (TON/TOF/TP).....	16
1.3.6 具有外部时钟输入的扩展定时器.....	22
1.3.7 普通计数器 C.....	23
1.3.8 高速计数器.....	25
1.3.9 IEC 计数器 (CTU/CTD).....	31
1.3.10 映像寄存器 R.....	33
1.3.11 临时数据寄存器 D.....	35
1.3.12 数据存储器 DM/LDM/FDM.....	36
1.3.13 变址寻址.....	40
1.3.14 位元件间接寻址.....	43
1.3.15 数据堆栈空间.....	44
1.3.16 中断及中断源.....	45
1.3.17 数据 (字符串) 表.....	47
1.3.18 字元件的双字组合 (@/!).....	49
1.3.19 多功能高速输入 X5、X6.....	50
1.4 基本逻辑指令.....	52
1.4.1 逻辑取及输出线圈 (LD/LDI/OUT).....	52
1.4.2 触点串联 (AND/ANI).....	52
1.4.3 触点并联 (OR/ORI).....	53
1.4.4 串联电路块的并联 (ORB).....	54
1.4.5 并联电路块的串联 (ANB).....	54
1.4.6 多重输出电路 (MPS/MRD/MPP).....	55
1.4.7 取反触点 (NOT).....	56
1.4.8 边缘检测触点 (PED/NED).....	57
1.4.9 比较触点 (=、<、>、<=、>=、<>).....	59
1.4.10 置位、复位与取反线圈 (SET/RST/INV).....	61
1.4.11 多点置位和多点复位线圈 (MSET/MRST).....	62
1.4.12 复位置位线圈 (RS).....	63
1.4.13 主控线圈 (MC/MCR).....	64
1.4.14 空操作指令 (NOP).....	65

1.4.15	程序结束 (END)	65
1.5	步进指令	66
1.5.1	步进梯形图的介绍	66
1.5.2	步进梯形图的详细动作	69
1.5.3	分支和汇合的编程	72
1.6	功能指令	74
1.6.1	主程序结束指令 (FEND 工作/MEND 调试)	74
1.6.2	程序暂停 (放置断点) 指令 (STOP)	75
1.6.3	立即跳转指令 (JMP)	76
1.6.4	减 1 不为 0 跳转指令 (DJNZ)	76
1.6.5	子程序指令 (CALL/SUBS/SRET/CRET)	77
1.6.6	编程口通讯地址设置指令 (CADDR)	78
1.6.7	中断处理指令 (TIM/INT/IRET/CRETI/EI/DI)	79
1.6.8	脉冲输出指令 (PTO/ PWM/PLS)	82
1.6.9	高速计数器及其高速输出指令 (HCN/HCR/HCO)	86
1.6.10	脉冲宽度、周期、频率测量指令 (PWD/HDT/HDF)	87
1.6.11	I/O 刷新指令 (REF/REFL/REFH)	90
1.6.12	查表和表格数据定义指令 (LDTAB/DW)	91
1.6.13	非易失性数据存储器读写指令 (RAMRD/RAMWR)	92
1.6.14	进栈和出栈指令 (PUSH/POP)	93
1.6.15	数据传送指令 (MOV/LMOV/HMOV)	95
1.6.16	字交换指令 (XCH)	96
1.6.17	半字交换指令 (LXCH/HXCH/SWAP)	96
1.6.18	加法指令 (ADD)	97
1.6.19	减法指令 (SUB)	98
1.6.20	乘法指令 (MUL)	98
1.6.21	除法指令 (DIV)	99
1.6.22	脉冲滤波定时器指令 (TPF)	100
1.6.23	毫秒延时器指令 (TMS)	100
1.6.24	可重触发脉冲定时器指令 (TPR)	101
1.6.25	增 1 和减 1 指令 (INC/DEC)	102
1.6.26	数据移位指令 (RLC/RRC/ROL/ROR/SHLA/SHRA/LSR)	103
1.6.27	逻辑与指令 (AND)	107
1.6.28	逻辑或指令 (OR)	108
1.6.29	逻辑异或指令 (XOR)	108
1.6.30	数据取反指令 (CPL)	109
1.6.31	数据转换指令 (BCD/BIN/NEG/ABS/ROT)	110
1.6.32	译码与编码指令 (DECO/ENCO)	113
1.6.33	数据分离与组合指令 (SPLIT/UNITE)	114
1.6.34	ASCII 码转换指令 (HTA/ATH)	115
1.6.35	字和字节转换指令 (WTOB/BTOW)	116
1.6.36	数据块操作指令 (BMOV/ BRMOV/ BXMOV/FILL/BLOD/BCMP)	117
1.6.37	数据保存指令 (DSAVE)	121
1.6.38	扫描时间测量指令 (SCANT)	122

1.6.39	SPI 接口指令 (SSET/SCLR/SOUT/SIO/SOUB/SIOB/SPAO)	123
1.6.40	IIC 接口指令 (IICSTART/IICSTOP/IICW/IICRACK/IICRNAK)	129
1.7	宏指令	131
1.7.1	预定义指令 (EQU/AS)	132
1.7.2	预定义文件包含指令 (INCLUDE)	132
1.7.3	注释指令 (;)	133
1.7.4	程序连接指令 (LINK)	134
1.7.5	取标号的步数 (\$)	135
1.7.6	取元件的编号或变量的地址 (#)	135
1.8	结构化的程序流控制指令	137
1.8.1	DO/DOEND 指令	137
1.8.2	NJP/NJPE 指令	138
1.8.3	BLOCK/BREAK/BEND 指令	139
1.8.4	LABEL/EXITLOOP/LOOP 指令	141
1.8.5	FOR/EXITFOR/NEXT 指令	142
1.9	变量定义、表达式和函数	144
1.9.1	数据类型和存储类型	144
1.9.2	使用 EQU/AS 定义变量	146
1.9.3	数组	148
1.9.4	结构体	154
1.9.5	使用全局符号表来定义全局变量	160
1.9.6	表达式及其运算符	162
1.9.7	函数的调用形式	165
1.9.8	自定义函数	166
1.9.9	使用函数参数表编辑自定义函数的参数	173
1.10	内部函数	176
1.10.1	变址函数	176
1.10.2	数据类型转换函数 (L / F)	177
1.10.3	取绝对值函数 (IABS / FABS)	178
1.10.4	取负函数 (INEG / FNEG)	179
1.10.5	长整数高低字交换函数 (DSWAP)	180
1.10.6	浮点数开平方函数 (SQRT)	181
1.10.7	读线圈和写线圈函数 (RdCoil / WrCoil / RdLCoil / WrLCoil)	182
1.10.8	编程通讯口设置函数 (CommSet)	184
1.10.9	编程口 ModBus-RTU 通讯函数	186
1.10.10	编程口自由通讯发送函数	192
1.10.11	编程口自由通讯接收函数	198
1.10.12	32 位数移位函数 (ROL / ROR / ASL / ASR / LSL / LSR)	203
1.10.13	取模 (取余数) 函数 (IMOD)	206
1.10.14	取整数值的最低字节有符号数值函数 (BYTE)	206
1.10.15	DM 存储器区按字节读写函数 (RB / WB / RW / WW / RL / WL)	207
1.10.16	指数和对数函数 (FEXP / FLN / FLG)	210
1.10.17	三角函数 (FCOS / FSIN / FTAN)	211
1.10.18	反三角函数 (FACOS / FASIN / FATAN)	212

1.10.19	双曲函数 (FCOSH/FSINH/FTANH)	214
1.10.20	取字符串变量字符长度函数 (STRLEN)	215
1.10.21	字符串变量加载字符串常数函数 (STRLOD)	215
1.10.22	字符串变量与字符串变量传送函数 (STRMOV)	216
1.10.23	字符串变量比较函数 (STRCMP/STRCMPV)	216
1.10.24	字符串变量相加函数 (STRADD/STRRAD)	217
1.10.25	字符串变量与字节字符串转换函数 (STRASCB/ASCBSTR)	218
1.10.26	字符串变量与 UTF8 字符串转换函数 (STRUTF8/UTF8STR)	220
1.10.27	整型数值转换为字符串变量函数 (VALSTR)	221
1.10.28	字符串字节格式转换数值函数 (STRTOF/STRTOL)	221
1.10.29	32 位无符号除法函数 (UDIV)	222
1.10.30	32 位无符号取模 (取余数) 函数 (UMOD)	223
第二章	外部函数	224
2.1	如何添加函数库连接和创建函数库	224
2.2	数据存储块操作函数库 (DMBlock.yf)	226
2.2.1	块传送函数 (BMOV)	226
2.2.2	块右向传送函数 (BRMOV)	226
2.2.3	块高低字节交换传送函数 (BXMOV)	227
2.2.4	块加载数据表函数 (BLOD)	228
2.2.5	块填充函数 (FILL)	228
2.2.6	块比较函数 (BCMP)	229
2.2.7	非易失存储器读函数 (RAMRD)	230
2.2.8	非易失存储器写函数 (RAMWR)	230
2.2.9	添加数据到队列函数 (ATT)	231
2.2.10	先进先出移出数据函数 (FIFO)	232
2.2.11	后进先出移出数据函数 (LIFO)	232
2.2.12	获得队列表中数据的个数函数 (GTDN)	233
2.2.13	查找为 ON 线圈位置函数 (FindON)	234
2.2.14	DM 存储器块高低字节交换函数 (SWAP)	235
2.2.15	DM 存储器块字节分离函数 (SPLIT)	235
2.2.16	DM 存储器块字节组合函数 (UNITE)	236
2.2.17	DM 存储器块字转字节函数 (WTOB)	237
2.2.18	DM 存储器块字节转字函数 (BTOW)	237
2.2.19	时间测量函数 (MTS/MTE)	238
2.2.20	按字节地址块复制函数 (MEMCPY)	240
2.2.21	数据块求 CRC16 校验函数 (CRC16)	240
2.2.22	字符串比较触点函数 (strEQ/strNE)	241
2.2.23	实时时钟比较函数 (RTCCMP)	242
第三章	配方和记录	244
3.1	配方设置	244
3.1.1	配方设置说明	244
3.1.2	单组参数设置例子	248
3.1.3	多组配方设置例子	249
3.1.4	数据记录保存例子	250

3.1.5 数据记录显示例子.....252

附录 A 边缘检测触点使用个数溢出处理253

附录 B 保持运行状态写程序注意事项253

第一章 内部指令编程系统

1.1 指令的基本格式

EasyLad 梯形图语言的指令的基本格式为：

指令码 [**操作数 1**], [**操作数 2**], ……

“指令码”是必选项，它表示执行哪一种操作或者运算。

“操作数 1~2 ……”是可选项，它表示哪个软设备（或常数）参与该操作。

对于不同的指令码，其操作数的个数也可能不同，有的指令不需要操作数，有的指令需要 1 个操作数，有的指令需要 2 个操作数。

指令码和操作数之间用空格隔开，操作数和操作数之间用逗号“,” 隔开。

例：

无操作数指令：END

1 个操作数指令：SET Y0

2 个操作数指令：ADD D0, K100

1.2 编程语言

1.2.1 语句表

语句表为文本型的编程语言，它有若干行指令行组成，其指令行的一般格式为：

[步数] [标号:] [指令] [; 注释]

“步数”为可选项，表示该行指令在整个程序中所在的步数。

“标号”为可选项，它可用来表示跳转指令或子程序调用指令（CALL）或表格的入口位置及自定义的函数名等。

“指令”为可选项，它表示该指令行执行哪一种操作，其格式见上节。若无该项，则表示该行为空行或注释行。

“注释”为可选项，添加在程序中以增加程序的可读性。

步数与标号之间用空格隔开，标号和指令之间用“:” 隔开，“;” 后面的均为注释。

例：

；语句表举例

```

LD X0      ; 起始位置
AND M0
OUT Y0
LD X1
JMP LABEL1
LDI X2
OUT T0, K100
LABEL1: LD X3      ; LABEL1 为标号
ADD D0, K10
END

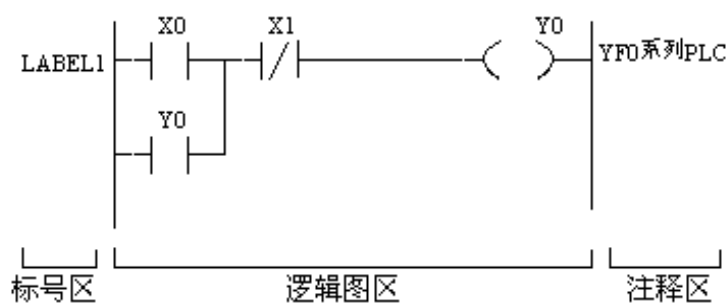
```

语句表举例

从上图可以看出，这样的文本型的程序和汇编语言程序非常相似，CPU 从上到下按照程序的次序执行每一条指令，程序结束后再返回到起始位置继续执行。

1.2.2 梯形图

(1) 梯形图是一种类似继电器控制电路图的图形式编程语言。EasyLad 梯形图主要由 3 大区域组成：标号区、逻辑图区、注释区。如下图：



梯形图的组成

标号区：标号区位于左母线的左边，它可用来表示跳转指令或子程序调用指令（CALL）或表格的入口位置（名称）及自定义的函数名等。标号中不能有“（”、“）”、“\$”、“#”等特殊符号，另外，标号“TABLE”已为系统所使用，用户不得再使用。

逻辑图区：逻辑图区位于左母线和右母线之间，为梯形图的逻辑电路图。

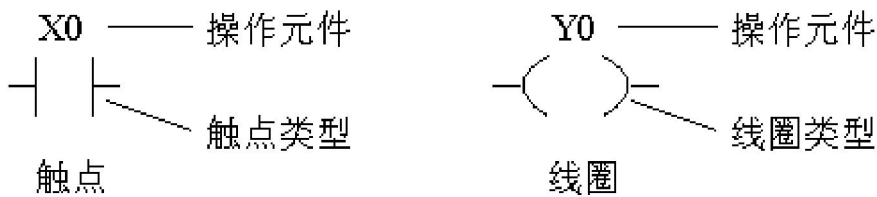
注释区：注释区位于右母线的右边，为梯形图的文字说明。

梯形图是按自上而下，从左至右的方式运行的。

梯形图是基于继电器梯形逻辑电气图的软件仿真。在梯形图中，有一个提供能量的左母线和一個收集能量的右母线。触点闭合可以使能量流过该元件，到下一个元件，触点打开将阻止能量流断过。线圈为能量的消耗元件，能量流必须经过线圈才能流入右母线。当线圈有能量流流过时（当前的逻辑运算结果为 ON），则该线圈被接通；当线圈无能量流流过时（当前的逻辑运算结果为 OFF），则该线圈被断开。

梯形图中的线圈可以并联，但不能串联，并且线圈只能配置在每一逻辑行的最右边。

(1) 逻辑图区主要有常开触点、常闭触点、取反触点、正边缘检测触点、负边缘检测触点、步进触点、输出线圈、指令线圈和连接线等元素组成。



触点和线圈的一般形式

- 常开触点



当对应的操作元件（继电器）为 1（线圈接通）时，该触点闭合。

该触点的操作元件为输入继电器 X、输出继电器 Y、辅助继电器 M、定时器 T、计数器 C 及数据寄存器 D 中的某一位。

- 常闭触点



当对应的操作元件（继电器）为 0（线圈断开）时，该触点闭合。

该触点的操作元件为输入继电器 X、输出继电器 Y、辅助继电器 M、定时器 T、计数器 C 及数据寄存器 D 中的某一位。


- 取反触点



当与该触点串联的左边的电路块为 ON（接通）时，经过该触点的逻辑运算结果为 OFF（断开）；当与该触点串联的左边的电路块为 OFF（断开）时，经过该触点的逻辑运算结果为 ON（接通）。


该触点无操作元件。

- 正边缘检测触点

 当且仅当该触点检测到与该触点串联的左边的电路块发生从 OFF（断开）到 ON（接通）的正跳变时，才使经过该触点的逻辑运算结果为 ON（接通），在其他情况下，经过该触点的逻辑运算结果为 OFF（断开）。


该触点无操作元件。

- 负边缘检测触点

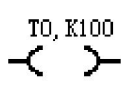
 当且仅当该触点检测到与该触点串联的左边的电路块发生从 ON（接通）到 OFF（断开）的负跳变时，才使经过该触点的逻辑运算结果为 ON（接通），在其他情况下，经过该触点的逻辑运算结果为 OFF（断开）。

该触点无操作元件。

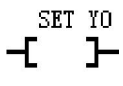
- 步进触点

 由 STL（步进开始）指令所驱动的触点，表示步进梯形程序中一个工步的开始。该触点左边只能与左母线或其他步进触点连接，母线（LD/LDI 点）移至该触点之后，除非下一条指令还是 STL 指令。该触点的操作元件只能是 M 继电器。

- 输出线圈

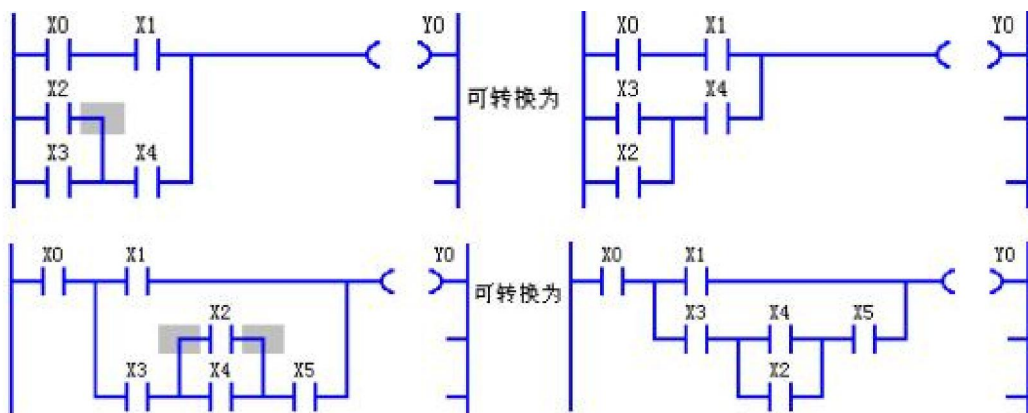
 ^{T0, K100} 表示由 OUT 指令所驱动的线圈，其操作元件为该指令的操作数，而不用该指令的操作码（OUT）。例如指令 OUT T0, K100，则对应的输出线圈的操作元件为 T0, K100。

- 指令线圈

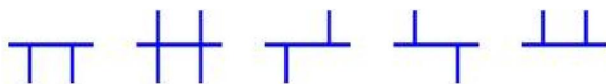
 ^{SET Y0} 表示其操作元件为一个完整的指令或表达式的线圈。若当前的逻辑运算结果为 ON，则该线圈被接通；若当前的逻辑运算结果为 OFF，则该线圈被断开。

1.2.3 画梯形图的几点注意事项

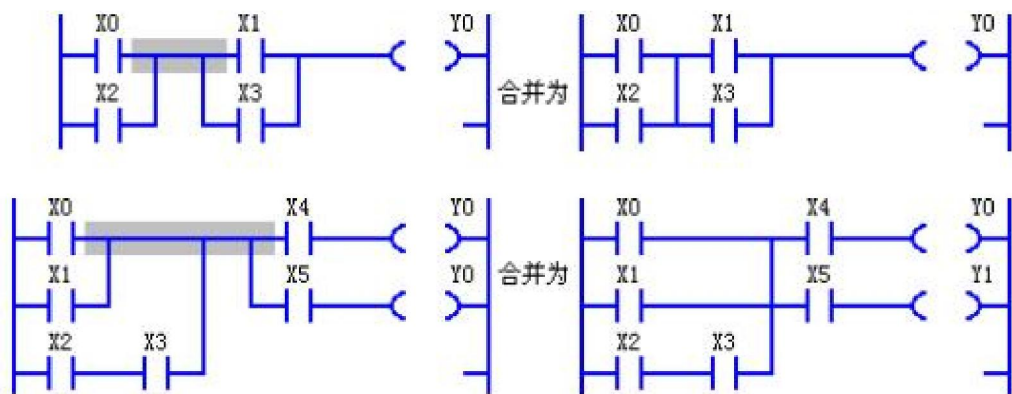
- (1) 梯形图中不允许有“┐”和“┌”型的连接线，用户应对这两种连接进行转换。例：



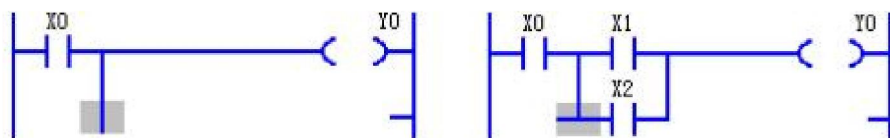
- (2) 在同一水平线上的两个垂直分支不允许直接（或用导线）连接，必须经过触点才能连接。用户必须对直接（或用导线）连接的两个（或以上）垂直分支进行合并处理。如下等连接均属非法：



必须对这些连接进行合并处理。例：



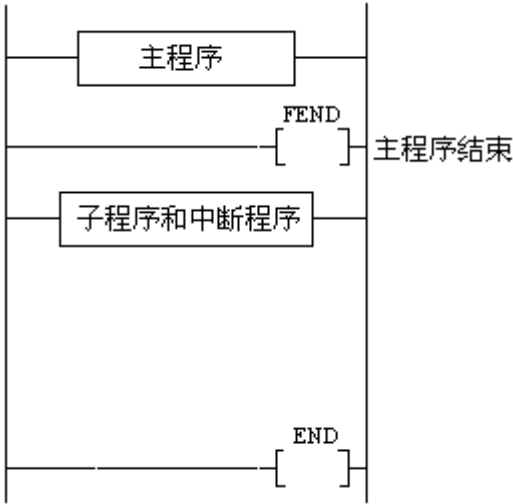
- (3) 在梯形图中不允许存在断线。例：



1.2.4 梯形图程序结构

梯形图程序结构由主程序、子程序、函数、中断程序组成，主程序会被系统自动循环扫描执行，子程序和函数被调用时才执行、不被调用时不会执行，中断程序是当对应的中断源事件发生时由系统自动执行。

从梯形图开始到 FEND 指令处为主程序区，FEND 指令到 END 指令处为子程序、函数、中断程序区。其程序结构图如下：



梯形图程序结构图

1.3 内部软元件介绍

1.3.1 输入/输出继电器 X、Y

- 输入继电器 (X0~X57)

输入继电器采用八进制编号。

输入继电器最多可达 48 点。

PLC 的输入端子是从外部开关接收信号的窗口。

PLC 内部与输入端子连接的输入继电器(X)是光电隔离的电子继电器，其常开触点(NO)和常闭触点(NC)的使用次数不限，这些触点在内部可以自由使用。

输入继电器既可以采用直接寻址方式如 X10，也可以采用寄存器位寻址方式如 RX0.8。

输入继电器不能用程序驱动。

- 输出继电器 (Y0~Y57)

输出继电器采用八进制编号。

输出继电器最多可达 48 点。

PLC 的输出端子是向外部负载输出信号的窗口。

输出继电器的外部输出触点(继电器触点，晶体管等输出元件)接到 PLC 的输出端子上。

输出继电器的电子常开和常闭触点使用次数不限，在 PLC 中可自由使用，然而外部触点(输出元件)与内部触点的动作有所不同。

输出继电器既可以采用直接寻址方式如 Y10，也可以采用寄存器位寻址方式如 RY0.8。

● 输入、输出继电器的动作时序

(1) 输入处理

程序执行前 PLC 的全部输入端子的通/断状态读入输入映像寄存器。

在程序执行中即使输入状态变化，输入映像寄存器的内容也不变。直到下一扫描周期的输入处理阶段才读入这变化。另外，输入触点从通（ON）→断（OFF）或从断（OFF）→通（ON）变化，到处于确定状态止，输入滤波器还有一响应延迟时间（约 10ms）。

(2) 程序处理

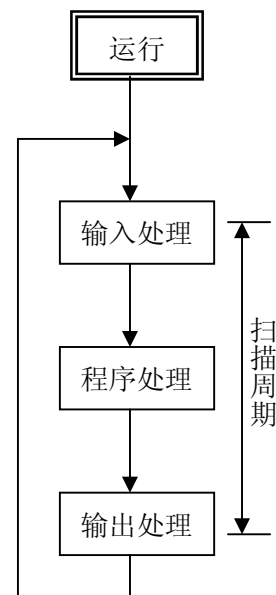
对应用户程序存储器所存的指令，从输入映像寄存器和其他软元件的映像寄存器中将有关软元件的通/断状态读出，从 0 步开始顺序运算，每次结果都写入有关的映像寄存器，因此，各软元件的映像寄存器的内容随着程序的执行在不断变化。

输出继电器的内部触点的动作由输出映像寄存器的内容决定。

(3) 输出处理

全部指令执行完毕，将输出 Y 的映像寄存器的通/断状态向输出锁存寄存器传送，成为 PLC 的实际输出。

PLC 内的外部输出触点对输出软元件的动作有一个响应时间，即要有一个延迟才动作。



1.3.2 辅助继电器 M

辅助继电器采用十进制编号。

辅助继电器为 208 点，其中通用辅助继电器 176 点，特殊辅助继电器 32 点。

辅助继电器为内部标志位存储器，可作为控制继电器存储中间操作状态或其他的控制信息，其线圈与输出继电器一样，由 PLC 内部各软元件的触点驱动。辅助继电器的电子常开和常闭触点使用次数不限，在 PLC 内可以自由使用。但是这些触点不能直接驱动外部负载，外部负载的驱动必须由输出继电器实行。

辅助继电器既可以采用直接寻址方式如 M8，也可以采用寄存器位寻址方式如 RM0.8。

- 通用辅助继电器（M0~M175）

其功能可由用户自己定义。

- 特殊辅助继电器（M176~M207）

这类继电器均有特殊的用途。其功能由 PLC 内部所定义，用户自己不能定义。M176~M177 为系统保留，用户不要使用。

每一个特殊辅助继电器均有一个名称（标识符）来表示。如 RUN 表示运行继电器 M207。

特殊辅助继电器表：（1—ON，0—OFF）

名称	编号	功 能
RUN	M207	当 PLC 为 RUN 状态时始终 1（ON），为 STOP 状态时为 0（OFF）
ERR	M206	PLC 正确/错误,当 PLC 出现错误时为 1，否则为 0
RUNP	M205	RUN 脉冲，仅在程序开始运行后的第 1 个扫描周期为 1
CLK50	M204	50HZ 时钟，周期为 20ms
CLK5	M203	5HZ 时钟，周期为 200ms
CF	M202	进位/借位继电器
OV	M201	溢出位继电器
HDF T	M200	测频率闸门时间到标记，每次闸门时间到时置 1，由用户软件清 0
	M199	保留
HSOE1	M198	高速 C1 高速输出控制继电器，为 1 时允许高速 C1 执行高速输出，为 0 时禁止
HSOE0	M197	高速 C0 高速输出控制继电器，为 1 时允许高速 C0 执行高速输出，为 0 时禁止
DF2M	M196	高速 C2 测频闸门时间选择，若 DF2M1 为 0，则为 0 时选 500ms，为 1 时选 50ms
HC2M	M195	高速 C2 自动重装载/不重装选择，为 1 选择自动重装功能，为 0 选择不重装功能
HC2F	M194	当高速 C2 为自动重装计数器时，M194 为计数次数到标志，即每当高速 C2 减为 0 或刷新频率测量值时接通 M194，高速 C2 复位并不使 M94 为 0
SPIM1	M193	SPI 接口接收方式控制继电器 1
SPIM0	M192	SPI 接口接收方式控制继电器 0
LT	M191	小于标记，执行比较指令时若第 1 操作数小于第 2 操作数则为 1，否则为 0
EQ	M190	相等标记，执行比较指令时若两操作数相等则为 1，不相等则为 0
GT	M189	大于标记，执行比较指令时若第 1 操作数大于第 2 操作数则为 1，否则为 0
INT5M	M188	中断源 5（X4 边缘触发中断）模式，0 为正边缘触发，1 为负边缘触发
INT4M	M187	中断源 4（X3 边缘触发中断）模式，0 为正边缘触发，1 为负边缘触发
DF2M1	M186	高速 C2 测频闸门时间选择 1，为 1 时：若 DF2M 为 0 选 1s，为 1 选 100ms
HC2CK	M185	高速 C2 计数时钟选择，为 0 选择外时钟（X2），为 1 选择内时钟（0.375us）
GF0	M184	通用标记 0，由用户设置或复位，可用作临时标记位
CBUSY	M183	通讯忙标记，当执行 NETR 或 NETW 等通讯函数后为 1，通讯完成后为 0
CLK10	M182	10HZ 时钟，周期为 100ms
PTO1F	M181	当 Y1 的设定个数的脉冲串输出完成后置 1，由用户软件清 0
PTO0F	M180	当 Y0 的设定个数的脉冲串输出完成后置 1，由用户软件清 0
CLK1	M179	1HZ 时钟，周期为 1 秒
PWRLF	M178	电源掉电标记，当电源掉电时为 ON，正常时为 OFF（YF0A 无该功能）
SPIHS	M177	SPI 接口高速模式，为 ON 时 SPI 为高速模式，为 OFF 时 SPI 为低速模式

注：为了产品的兼容性，建议用户使用特殊辅助继电器的名称（标识符），而不要使用其编号。其他的特殊元件也如此。

1.3.3 常数 K/H/字符/字符串

在 EasyLad 梯形图语言中，有许多指令使用常数。CPU 以二进制补码方式存储所有常数，但可用十进制或十六进制或字符串形式来表示。

十进制格式：K 十进制值。分整数和浮点数，无小数点的为整数，如 K100、K1234；有小数点的为浮点数，如 K1.234、K12.0、K-2.0E10。

十六进制格式：H0 十六进制值或者 0x 十六进制值。如 H07FEA，H0FFFF，0xAB6D，0x5EF7，而 HFFFF 则不为十六进制数。

字符格式：由单引号所引起来的一个字符。单引号内只能有一个字符，超过一个字符则错误。如 'A'，'0'，'b'。

字符串格式：由双引号所引起来的一个或多个字符。如 "字符串"，"A"，"控制器"。

在字符或字符串格式中，每个字符均占用 1 个字的存储单元，而不论该字符是 ASCII 码还是汉字。但可通过该字符的值来判断该字符是否为 ASCII 码：若其值大于 0 且小于 256（高字节为 0）则是 ASCII 码；另一个较简便的方法是可通过该值的最高位（位 15）来判断，若最高位为 0 则是 ASCII 码，若最高位为 1 则是汉字。

当字符串作为指令的操作数时，虽然字符串可以由多个字符组成，但指令总是使用字符串的第 1 个字符的编码。例如：

MOV D0,"A" 等效于 MOV D0,'A' 则执行该指令后 D0 中的内容为 65（字符“A”的编码）。

MOV D1,"控制器" 等效于 MOV D1,'控' 则执行该指令后 D1 中的内容为-16424（字符“控”的国标编码）。

当字符串在表达式中使用，则字符串是作为一个整型的地址信息来使用，即使用的是该字符串在程序中的地址。而字符格式在表达式中使用，则使用的是该字符的编码。

1.3.4 内部定时器 T

定时器有两个相关变量：当前值寄存器和定时器标志位。当前值寄存器（位 15、位 14 被清 0 后）表示定时器还剩下的时间，定时器标志位为其输出触点。对于每个定时器，这两个变量使用同一名称，但使用场合不同，其所指也不同。

(1) 非保持型定时器的动作及编号

定时器采用十进制编号。

在 PLC 内定时器是根据时钟脉冲累积计时的，时钟脉冲有 10ms、100ms、1s，当所计时间到达设定值时，其输出触点动作。定时器编号如下：

- 100ms 定时器：

T0~T23 (24 点)：设定值范围：1~16383，定时范围：0.1~1638.3s。其当前值在执行 END 指令时刷新。

T272~T1979 (1708 点)：设定值范围：1~16383，定时范围：0.1~1638.3s。其当前值在执行线圈驱动指令（OUT）时刷新。**注：YF0H4K 为 T384~T4349 (3966 点)**

- 10ms 定时器：

T0~T23 (24 点)，设定值范围：1~16383，定时范围：0.01~163.83s。其当前值由系统中断刷新。因此，当扫描周期大于 10ms 时不影响定时器的运行。

T272~T1979 (1708 点)：设定值范围：1~16383，定时范围：0.01~163.83s。其当前值在执行线圈驱动指令（OUT）时刷新。**注：YF0H4K 为 T384~T4349 (3966 点)**

- 1ms 定时器：**T24~T47 (24 点)**，设定值范围：1~16383，定时范围：1ms~16.383s。其当前值由系统中断刷新。

- 1s 定时器：**T272~T1979 (1708 点)**，设定值范围：1~16383，定时范围：1~16383s。其当前值在执行线圈驱动指令（OUT）时刷新。**注：YF0H4K 为 T384~T4349 (3966 点)**

- 自定义时基定时器：**T272~T511 (240 点)**，设定值范围：1~16383。其当前值在执行 TDEC 指令时刷新，时基由刷新 TDEC 指令周期决定。**注：YF0H4K 为 T384~T4349 (3966 点)**

定时器 T272~…与计数器 C272~…、数据存储器 DM272~…共用同一编号，即若某一编号（272~…）用作定时器，则该编号不能再用作计数器或数据存储器，反之亦然。并且使用 PLC 的数据存储器 DM272~…作为当前值寄存器。

非保持型定时器使用 OUT 指令驱动，其指令的格式如下：

OUT 定时器号，设定值，时基选择

定时器号为所驱动的定时器的编号，可使用对象为 T0~T23，T272~…。

设定值为定时器的定时时间值，可使用对象为常数、D、RM、RY、RX、RC、R T、DM。若设定值为 0，则定时器触点只延时 1 个扫描周期接通。

时基选择为可选参数，用于选择定时器的时基，分别为“1s”、“0.1s”、“0.01s”。定时器为 T272~…时，若无该参数，则表示时基为自定义时基，时基由刷新 TDEC 指令周期决定。定时器为 T0~T23（只有 0.1s 或 0.01s 两种时基）时，若无该参数则表示时基为 100ms。1ms 定时器 T24~T39 无该参数。

对于自定义时基定时器，可在 1ms 定时中断程序中使用 TDEC 指令刷新定时器当前值，则可实现 1ms 高精度定时器，可参见“1ms 定时器例子”程序。

例如：

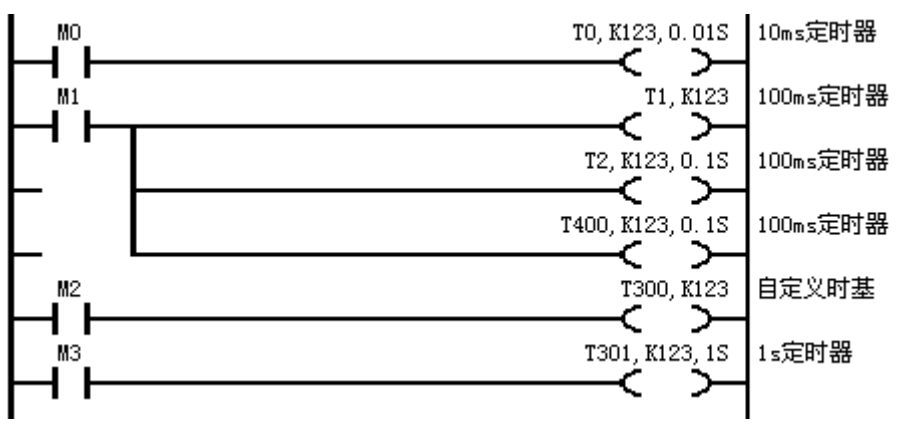
10ms 定时器： OUT T0, K123, 0.01s

100ms 定时器： OUT T10, K123 或 OUT T400, K123, 0.1s

自定义时基定时器： OUT T300, K1000

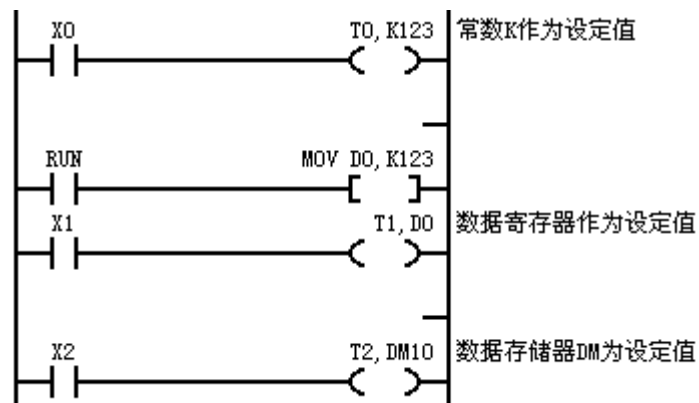
1s 定时器： OUT T310, K1234, 1s

在梯形图中表示如下：



定时器的梯形图表示

定时器可以用用户程序存储器内的常数 K 作为设定值，也可以用数据寄存器 D 或数据存储器 DM 的内容作为设定值。



作为定时器设定值的几种方法

- 常数 K 作为设定值

设定值只能在用户编制程序时设置好，在程序运行期间不能改变。

- 数据寄存器 D 或数据存储器 DM 作为设定值

设定值放在数据寄存器 D 或数据存储器 DM 中，因此设定值在程序运行期间可以动态改变。

数据的内容只有低 14 位有效，忽略最高位和次高位，范围：0~16383。

定时器为减 1 计数型。当定时器的线圈（OUT 指令）由断开（OFF）到接通（ON）的跳变时，把设定值装入定时器的当前值寄存器；此后，每经过一个单位时间，当前值寄存器的内容减 1，当当前值寄存器的内容减到 0 时，定时器的触点被接通（定时器标志位被置 1）。若定时器的线圈（OUT 指令）被断开，则定时器被复位，其当前值寄存器被复位为 0，其触点被断开（定时器标志位被清 0）。

(2) 保持型定时器的动作及编号

保持型定时器使用 TONR 指令来驱动，其定时器的编号也为 T0~T23（100ms 或 10ms），当某一定时器使用 TONR 指令线圈驱动时就为保持型定时器、使用该 OUT 输出线圈驱动时就为非保持型定时器。

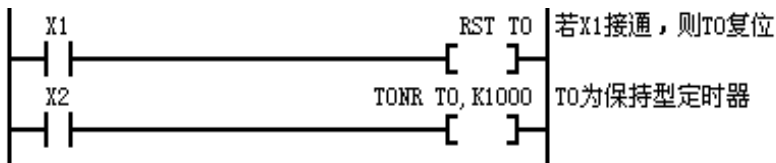
TONR 指令的格式如下：

TONR OUT , IN1 , IN2

操作数	可使用的元件
OUT	T0~T23
IN1	设定值。常数、D、RM、RY、RX、RC、RT、DM
IN2	可选参数。无该项或为 K0，表示时基 100ms；为 K1，表示时基 10ms

动作说明：

若该指令被接通，则指定的定时器从其当前值寄存器的内容开始进行减 1 定时，当减到 0 时使其触点接通；若该指令被断开，则定时器停止定时，其当前值保持不变。当定时器被复位后，将把设定值的内容装入当前值寄存器中（执行该指令时）。



保持型定时器的使用

(3) 触点的动作时序及精度

定时器在其线圈（OUT 指令）被接通时开始定时，定时时间到后，当程序扫描到该定时器的线圈（OUT 指令）时其输出触点接通。

因此，从驱动定时器线圈（OUT 指令）到其触点动作，定时器触点的动作精度大致表示为：

$$T^{+T0}_{-a}$$

其中，T： 定时器设定时间（s）

T0： 扫描周期（s）

a： 定时器时钟周期，10ms、100ms 的定时器对应为 0.01、0.1（s）

如果编程时定时器触点指令写在线圈指令之前，在最坏的情况下，定时器输出触点动作误差为+2T0。

(4) 如何读取定时器的当前值

定时器的当前值寄存器只有低 14 位为实际的当前值，当要读取定时器的当前值时，其当前值寄存器的最高 2 位必须被屏蔽为 0，使用表达式可以很方便地完成这一功能。

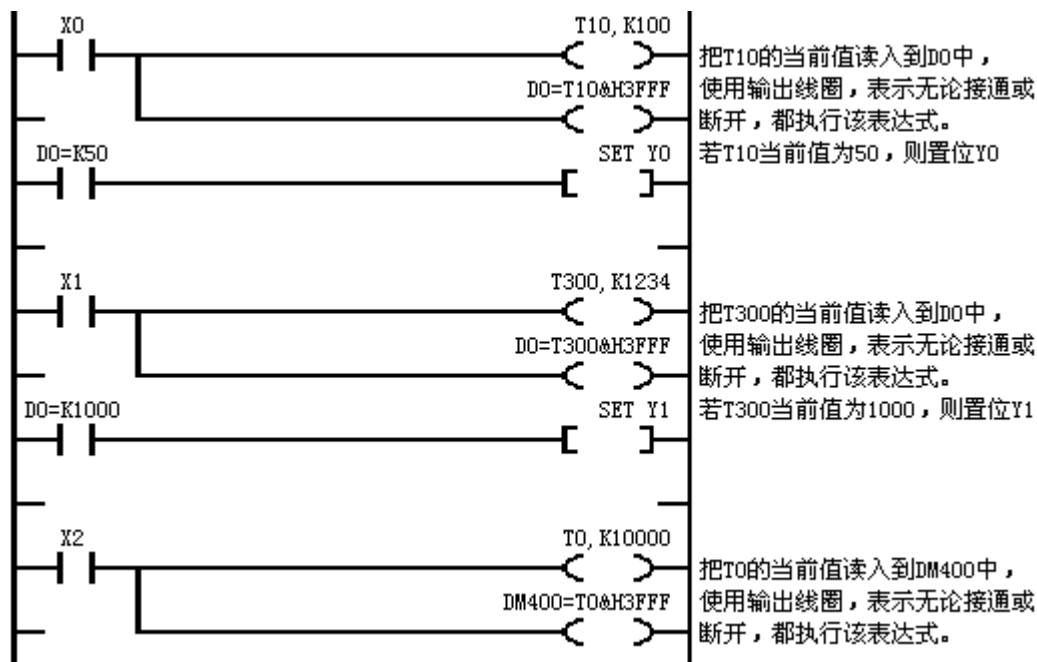
例如：

$D0 = T10 \& H3FFF$ ，即是把定时器 T10 的当前值读入到数据寄存器 D0 中。

$D0 = T300 \& H3FFF$ ，即是把定时器 T300 的当前值读入到数据寄存器 D0 中。

$DM400 = T0 \& H3FFF$ ，即是把定时器 T0 的当前值读入到变量 DM400 中。

在梯形图中的表示如下：



如何读取定时器的当前值

注：定时器 Tx 用在表达式（包含函数调用）中时，总是使用定时器当前值，T0～T47 无法在表达式（包含函数调用）中使用定时器触点，而对于 T256 以后编号的定时器，若要在表达式（包含函数调用）中使用定时器触点，则可使用 Tx.15 来访问 Tx 的触点。

1.3.5 IEC 定时器（TON/TOF/TP）

IEC 定时器为 IEC61131-3 标准所规定的定时器，共有 3 种：TON（接通延时定时器）、TOF（断开延时定时器）、TP（脉冲定时器），这 3 种定时器均应使用指令线圈驱动。

(1) TON（接通延时定时器）

指令的格式 1 如下：

TON Timer, PV, CLK

Timer: T 型，为所使用的内部定时器，可使用对象为 **T384~…**。

PV: INT 型，为定时器的定时时间设定值（范围为 0~16383），可使用对象为常数、D、RM、RY、RX、RC、RT、DM。

CLK: 定时器的时基选择，分别为“1S”、“0.1S”、“0.01S”、“1MS”。

动作说明：当该指令被接通时，把设定值 PV 装入定时器 Timer 的当前值寄存器，开始启动定时，每经过一个单位时间，定时器当前值寄存器的内容减 1，当当前值寄存器的内容减到 0 时，定时器触点变为 ON，同时停止定时；若该指令被断开，则定时器和定时器触点被复位为 0。

例如：TON T500, K1000, 0.1S

指令的格式 2 如下：

TON Timer, PV, Q, Sel

Timer: T 型，为所使用的内部定时器，可使用对象为 **T0~T23, T272~…**。

PV: INT 型，为定时器的定时时间设定值（范围为 0~16383），可使用对象为常数、D、RM、RY、RX、RC、RT、DM。

Q: BOOL 型，定时器的输出位，可使用对象为 **Y、M、Dx.y、DMx.y**。

Sel: 为可选参数，用于选择定时器的时基，分别为“1s”、“0.1s”、“0.01s”、“0.2s”。定时器为 T272~…时，若无该参数，则表示时基为 200ms。定时器为 T0~T23（只有 0.1s 或 0.01s 两种时基）时，若无该参数，则表示时基为 100ms。

当内部定时器用作该格式的 IEC 定时器时，其定时器触点就没有意义了，用户不要再使用。

例如：

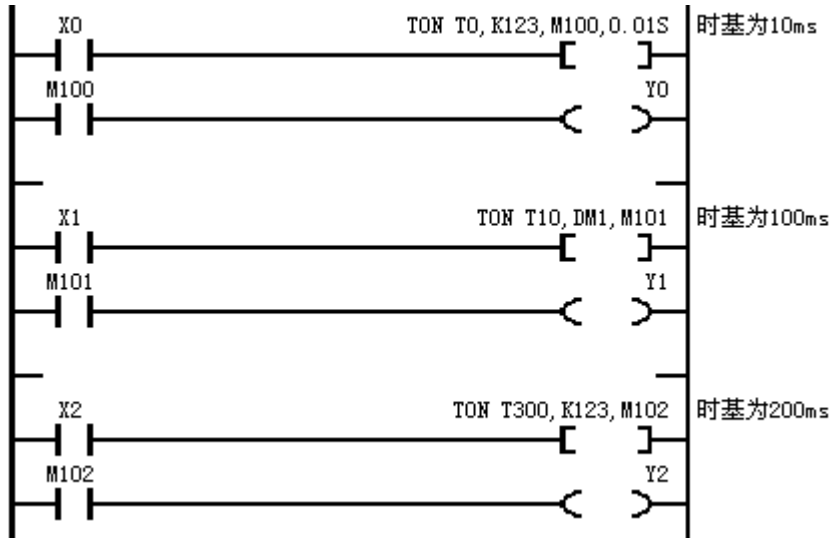
10ms 定时器： **TON T0, K123, M100, 0.01S**

100ms 定时器： **TON T10, DM1, M101 或 TON T400, DM1, M101, 0.1S**

200ms 定时器： **TON T300, K123, M102**

1s 定时器： **TON T310, K123, M103, 1S**

在梯形图中表示如下：



TON 定时器的梯形图表示

动作说明：

当该指令被接通时，把设定值 PV 装入定时器 Timer 的当前值寄存器，开始启动定时，每经过一个单位时间，定时器当前值寄存器的内容减 1，当当前值寄存器的内容（忽略位 15 和位 14）减到 0 时，定时器输出位 Q 变为 ON，同时停止定时；若该指令被断开，则定时器和定时器输出位 Q 被复位为 0。

注：当使用 RST 指令复位 IEC 定时器时（如：RST T0），并不能使 TON 定时器的输出位 Q 复位，必须同时使用“RST Q”指令使 Q 复位。

(2) TOF（断开延时定时器）

指令的格式 1 如下：

TOF Timer, PV, CLK

Timer: T 型，为所使用的内部定时器，可使用对象为 **T384~…**。

PV: INT 型，为定时器的定时时间设定值（范围为 0~16383），可使用对象为**常数、D、RM、RY、RX、RC、RT、DM**。

CLK: 定时器的时基选择，分别为“1S”、“0.1S”、“0.01S”、“1MS”。

动作说明：若该指令被接通，则 Timer 复位，定时器触点为 ON，停止定时；当该指令被断开时，把设定值 PV 装入定时器 Timer 的当前值寄存器，开始启动定时，每经过一个单位时间，定时器当前值寄存器的内容减 1，当当前值寄存器的内容减到 0 时，定时器触点变为 OFF，同时停止定时。

例如：TOF T500, K1000, 0.1S

指令的格式 2 如下：

TOF Timer, PV, Q, Sel

Timer: T 型，为所使用的内部定时器，可使用对象为 **T0~T23, T272~…**。

PV: INT 型，为定时器的定时时间设定值（范围为 0~16383），可使用对象为**常数、D、RM、RY、RX、RC、RT、DM**。

Q: BOOL 型，定时器的输出位，可使用对象为 **Y、M、Dx.y、DMx.y**。

Sel: 为可选参数，用于选择定时器的时基，分别为“1s”、“0.1s”、“0.01s”、“0.2s”。定时器为 T272~…时，若无该参数，则表示时基为 200ms。定时器为 T0~T23（只有 0.1s 或 0.01s 两种时基）时，若无该参数，则表示时基为 100ms。

当内部定时器用作该格式的 IEC 定时器时，其定时器触点就没有意义了，用户不要再使用。

例如：

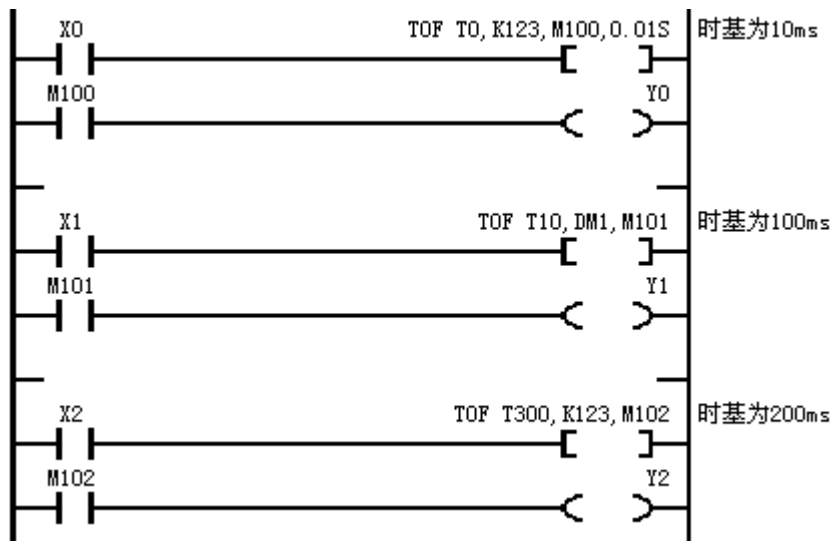
10ms 定时器： **TOF T0, K123, M100, 0.01S**

100ms 定时器： **TOF T10, DM1, M101 或 TOF T400, DM1, M101, 0.1S**

200ms 定时器： **TOF T300, K123, M102**

1s 定时器： **TOF T310, K123, M103, 1S**

在梯形图中表示如下：



TOF 定时器的梯形图表示

动作说明：

若该指令被接通，则 Timer 复位，定时器输出位 Q 为 ON，停止定时；当该指令被断开时，把设定值 PV 装入定时器 Timer 的当前值寄存器，开始启动定时，每经过一个单位时间，定时器当前值寄存器的内容减 1，当当前值寄存器的内容（忽略位 15 和位 14）减到 0 时，定时器输出位 Q 变为 OFF，同时停止定时。

注：当使用 RST 指令复位 IEC 定时器时（如：RST T0），并不能使 TOF 定时器的输出位 Q 复位，必须同时使用“RST Q”指令使 Q 复位。

(3) TP（脉冲定时器）

指令的格式 1 如下：

TP Timer, PV, CLK

Timer: T 型，为所使用的内部定时器，可使用对象为 **T384~...**。

PV: INT 型，为定时器的定时时间设定值（范围为 0~16383），可使用对象为**常数、D、RM、RY、RX、RC、RT、DM**。

CLK: 定时器的时基选择，分别为“1S”、“0.1S”、“0.01S”、“1MS”。

动作说明：当该指令被接通时，把设定值 PV 装入定时器 Timer 的当前值寄存器，定时器触点被接通，开始启动定时，每经过一个单位时间，定时器当前值寄存器的内容减 1，此时该指令的通断状态不会影响定时，当当前值寄存器的内容减到 0 时，定时器触点变为 OFF，同时停止定时。当定时器触点变为 OFF 后，若该指令被断开，则 Timer 被复位。

例如：TP T500, K1000, 0.1S

指令的格式 2 如下：

TP Timer, PV, Q, Sel

Timer: T 型，为所使用的内部定时器，可使用对象为 **T0~T23, T272~...**。

PV: INT 型，为定时器的定时时间设定值（范围为 0~16383），可使用对象为**常数、D、RM、RY、RX、RC、RT、DM**。

Q: BOOL 型，定时器的输出位，可使用对象为 **Y、M、Dx.y、DMx.y**。

Sel: 为可选参数，用于选择定时器的时基，分别为“1s”、“0.1s”、“0.01s”、“0.2s”。定时器为 T272~... 时，若无该参数，则表示时基为 200ms。定时器为 T0~T23（只有 0.1s 或 0.01s 两种时基）时，若无该参数，则表示时基为 100ms。

当内部定时器用作该格式的 IEC 定时器时，其定时器触点就没有意义了，用户不要再使用。

例如：

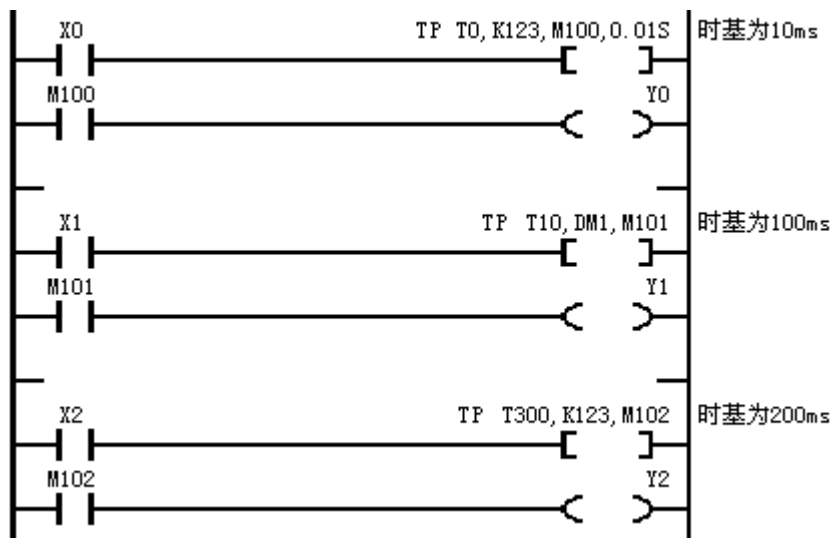
10ms 定时器：TP T0, K123, M100, 0.01S

100ms 定时器：TP T10, DM1, M101 或 TP T400, DM1, M101, 0.1S

200ms 定时器：TP T300, K123, M102

1s 定时器：TP T310, K123, M103, 1S

在梯形图中表示如下：



TP 定时器的梯形图表示

动作说明：

当该指令被接通时，把设定值 PV 装入定时器 Timer 的当前值寄存器，定时器输出位 Q 被接通，开始启动定时，每经过一个单位时间，定时器当前值寄存器的内容减 1，此时该指令的通断状态不会影响定时，当当前值寄存器的内容（忽略位 15 和位 14）减到 0 时，定时器输出位 Q 变为 OFF，同时停止定时。当定时器输出位 Q 变为 OFF 后，若该指令被断开，则 Timer 被复位。

注：当使用 RST 指令复位 IEC 定时器时（如：RST T0），并不能使 TP 定时器的输出位 Q 复位，必须同时使用“RST Q”指令使 Q 复位。

1.3.6 具有外部时钟输入的扩展定时器

TONE 为具有外部时钟输入的扩展接通延时定时器，即定时器的时钟不是由内部固定提供，而是由外部（通过指令输入）提供。

指令的格式如下：

TONE CLK, Timer, PV

CLK: BOOL 型，定时器的时钟输入，可使用对象为 **X、Y、M、T、C、Dx.y、DMx.y**。

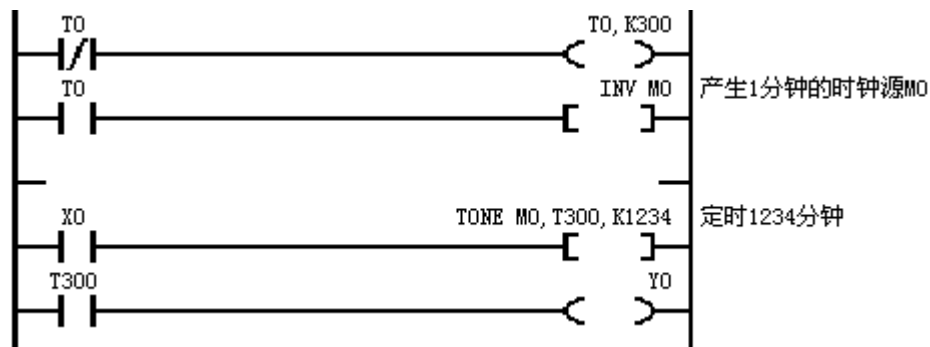
Timer: T_200 型，为所使用的内部定时器，可使用对象为 **T272~...**。

PV: INT 型，为定时器的定时时间设定值，可使用对象为**常数、D、RM、R Y、RX、RC、RT、DM**。设定值范围：1~16383，定时值=设定值×时钟周期。

例如：

TONE CLK1, T300, K1234

在梯形图中表示如下：



TONE 定时器的梯形图表示

动作说明：

当该指令被接通时，把设定值 PV 装入定时器 Timer 的当前值寄存器，开始启动定时，以输入时钟 CLK 的周期为单位进行减计数，当当前值寄存器的内容（忽略位 15 和位 14）减到 0 时，定时器 Timer 标志位变为 ON，同时停止定时；若该指令被断开，则定时器 Timer 被复位。

注：在**主控指令**和**步进指令**中使用该定时器时，若**主控指令**和**步进指令**执行**断开操作**（**强制线圈复位**）则并不能使该定时器复位。

1.3.7 普通计数器 C

普通计数器有两个相关变量：当前值寄存器和计数器标志位。当前值寄存器（最高位被清 0 后）表示计数器所计的脉冲数，计数器标志位为其输出触点。对于每个计数器，这两个变量使用同一名称，但使用场合不同，其所指也不同。

计数器采用十进制编号。

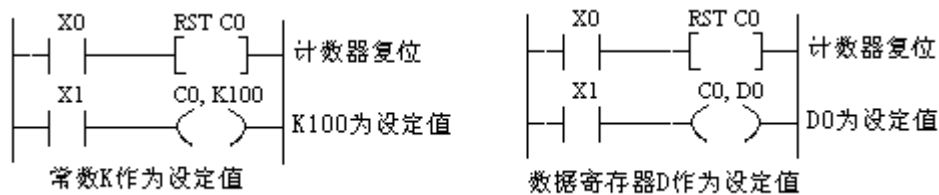
普通计数器是在执行扫描操作时对内部元件（如 X、Y、M、T 等）或其逻辑组合进行计数的计数器。因此，其输入信号频率应大于 PLC 扫描周期。

EasyLad 梯形图语言的普通计数器其编号为：

C0~C23，最大计数值为 32767。

C272~...，最大计数值为 16383，与定时器 T272~...、数据存储器 DM272~... 共用同一编号，即若某一编号（272~...）用作计数器，则该编号不能再用作定时器或数据存储器，反之亦然。

计数器可以用用户程序存储器内的常数 K 作为设定值，也可以用数据寄存器 D 或数据存储器 DM 的内容作为设定值。



作为计数器设定值的几种方法

- 常数 K 作为设定值

设定值只能在用户编制程序时设置好，在程序运行期间不能改变。

- 数据寄存器 D 或数据存储器 DM 作为设定值

设定值放在数据寄存器 D 或数据存储器 DM 中，因此设定值在程序运行期间可以动态改变。

数据的内容只有低 15 位有效，忽略最高位，范围：0~32767。

计数器为减 1 计数型。当程序扫描到计数器的线圈（OUT 指令）时，若计数器还没有装入设定值，则把设定值装入计数器的当前值寄存器（无论该 OUT 指令是否被执行）；此后，每当计数器的线圈（OUT 指令）发生由断开（OFF）到接通（ON）的跳变时，当前值寄存器的内容就减 1；当当前值寄存器的内容减

到 0 时，计数器的触点被接通（计数器标志位被置 1），计数器停止计数，除非计数器被复位。若要计数器复位，必须执行 RST 指令。当计数器的 RST 指令被接通时，对应的计数器被复位，其当前值寄存器被复位为 0，其触点被断开（计数器标志位被清 0）。

● 如何读取普通计数器的当前值

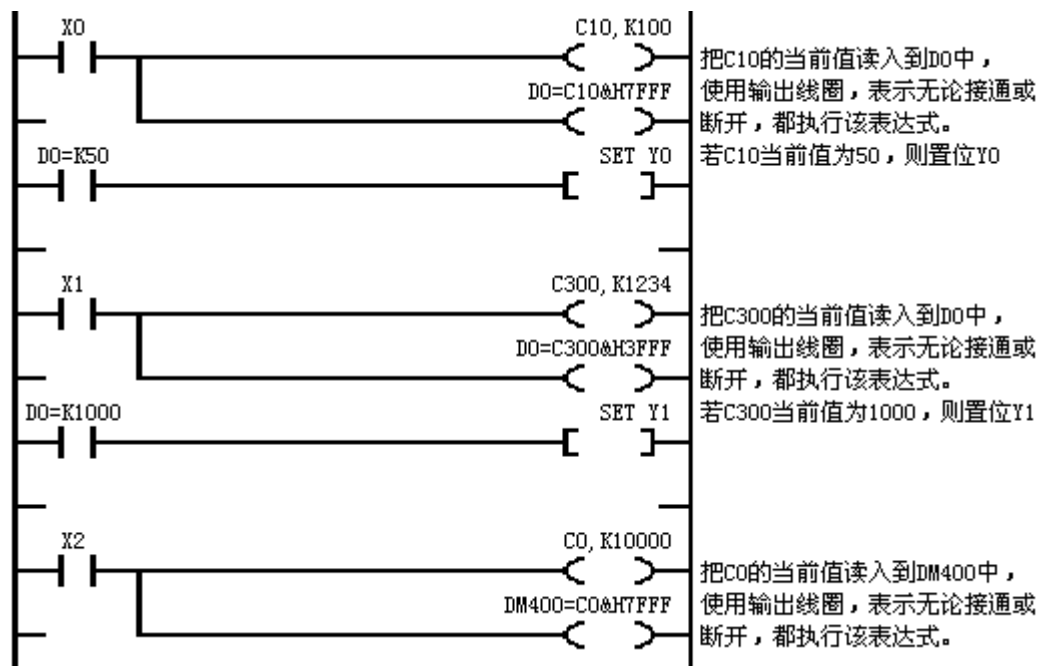
计数器的当前值寄存器只有低 15 位（C0~C23）或低 14 位（C272~…）为实际的当前值，当要读取计数器的当前值时，其当前值寄存器的最高位（C0~C23）或最高 2 位（C272~…）必须被屏蔽为 0，使用表达式可以很方便地完成这一功能。例如：

$D0 = C10 \& H7FFF$ ，即是把计数器 C10 的当前值读入到数据寄存器 D0 中。

$D0 = C300 \& H3FFF$ ，即是把计数器 C300 的当前值读入到数据寄存器 D0 中。

$DM400 = C0 \& H7FFF$ ，即是把计数器 C0 的当前值读入到变量 DM400 中。

在梯形图中的表示如下：



如何读取普通计数器的当前值

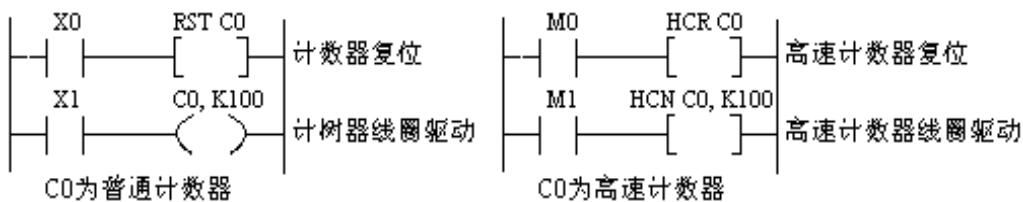
注：计数器 Cx 用在表达式（包含函数调用）中时，总是使用计数器当前值，C0~C23 无法在表达式（包含函数调用）中使用计数器触点，而对于 C256 以后编号的计数器，若要在表达式（包含函数调用）中使用计数器触点，则可使用 $Cx.15$ 来访问 Cx 的触点。

1.3.8 高速计数器

(1) 16 位高速计数器

高速计数器可以对输入端子上的高速脉冲进行计数。

EasyLad 梯形图共有 3 个 16 位高速计数器，它们与普通计数器共享为 C0、C1、C2，其计数脉冲输入端分别为 X0、X1、X2。C0~C2 用作高速计数器时，就不能再用作普通计数器。C0~C2 是用作 16 位高速计数器还是用作普通计数器，是根据驱动其线圈所用的指令来区分：若用 OUT 指令驱动，则为普通计数器；若用 HCN 指令驱动，则为 16 位高速计数器。如下图：



16 位高速计数器和普通计数器的区分

16 位高速计数器的设定值的装载方法同普通计数器的一样。

● 高速计数器的类型及动作

1. C0、C1 的类型及动作

C0、C1 均采用中断方式计数的，它独立于扫描周期，但会增加程序的扫描时间。

C0、C1 为单相高速计数器，其动作如下：

- ① 计数器为减 1 计数型。
- ② 当程序扫描到高速计数器的线圈（HCN 指令）时，若该指令线圈被接通（为 ON），则允许计数器计数，同时若高速计数器还没有装入设定值，则把设定值装入计数器的当前值寄存器。
- ③ 若允许计数器计数，则每当对应的计数脉冲输入端发生由通（ON）到断（OFF）的跳变时，使对应的当前值寄存器的内容减 1，当减到 0 时，对应的输出触点被接通。此后，计数器停止高速计数，除非高速计数器被复位。
- ④ 若要高速计数器复位，必须执行高速计数器复位指令 HCR。当该指令被接通时，使高速计数器复位，其当前值寄存器被复位为 0，其输出触点被断开，高速计数器停止计数。

2. C2 的类型及动作

C2 采用专用的硬件计数器进行计数，因此它独立于扫描周期，并且不占用程序的扫描时间。

若特殊辅助继电器 M195 (HC2M) 为 0 (OFF)，则 C2 为不可自动重装设定值的单相高速计数器；若特殊辅助继电器 M195 (HC2M) 为 1 (ON)，则 C2 为可自动重装设定值的单相高速计数器。

高速计数器 C2 不共用普通计数器 C2 的当前值寄存器 RC2，而是采用专用的当前值寄存器 RHC2，但是共用普通计数器 C2 的触点。高速计数器 C2 复位时，RC2 与 RHC2 一起被复位为 0。

高速计数器 C2 为不可自动重装设定值的单相高速计数器时，当高速计数器 C2 计数时不影响 RC2 的内容，其动作同高速计数器 C0、C1 的单相方式一样。

高速计数器 C2 为可自动重装设定值的单相高速计数器时，RC2 的值决定是否对高速计数器 C2 的溢出次数（减到 0 的次数）进行计数，若 RC2 为 0，则不对溢出次数进行计数；若 RC2 不为 0，则对溢出次数进行计数，其 RC2 的值即为溢出次数的计数值。其动作如下：

- ① 计数器为减 1 计数型。
- ② 当程序扫描到高速计数器的线圈 (HCN 指令) 时，若该指令线圈被接通，则允许计数器计数，同时若高速计数器还没有装入设定值，则把设定值装入计数器的当前值寄存器 RHC2。
- ③ 若允许计数器计数，则每当对应的计数脉冲输入端 (X2) 发生由通到断的跳变时，使当前值寄存器的内容减 1。
- ④ 每当减到 0 时，把设定值的内容重装给高速计数器 C2，若 RC2 为 0，则不接通 C2 的触点，而是接通特殊辅助继电器 M194 (HC2F)，计数器继续处于运行状态；若 RC2 不为 0，则把 RC2 的值减 1，若减 1 后不为 0，则不接通 C2 的触点，而是接通 M194 (HC2F)，计数器继续处于运行状态，若 RC2 减 1 后为 0，则接通 C2 的触点，而不接通 M194 (HC2F)，同时计数器停止计数，除非被复位。

- ⑤ 若高速计数器 C2 被复位（执行 HCR 指令），则 RC2 与 RHC2 一起被复位为 0，计数器停止计数，但不影响 M194（HC2F）。
- ⑥ M186（HC2CK）为高速计数器 C2 计数时钟选择，0 选择外时钟 X2，1 选择内时钟（默认 0.375us，也可由指令“SFRWR 微秒数, HC2USCK”设置内时钟，例如指令 SFRWR 1,HC2USCK 把内时钟设置为 1us，微秒数范围为 1-8）

(2) 32 位高速计数器

32 位高速计数器可以对输入端子上的高速脉冲进行 32 位计数。

EasyLad 梯形图有 3 个 32 位高速计数器 C0、C1、C2。C0 的 32 位当前值寄存器为@RC0（RC0、RC1），预置值寄存器为@RC5（RC5、RC6）。C1 的 32 位当前值寄存器为@RC3（RC3、RC4），预置值寄存器为@RC7（RC7、RC8）。C2 的 32 位当前值寄存器为@RHC2（RHC2、RC2），无预置值。因此，当 C0、C1、C2 用于 32 位高速计数器时，其对应的 C0、C1、C2、C3、C4、C5、C6、C7、C8 就不能再用于 16 位高速计数器或普通计数器。

32 位高速计数器有 4 种计数方式：单相减计数、单相增计数、AB 相增减计数、AB 相倍频增减计数。由指令“DHCN”驱动。如下：

DHCN OUT , IN1

操作数	可使用的元件
OUT	C0、C1、C2
IN1	计数模式设定值。K-1、K0~K4、K8~K10、K16~K20、K24~K26

该指令的动作说明如下：

若该指令被接通，则允许指定的高速计数器计数（IN1 不为-1 时）。若该指令被断开，则不执行任何操作。

IN1 为计数模式。●K-1：暂停计数。●K0：单相减计数。●K1：单相增计数。●K2：AB 相增减计数。●K3：AB 相 2 倍频增减计数（B 相跳变计数）。●K4：AB 相 4 倍频增减计数。●K8：单相 2 倍频减计数。●K9：单相 2 倍频增计数。●K10：AB 相 2 倍频增减计数（A 相跳变计数）。●K16：单相减计数、到预置值自动复位。●K17：单相增计数、到预置值自动复位。●K18：AB 相增减计数、到预置值自动复位。●K19：AB 相 2 倍频增减计数、到预置值自动

复位。●**K20**：AB 相 4 倍频增减计数、到预置值自动复位。●高速计数器 C2 只有 K0（单相减计数）。

当为单相计数时，C0 的计数输入端为 X0，C1 的计数输入端为 X1，C2 的计数输入端为 X2。当为 AB 相计数时，C0 的计数输入端为 X0（A 相）、X3（B 相），C1 的计数输入端为 X1（A 相）、X4（B 相）。A 相和 B 相的相位关系决定了计数器是增计数还是减计数：当 A 相输入端由 ON→OFF 的跳变时，若 B 相为 ON，则执行减计数，若 B 相为 OFF，则执行增计数。

该指令不向计数器装入设定值，用户可直接向计数器的当前值寄存器设定其初值。

高速 C0 计数，当当前值寄存器等于预置值寄存器时：将会触发中断（若允许），若 HSOE0（M197，高速 C0 高速输出控制继电器）为 ON，则执行高速输出并使 HSOE0 复位，若设置为到预置值自动复位则把当前值寄存器复位为 0。

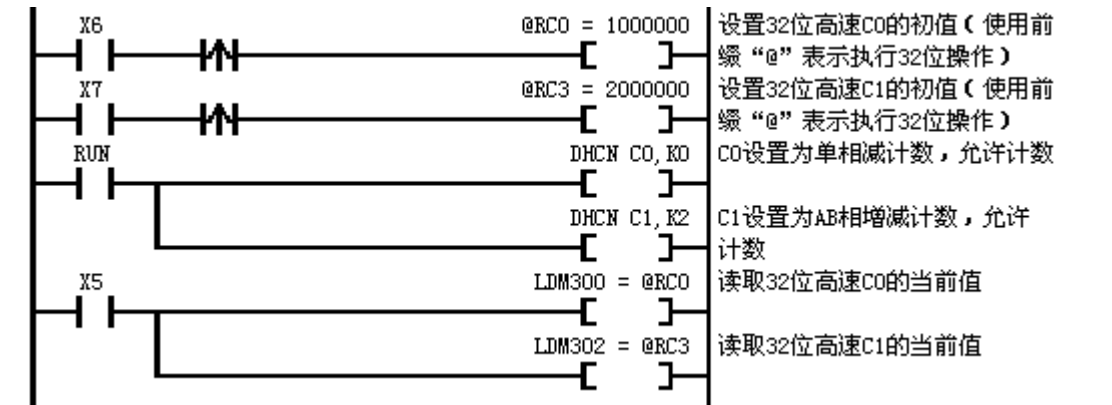
高速 C1 计数，当当前值寄存器等于预置值寄存器时：将会触发中断（若允许），若 HSOE1（M198，高速 C1 高速输出控制继电器）为 ON，则执行高速输出并使 HSOE1 复位，若设置为到预置值自动复位则把当前值寄存器复位为 0。

高速 C2 计数，当当前值寄存器减到 0 或增到 0 时，将会触发中断（若允许）。

C0、C1 的触点分别表示各自的计数方向，OFF：增计数，ON：减计数。

不要使用 HCR 指令（高速计数器复位）或 RST 指令（普通计数器复位）复位 32 位高速计数器。若想使 32 位高速计数器复位，可直接把计数器的当前值寄存器设定为 0。

程序例子如下：



32 位高速计数器程序例子

32 位高速计数和高速输出应用例子：

用高速计数器 C0 和编码器来检测位置，用 Y2、Y1、Y0 来控制电机。

动作如下：

开始时，Y2、Y1、Y0 均断开；

当计数值由 0 到达 1000 时，使 Y0 接通，Y2、Y1 断开；

当计数值由 1000 到达 2000 时，使 Y1 接通，Y2、Y0 断开；

当计数值由 2000 到达 4000 时，使 Y2 接通，Y1、Y0 断开；

则对应的梯形图程序如下：



32 位高速计数和高速输出应用例子

注：没有使用的计数器的当前值寄存器可以作为 16 位的数据存储器来存储数据，即可以存储-32768~32767 之间的数据。计数器的当前值寄存器 RC0~RC9 可进行寄存器位寻址，若没使用，则也可作为内部辅助继电器来使用。

高速计数器的掉电保持：

高速计数器在掉电后，其当前值不保持。若需要高速计数器在掉电后保持其内容，可在编程软件的配置菜单中配置高速计数器当前值寄存器为掉电保持。

1.3.9 IEC 计数器（CTU/CTD）

IEC 计数器为 IEC61131-3 标准所规定的普通计数器，共有 2 种：CTU（增计数器）、CTD（减计数器），这 2 种定时器均应使用指令线圈驱动。

(1) CTU（增计数器）

指令的格式如下：

CTU R, PV, CV, Q

R: BOOL 型，为计数器的复位输入，可使用对象为 **X、Y、M、T、C、Dx.y、DMx.y**。

PV: INT 型，为计数器的设定值，可使用对象为**常数、D、RM、RY、RX、RC、RT、DM**。

CV: INT 型，为计数器的当前值存储，可使用对象为 **DM272~…**。

Q: BOOL 型，计数器的输出位，可使用对象为 **Y、M、Dx.y、DMx.y**。

例如：

CTU X1, K123, DM300, M100

在梯形图中表示如下：



CTU 计数器的梯形图表示

动作说明：

对该指令的通断状态（上升沿）进行增计数，当计数器的当前值 CV（忽略最高位）大于等于预设值 PV 时，计数器的输出位 Q 接通，同时停止计数。当复位输入 R 为 ON 时，计数器复位（CV 和 Q 复位为 0）。

CTU 计数器的当前值 CV 只有低 15 位为实际的当前值，当要读取 CTU 的当前值时，其当前值 CV 的最高位必须被屏蔽为 0，其当前值的读取方法与前面的内部普通计数器的相同。

(2) CTD (减计数器)

指令的格式如下：

CTD LD, PV, CV, Q

LD: BOOL 型，为计数器的装载输入，可使用对象为 **X、Y、M、T、C、D x.y、DMx.y**。

PV: INT 型，为计数器的设定值，可使用对象为**常数、D、RM、RY、RX、RC、RT、DM**。

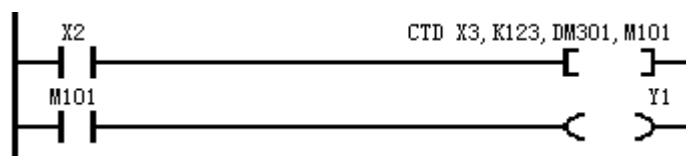
CV: INT 型，为计数器的当前值存储，可使用对象为 **DM272~…**。

Q: BOOL 型，计数器的输出位，可使用对象为 **Y、M、Dx.y、DMx.y**。

例如：

CTD X3, K123, DM301, M101

在梯形图中表示如下：



CTD 计数器的梯形图表示

动作说明：

对该指令的通断状态（上升沿）进行减计数，当计数器的当前值 CV（忽略最高位）等于 0 时，计数器的输出位 Q 接通，同时停止计数。当装载输入 LD 为 ON 时，计数器复位并把预设值 PV 装入当前值 CV。

CTD 计数器的当前值 CV 只有低 15 位为实际的当前值，当要读取 CTD 的当前值时，其当前值 CV 的最高位必须被屏蔽为 0，其当前值的读取方法与前面的内部普通计数器的相同。

1.3.10 映像寄存器 R

输入继电器 X、输出继电器 Y、辅助继电器 M、定时器当前值寄存器、计数器当前值寄存器这些元件是按字存储在内存中的。用户可使用映像寄存器 R 来访问这些内存。映像寄存器 R 即是对这些内存按字来进行编号的。

映像寄存器 R 的编号分绝对地址方式和相对地址方式。绝对地址方式是对这些内存统一进行编号的。相对地址方式是把这些内存分为 X 区、Y 区、M 区、T 区、C 区等不同的区域，相对于各个区域来分别进行编号的。如 R0、R11、R19 是按绝对地址来编号的，RM0、RY1、RC3 是按相对地址来编号的。

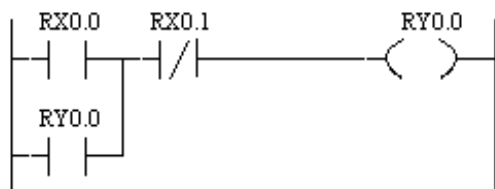
映像寄存器采用十进制编号。

采用绝对地址方式的 R 寄存器不能进行变址寻址，并且只能作为 MOV 指令的操作数。

映像寄存器中 X 区、Y 区、M 区、C 区的 RC0~RC9 的内存可进行寄存器位寻址。

其格式为：“区域标识符” + “编号.位号”

“区域标识符”可为 RX、RY、RM、RC，“编号”为映像寄存器的相对地址编号，“位号”为要访问的位在寄存器中的位置（0~15）。编号和位号之间用点号“.”隔开。如 RX0.5、RM9.15。



按寄存器位寻址方式访问

映像寄存器编号对应表

相对地址		绝对地址	对应的元件															
RHC2		R111	高速计数器 C2 的当前值寄存器															
T 区	RT23	R66	定时器 T23 的当前值寄存器															
	RT22	R65	定时器 T22 的当前值寄存器															
															
	RT1	R44	定时器 T1 的当前值寄存器															
	RT0	R43	定时器 T0 的当前值寄存器															
C 区	RC23	R42	计数器 C23 的当前值寄存器															
	RC22	R41	计数器 C22 的当前值寄存器															
															
	RC1	R20	计数器 C1 的当前值寄存器															
	RC0	R19	计数器 C0 的当前值寄存器															
X 区	RX2	R18	X57	X56	X55	X54	X53	X52	X51	X50	X47	X46	X45	X44	X43	X42	X41	X40
	RX1	R17	X37	X36	X35	X34	X33	X32	X31	X30	X27	X26	X25	X24	X23	X22	X21	X20
	RX0	R16	X17	X16	X15	X14	X13	X12	X11	X10	X7	X6	X5	X4	X3	X2	X1	X0
Y 区	RY2	R15	Y57	Y56	Y55	Y54	Y53	Y52	Y51	Y50	Y47	Y46	Y45	Y44	Y43	Y42	Y41	Y40
	RY1	R14	Y37	Y36	Y35	Y34	Y33	Y32	Y31	Y30	Y27	Y26	Y25	Y24	Y23	Y22	Y21	Y20
	RY0	R13	Y17	Y16	Y15	Y14	Y13	Y12	Y11	Y10	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
M 区	RM12	R12	M207	M206	M205	M204	M203	M202	M201	M200	M199	M198	M197	M196	M195	M194	M193	M192
	RM11	R11	M191	M190	M189	M188	M187	M186	M185	M184	M183	M182	M181	M180	M179	M178	M177	M176
															
	RM1	R1	M31	M30	M29	M28	M27	M26	M25	M24	M23	M22	M21	M20	M19	M18	M17	M16
	RM0	R0	M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
寄存器中的位			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			<div> <div>高位</div> <div>←</div> <div>低位</div> </div>															

1.3.11 临时数据寄存器 D

临时数据寄存器采用十进制编号。

数据寄存器是存储数据并能对数据进行处理如加、减、乘、除等的软元件。

每一个数据寄存器都是为 16 位的二进制补码数（最高位为符号位）；可以用二个数据寄存器合并起来存放 32 位数据（最高位为符号位），如用于乘、除操作时。

- 临时数据寄存器（D0~D15：16 点）

临时数据寄存器用作局部变量来存放临时的数据，如向子程序或函数传递参数、用作变址寄存器、作为逻辑或数据运算的中间变量或数据缓存等，而不要用来存放长期有效（超过 1 个扫描周期）的数据。

临时数据寄存器在所有字元件中访问速度是最高的，所需的程序步数也是最少的。

D0~D15 可进行寄存器位寻址，可以与 X、Y、M 等一样按位来参与各种逻辑操作。

寄存器位寻址的格式为：“D” + “编号.位号”

其中，“编号”为数据寄存器 D 的编号，“位号”为要访问的位在寄存器中的位置（0~15）。编号和位号之间用点号“.”隔开。如 D0.7。

- 特殊数据寄存器（D16：1 点）

D16 为设置用户 ID 时使用。

1.3.12 数据存储 DM/LDM/FDM

数据存储采用十进制编号。

数据存储分为非易失性存储器和易失性存储器。非易失性存储器在断电后数据仍能保持不变，通常用来存断电后仍需保持储在不变的一些数据。易失性存储器在断电后数据不保持，但读写次数不限，并且能参与加、减、乘、除等数据运算，因此通常用作整型（字）变量（范围：-32768~32767）、长整型（双字）变量（范围：-2147483648~2147483647）或浮点型（实型）变量。

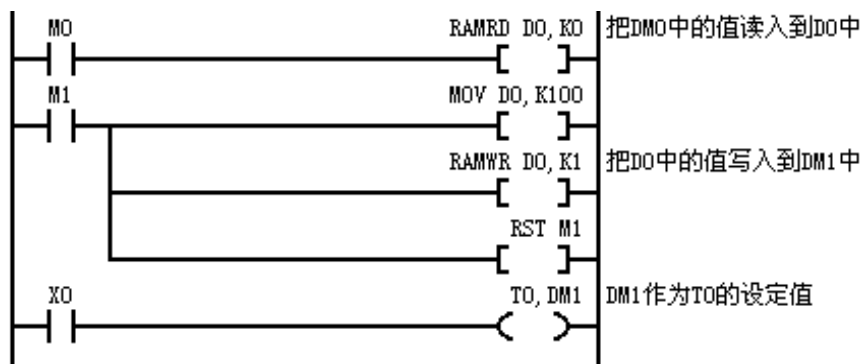
(1) 非易失性存储器

YF0A: DM0~DM255。256 字

YF0H: DM0~DM255, K16640~K32767。16384 字。

非易失性存储器是按 16 位为单元来存放数据的，因此只能以整型（字）为单位来访问。访问非易失性存储器只能使用 **RAMRD**（存储器读）和 **RAMWR**（存储器写）这两条指令（注：DM0~DM255 非易失性存储器可直接作为定时器或计数器的设定值）。

非易失性存储器是有写次数限制的（最少 100 万次），并且写入时间较长（5 ms），使用时应注意这些特性。例如：每写入一页（最多 16 字）数据，将会增加 5ms 的程序扫描时间，并且在写入过程中不会响应中断。每 100ms 向某一单元写入一个数据，那么该单元最短只能使用 28 小时，若每 1 小时写一个数据，则该单元最短可使用 114 年。因此，请注意只有在必要时才进行写入操作，原则上，当特殊事件发生时，才执行写操作，而这种事件又是不很频繁发生的。



非易失性存储器的读写

(2) 易失性整型存储器

CPU	临时用	通用
YF0H	16 个: DM256~DM271	1708 个: DM272~DM1979
YF0H4K	128 个: DM256~DM383	3966 个: DM384~DM4349

易失性整型（字）存储器是按 16 位（整型）为单元来存放数据的。采用二进制补码表示方式，其最高位为符号位，该位为 0 表示正数，为 1 表示负数，所能表示数的范围为-32768~32767。前缀“DM”为整型数据存储器标识符。

易失性整型（字）存储器 DM 可进行存储器位寻址，可以与 X、Y、M 等一样按位来参与各种逻辑操作。

存储器位寻址的格式为：“DM” + “编号.位号”

其中，“编号”为易失性存储器 DM 的编号，可以是变址寻址，“位号”为要访问的位在寄存器中的位置（0~15）。编号和位号之间用点号“.”隔开。如 DM 300.7、DM256[10].9、DM300[D0].10 等。

(3) 易失性长整型存储器

CPU	临时用	通用
YF0H	8 个: LDM256~LDM270	854 个: LDM272~LDM1978
YF0H4K	64 个: LDM256~LDM382	1983 个: LDM384~LDM4348

采用两个连续的易失性整型（字）存储器 DM 存储长整型（双字）变量，若要编号小的单元存放高 16 位、编号大的单元存放低 16 位，应使用长整型标识符 **LDM+小编号** 来访问，例如：LDM256 由 DM256（高 16 位）、DM257（低 16 位）组成，LDM258 由 DM258（高 16 位）、DM259（低 16 位）组成。若要编号小的单元存放低 16 位、编号大的单元存放高 16 位，应使用 **!LDM+小编号** 来访问，例如：!LDM300 由 DM300（低 16 位）、DM301（高 16 位）组成。

(4) 易失性浮点型存储器

CPU	临时用	通用
YF0H	8 个: FDM256~FDM270	854 个: FDM272~FDM1978
YF0H4K	64 个: FDM256~FDM382	1983 个: FDM384~FDM4348

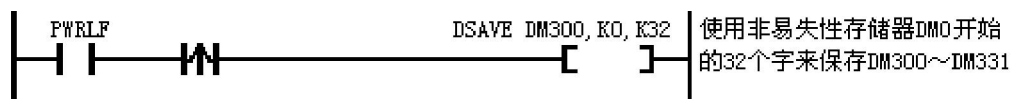
浮点格式：

YF0H 系列浮点数采用标准的 IEEE-754 单精度浮点格式，占用 2 个字。使用

浮点型标识符 **FDM+编号**小的单元的编号来访问，例如：FDM256 由 DM256、DM257 组成，其中 DM256 的低字节存尾数低 8 位，高字节存尾数中 8 位；DM257 的低字节存尾数高 7 位和阶码最低位，高字节存阶码高 7 位和数符。

(5) 易失性存储器的掉电保持

易失性存储器掉电后数据都是不保持的，若希望掉电后数据保持，可使用掉电检测功能，在掉电时把要保持的易失性存储器保存到内部的非易失性存储器中即可。其梯形图例子（PWRLF 为掉电检测输入，DM300~DM331 掉电保持）：



易失性存储器的掉电保持程序例子

推荐使用梯形图编程环境 EasyLad 中的配置菜单来配置易失性存储器等的掉电保持。

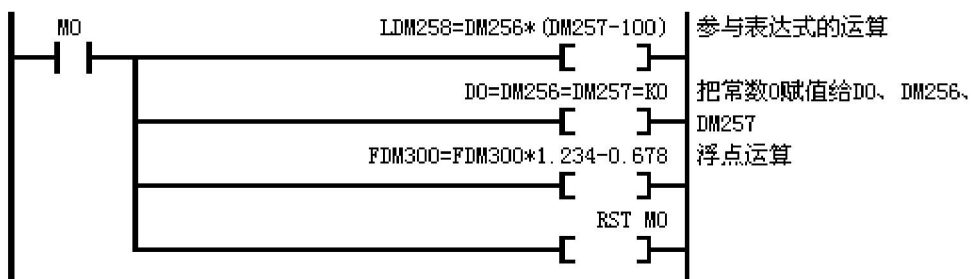
易失性存储器 **DM/LDM/FDM** 作为整型/长整型/浮点型变量，可以用在表达式中来参与各种运算及进行赋值等操作，也可以用在函数的定义指令中作为函数的参数传递元件等。例如：

若要使 DM300 设置为 1000，则可使用“=”赋值表达式 $DM300=1000$ 即可。

若要把 LDM300 的内容送给 LDM301，则可使用表达式 $LDM301=LDM300$ 即可。

若要把 DM300 的内容加上 DM301 的内容，其和送给 DM302，则可使用表达式 $DM302=DM300+DM301$ 即可。

在梯形图中的例子如下：



数据存储 DM/LDM/FDM 在表达式中的使用

注：DM256~DM383、LDM256~LDM382、FDM256~FDM382 为临时数据存储，

推荐用作局部变量来存放临时的数据，如向子程序或函数传递参数、作为数据运算的中间变量或数据缓存等，而不要用来存放长期有效（超过 1 个扫描周期）的数据。DM384～…、LDM384～…、FDM384～…则用来存放长期有效的数据。

1.3.13 变址寻址

所谓寻址方式就是寻找确定参与操作的元件（变量）的真正地址。

EasyLad 梯形图语言共有 2 种寻址方式：直接寻址和变址寻址。

直接寻址就是直接使用操作元件的地址（编号）。如：M0，X1，Y3，D2 等。

变址寻址就是采用基址+偏移量来作为操作元件的地址。其表达格式为：**基址[偏移量]**。根据偏移量的不同，变址寻址又分两种形式：偏移量为常数的变址寻址和偏移量为数据寄存器 D0、D1 等的内容的变址寻址。

1、 偏移量为常数的变址寻址

可寻址的元件为：X、Y、M、T、C、D、RM、RY、RX、RT、RC、DM、LDM、FDM、K。例如：

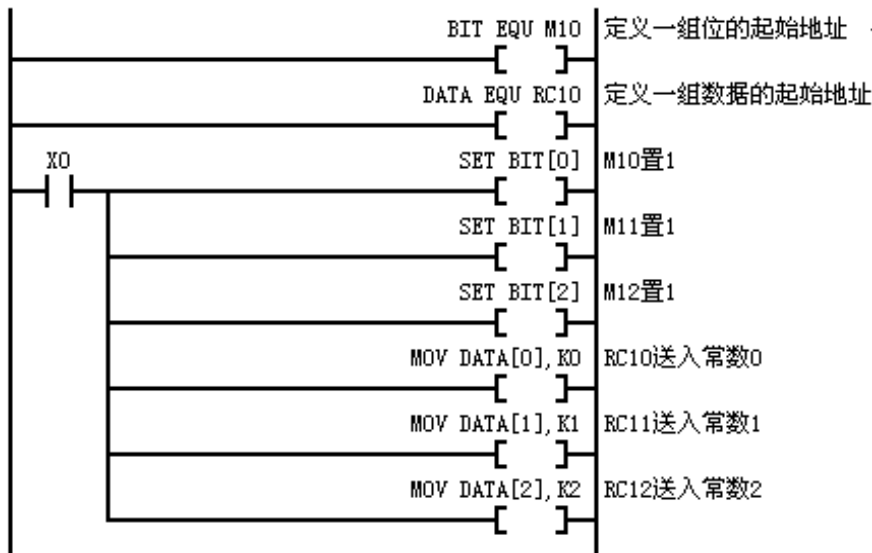
X2[3] 则实际的操作元件为 X5。

M10[5] 则实际的操作元件为 M15。

C5[0] 则实际的操作元件为 C5。

DM500[4] 则实际的操作元件为 DM504。

K10[2] 则实际的操作元件为 K12。



偏移量为常数的变址寻址的应用

2、 偏移量为数据寄存器 D0、D1 的内容的变址寻址

D0 可寻址的元件为：D、DM、LDM、FDM、T272~…、C272~…。

D1 可寻址的元件为：DM、LDM、FDM、T272~…、C272~…。

注：@Dx[D0]为错误的变址寻址。

例如：

若 $D0 = 2$ 则

$D2[D0]$ 的实际操作元件为 $D4$ 。

$RT8[D0]$ 的实际操作元件为 $RT10$ 。

$DM256[D0]$ 的实际操作元件为 $DM258$ 。

若 $D0 = 4$ 则

$D2[D0]$ 的实际操作元件为 $D6$ 。

$RT8[D0]$ 的实际操作元件为 $RT12$ 。

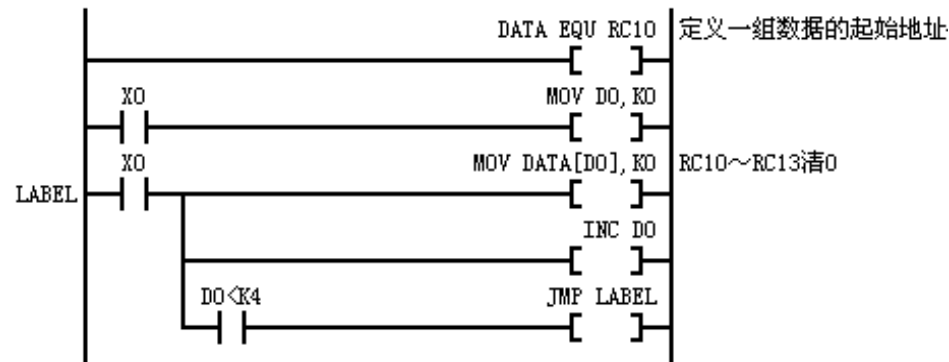
$DM256[D0]$ 的实际操作元件为 $DM260$ 。

若 $D1=300$ 则

$DM0[D1]$ 的实际操作元件为 $DM300$ 。

$DM1[D1]$ 的实际操作元件为 $DM301$ 。

$DM2[D1]$ 的实际操作元件为 $DM302$ 。



偏移量为数据寄存器 $D0$ 的内容的变址寻址的应用

3、偏移量为 $D2 \sim D15$ 、 $DM256 \sim DM511$ 的内容的变址寻址

可寻址的元件为： DM 、 LDM 、 FDM 、 $T272 \sim \dots$ 、 $C272 \sim \dots$ 。

该类型的变址寻址应用于指令或触点时，一个指令或触点只能用同一个偏移量元件（ $D0$ 、 $D1$ 除外），而应用于表达式或函数调用时则没限制。例如：

$ADD \quad DM5[DM300], DM10[DM300]$ 是正确的，

$ADD \quad DM500[DM300], DM600[D0]$ 是正确的，

$ADD \quad DM500, DM600[DM300]$ 是正确的，

$ADD \quad DM5[DM300], DM10[DM400]$ 是错误的，

$DM5[DM300] = DM10[DM400] + DM800[DM500]$ 是正确的。

4、多级变址寻址

多级变址寻址格式为：**基址[偏移量 1][偏移量 2]…[偏移量 n]**。其中只有最后一个偏移量才能为变址元件（也可为常数），其他偏移量必须为常数（若为变量则取变量的地址作为常数偏移量）。最终地址为基址+偏移量 1+偏移量 2+ … +偏移量 n

对于多级变址寻址形式 **a.b[c.a][d.c][e.d]…**，其中 a、b、c、d、e 均为符号名，可以简化为：**a.b[c.][d.][e.]…**。

多级变址寻址例子：

ADD DM5[7][DM300], DM10[DM300]

ADD DM10[GP][#DM300], DM600[D0] 其中 GP 为定义为常数的符号名

LDM8[IN1][#DM600] = 12345678 其中 IN1 为定义为常数的符号名

注 1：字.位[偏移量]变址格式自动转换为字[偏移量].位变址格式

注 2：对于“基址[#符号名]”这种类型的变址寻址（取符号名的地址为变址偏移量），可以简化为“基址@符号名”，例如：符号名 1[#符号名 2] 可以简化为符号名 1@符号名 2，符号名 1[#符号名 2].5 可以简化为符号名 1@符号名 2.5，符号名 1[#符号名 2][D0] 可以简化为符号名 1@符号名 2[D0]。

1.3.14 位元件间接寻址

可使用数据元件 D0~D15、LDM256~LDM510 来对位元件（M、Y、X、DMx.y）进行间接寻址访问，即把位元件的地址放在上述数据元件中，在操作数中使用位元件间接寻址格式来操作位元件，位元件间接寻址格式如下：

Dx#偏移

LDMx#偏移

Dx 为 D0~D15，**LDMx** 为 LDM256~LDM510，其中存的为位元件的地址。

偏移为十进制常数，也可为某个位元件或定义为位元件的符号名（M、Y、X、DMx.y），此时把该元件的编号作为偏移，DMx.y 的编号为 $16*x+y$ （D 寄存器存地址时 $x < 4096$ 、LDM 存储器存地址时 x 无限制）。

实际的寻址地址为 Dx 的值加上偏移。

例如：D1 的值为 10，则

D1#0 寻址的为 M10

D1#5 寻址的为 M15

D1#M20 寻址的为 M30

D1#DM300.1 寻址的为 DM300.11

位元件（M、Y、X、DMx.y）的地址分配如下：

M 元件的地址为其编号。

Y 元件的地址为其编号+208，注意其编号为八进制，要转换为十进制数值。

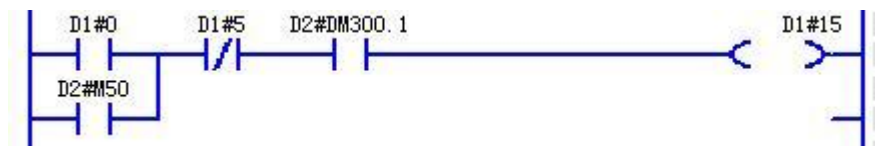
X 元件的地址为其编号+256，注意其编号为八进制，要转换为十进制数值。

DMx.y 元件的地址为其编号（ $16*x+y$ ）。

获取某个位元件编号可使用“#位元件”格式，例如：

D1 = #DM500.3 （把 DM500.3 的编号 8003 送给 D1）

位元件间接寻址在梯形图中的例子如下：

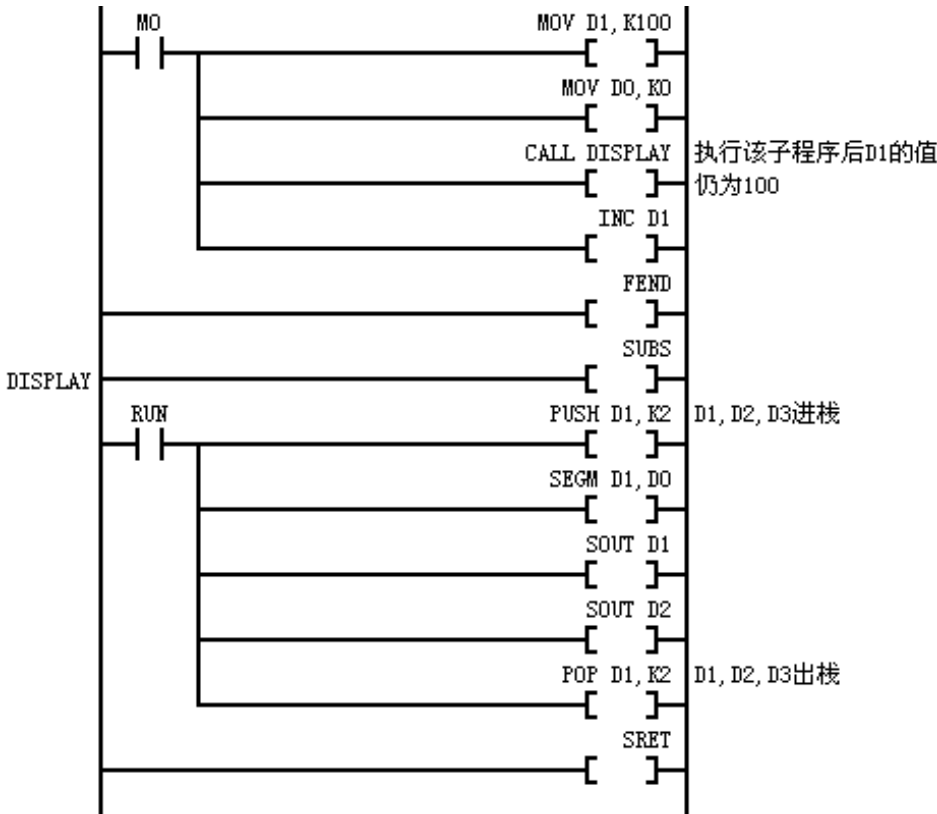


注：位元件间接寻址的输出线圈不要用在步进指令和主控指令中

1.3.15 数据堆栈空间

EasyLad 梯形图语言提供了 256 字节的数据堆栈空间，系统按照后进先出的原则来管理这些堆栈空间。表达式的运算、函数的参数的传递、为函数分配局部变量空间等均要使用数据堆栈空间。用户也可使用这些数据堆栈空间来完成现场数据的保护，如在中断程序或子程序中，为防止破坏被中断处或子程序调用处的程序中的某些数据变量，可把这些数据变量压入数据堆栈空间中保护起来，待中断程序或子程序执行完成后再从数据堆栈空间中恢复这些数据。

用户可使用两条指令来访问这些数据堆栈空间：PUSH（进栈），POP（出栈）。每个字元件进栈将占用 2 个字节的堆栈空间。注意进栈与出栈的数据字节数应一致，否则将可能引起数据堆栈溢出，EasyLad 梯形图语言具有数据堆栈溢出判断能力，当数据堆栈溢出时将使错误指示灯（ERROR）发亮。



数据堆栈空间的使用

1.3.16 中断及中断源

中断即是当某些事件发生时，中断正在运行的程序，转去执行这些事件的中断服务程序，当中断服务程序执行完后再返回到被中断的程序处继续运行。

EasyLad 梯形图具有 9 个中断源：定时中断，高速计数器 C0 中断，高速计数器 C1 中断，高速计数器 C2 中断，外部输入 X3 边缘触发中断，外部输入 X4 边缘触发中断，PTO 脉冲串输出完成中断，外部输入 X5 中断，外部输入 X6 中断。其外部输入边缘触发中断可选择为正边缘触发或负边缘触发。

使用“INT”指令可以定义一个中断源的中断服务程序。使用“EI”指令可以允许某个中断源中断。使用“DI”指令可以禁止某个中断源中断。具体使用方法可参见“中断处理指令”的说明。

当相应的中断源的中断被允许时，系统才查询并记录这些被允许中断的中断源是否有事件发生，若有事件发生，系统将准备执行对应的中断服务程序，当中断服务程序开始执行后，系统将自动清除本次中断事件。系统并不查询那些被禁止中断的中断源，可能这些中断源也有事件发生，但系统不记录这些事件，这些事件被丢失。

EasyLad 梯形图的中断只有一个优先级，即当一个中断服务程序正在运行时，不会被另一个中断所打断，直到这个中断服务程序运行完后，才会响应其他的中断。

1、定时中断

定时中断可使用户以设定的时间间隔（1ms~255ms）来运行某些程序。当定时中断被允许时，每隔设定的时间间隔将执行一次定时中断服务程序。

2、高速计数器 C0、C1、C2 中断

当高速计数器的当前值减到 0（C2）或到预置值（C0、C1）时，若其中断被允许，将产生中断事件，去执行相对应的中断服务程序。

3、外部输入 X3、X4 边缘触发中断

当外部输入 X3、X4 发生正跳变或负跳变时，若对应的中断被允许，将产生边缘触发中断事件。具体中断事件是由正跳变还是负跳变产生，则由特殊辅助继电器 INT4M（对应 X3）、INT5M（对应 X4）控制：若为 OFF，则由正跳变产生；若为 ON，则由负跳变产生。

当改变 INT4M 或 INT5M 的值后，必须执行 EI（开中断）指令才会使新的设置有效。

4、PTO 脉冲串输出完成中断

若 Y0 或 Y1 工作于脉冲串输出（PTO）方式，当设定个数的脉冲串输出完成时，若其中断被允许，将产生该中断事件，同时对应的脉冲串输出完成标记（PTO0F 或 PTO1F）被置 1。Y0 和 Y1 共用该中断，具体是由 Y0 还是 Y1 产生的中断，可通过判断 PTO0F 和 PTO1F 来完成。

5、外部输入 X5、X6 中断

外部输入 X5、X6 可选正跳变或负跳变中断或高速计数器模式下归零中断，若对应的中断被允许，将产生中断事件。

6、中断的响应时间

当中断事件发生时，假设没有其他的中断服务程序正在执行，则中断的响应时间取决于正在执行的那条指令的执行时间，系统必须等到该条指令执行完毕后才响应中断，因此，假设没有其他的中断服务程序正在执行，则 YF0A 中断的响应时间为 $(40\mu\text{s} \sim 250\mu\text{s}) + \text{单条指令执行时间}$ ，YF0H 中断的响应时间为 $(1.5\mu\text{s} \sim 2\mu\text{s}) + \text{单条指令执行时间}$ 。

1.3.17 数据（字符串）表

EasyLad 梯形图语言允许用户把一些数据或字符串组织到一起，形成数据（字符串）表，然后可使用索引和查表函数来访问这些表中的数据或字符串。

1、数据表

数据表中的数据只能为整型数据（-32768~32767），不能超出该范围或为浮点型数据。其表形式如下：

数据（字符串）表

表名

- MainMenu
- Screen0
- SetDpTab
- SetMaxVal
- SetMinVal
- SetStr

描述

设置参数的上限值表，该表必须与参数设置字符串表按顺序和数量一一对应（表示各个参数的上限值）

新建 删除 改名

表类型

数据

表名查找

确定

表内容

	0	1	2	3	4	5	6	7	8	9
0	255	7500	7000	7500	1000	999	1000	1000	1000	1000
10	800	500	100	300	600	200				

插入 删除 确定 取消

数据表的形式

使用查表函数 **\$表名(Index)** 即可访问表中的各个数据。其中 Index 为表中数据的索引，可以是常数、变量或表达式，但必须为整型数，例如当为 0 时表示访问表中的第 0 个数，当为 10 时表示访问表中的第 10 个数。

每个数据表均有一个长度（Len）属性，表示该表中存储了多少个数据。用户可使用 **\$表名.Len(0)** 来访问该长度属性的值。

例如在上表中：

D0 = \$SetMaxVal(0) 则 D0 的值即为表 SetMaxVal 中第 0 个数据“255”。

D1 = \$SetMaxVal(12) 则 D1 的值即为表 SetMaxVal 中第 12 个数据“100”。

D2 = \$SetMaxVal.Len(0) 则 D2 的值即为表 SetMaxVal 的长度 16。

2、字符串表

字符串表是把一些字符串组织在一起。每个字符串都可使用一个索引来进行访问，因而能更有效的来访问这些字符串。其表形式如下：

序号	字符串	描述
0	通讯地址	保存于DM0中
1	上限温度	保存于DM1中
2	下限温度	保存于DM2中
3	目标温度	保存于DM3中
4	输出上限	保存于DM4中
5	输出下限	保存于DM5中
6	启动频率	保存于DM6中
7	比例设置	保存于DM7中
8	积分时间	保存于DM8中
9	微分时间	保存于DM9中

字符串表的形式

使用查表函数 **\$表名(Index)** 即可访问表中的各个字符串。其中 Index 为表中字符串的索引，可以是常数、变量或表达式，但必须为整型数，例如当为 0 时表示访问表中的第 0 个字符串，当为 10 时表示访问表中的第 10 个字符串。（注：该函数返回的为所索引的字符串的存储首地址）

每个字符串表均有一个长度（Len）属性，表示该表中存储了多少个字符串。用户可使用 **\$表名.Len(0)** 来访问该长度属性的值。

例如在上表中：

D2 = \$SetStr.Len(0) 则 D2 的值即为表 SetStr 的长度 10。

DisStr(\$SetStr(5), H00) 显示表 SetStr 中的第 5 个字符串“输出下限”。

1.3.18 字元件的双字组合 (@/!)

@符号的作用是把某 16 位（字）元件（高 16 位）与下一编号的元件（低 16 位）一起组合为 32 位（长整型）的操作数。其格式为：

@字元件

例如

@D0 表示由 D0（高 16 位）、D1（低 16 位）组合成的 32 位操作数。

@RC10 表示由 RC10（高 16 位）、RC11（低 16 位）组合成的 32 位操作数。

@DM300 = @D2 + 10000

@D4 = 1234567

!符号的作用是把某 DM 元件（低 16 位）与下一编号的 DM 元件（高 16 位）一起组合为 32 位（长整型）的操作数。其格式为：

!DM 元件 或者 !符号名

若要组合为浮点型的操作数，其格式为：

F!DM 元件 或者 F!符号名

其中符号名定义为 DM 元件。

例如

!DM300 表示由 DM300（低 16 位）、DM301（高 16 位）组合成的 32 位操作数。

!DM300 = !DM400 + 10000

!DM500 = 12345678

F!VAR1=123.45 ;VAR1 为定义为 DM 元件的符号名

1.3.19 多功能高速输入 X5、X6

X5、X6 的高速输入可用作外部中断、16 位高速计数、测频率、跳变捕获时间值、复位 32 位高速计数器 C0-C1 等功能。

1、设置高速输入工作模式

使用指令“SFRWR 工作模式, HX5M”设置 X5 的高速输入工作模式。

使用指令“SFRWR 工作模式, HX6M”设置 X6 的高速输入工作模式。

指令中工作模式参数可为常数、D 寄存器、DM 存储器，对应的数值如下：

0：禁止高速输入功能，**2**：上升沿中断，**3**：下降沿中断，**4**：增计数，**5**：减计数，**H12**：上升沿中断并复位 32 位高速计数器（X5 复位 C0、X6 复位 C1），**H13**：下降沿中断并复位 32 位高速计数器（X5 复位 C0、X6 复位 C1），**H14**：倍频增计数，**H15**：倍频减计数。

必须使用 EI 指令允许 X5 或 X6 中断源才能响应 X5 或 X6 中断事件。

2、16 位高速计数

当 X5、X6 工作于 16 位高速计数模式时：

使用指令“SFRWR VAL, HX5CNT”设置 X5 的高速计数当前值。

使用指令“SFRWR VAL, HX6CNT”设置 X6 的高速计数当前值。

指令中 VAL 参数为要设置的高速计数当前值，可为常数、D 寄存器、DM 存储器。

使用指令“SFRRD VAL, HX5CNT”读取 X5 的高速计数当前值。

使用指令“SFRRD VAL, HX6CNT”读取 X6 的高速计数当前值。

指令中 VAL 参数为存储读取到的高速计数当前值，可为 D 寄存器、DM 存储器。

当高速计数当前值增为 0 或减为 0 时，若 EI 指令允许 X5 或 X6 中断源，则会响应中断，执行对应的中断程序。

3、测频率

当 X5、X6 工作于 16 位高速计数模式时自动进行测频率功能（不影响高速计数），可使用以下指令读出测频值（闸门时间内的计数值）：

使用指令“SFRRD VAL, HX5SPD”读取 X5 的高速计数测频值。

使用指令“SFRRD VAL, HX6SPD”读取 X6 的高速计数测频值。

指令中 VAL 参数为存储读取到的测频值，可为 D 寄存器、DM 存储器。

使用指令“SFRWR 闸门时间值, HDF_TV” 设置测频率闸门时间。

指令中闸门时间值参数可为常数、D 寄存器、DM 存储器，闸门时间值单位为 10ms，范围 0~255，上电默认闸门时间 500ms。每隔一个闸门时间特殊继电器 HDF_T 都会被接通，HDF_T 由用户来复位。

4、跳变捕获时间值

当 X5、X6 工作于下降沿中断和上升沿中断时，每当发生有效的沿跳变会自动捕获当时的内部时间值（0-65535），捕获到的时间值可由以下指令读取：

使用指令“SFRRD VAL, HX5CCR” 读取 X5 的跳变捕获时间值。

使用指令“SFRRD VAL, HX6CCR” 读取 X6 的跳变捕获时间值。

指令中 VAL 参数为存储读取到的跳变捕获时间值，可为 D 寄存器、DM 存储器。

捕获时间值单位可由以下指令设置：

使用指令“SFRWR VAL, HX5X6CK” 设置捕获时间值单位。

指令中 VAL 参数为要设置的捕获时间值单位（值*0.25us，默认 0.25us），可为常数、D 寄存器、DM 存储器。

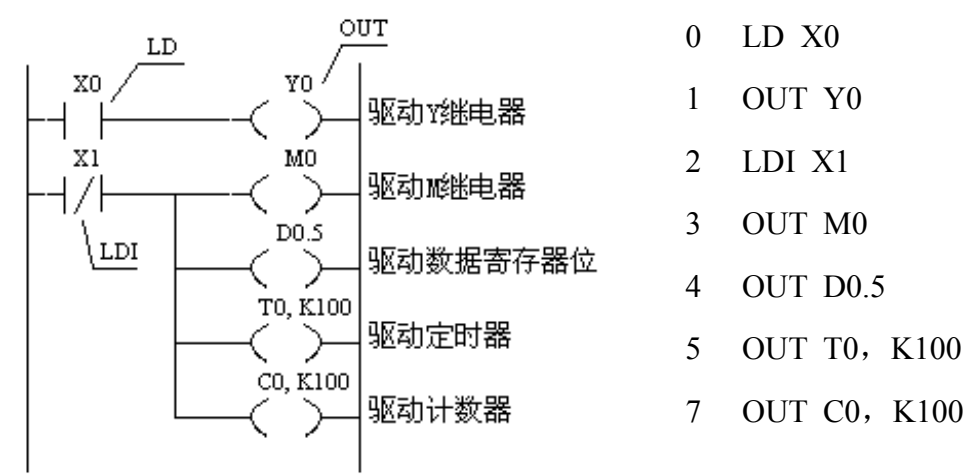
1.4 基本逻辑指令

1.4.1 逻辑取及输出线圈（LD/LDI/OUT）

LD、LDI、OUT 指令的功能、操作数等如下：

指令码	功能	操作数 1	操作数 2	步数
LD	常开触点逻辑取	X,Y,M,T,C,Dx.y, DMx.y	无	1 或 2
LDI	常闭触点逻辑取	X,Y,M,T,C,Dx.y, DMx.y	无	1 或 2
OUT	输出线圈驱动	Y,M,Dx.y, DMx.y	无	1 或 2
		T,C	K,D,RM,RY,RX,RC,DM	2

（注：x、y 表示元件的编号或数值，可以相同，也可以不同。下同）



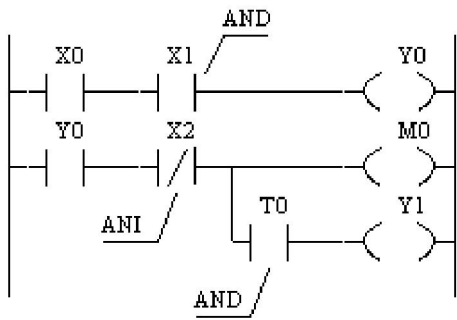
LD、LDI、OUT 的使用

- LD、LDI 指令用于将触点接到母线上。另外，与后述的 ANB 指令组合，在分支起点处也可使用。
- OUT 是驱动线圈的输出指令。可用于对输出继电器、辅助继电器、数据寄存器位、定时器、计数器的线圈驱动。但对输入继电器不能使用。
- OUT 指令可以连续使用若干次，相当线圈的并联（如上图中 OUT M0 和 OUT D0.5 等）。

1.4.2 触点串联（AND/ANI）

AND、ANI 指令的功能、操作数等如下：（无操作数 2）

指令码	功能	操作数 1	步数
AND	常开触点串联连接	X, Y, M, T, C, Dx.y, DMx.y	1 或 2
ANI	常闭触点串联连接	X, Y, M, T, C, Dx.y, DMx.y	1 或 2



```

0  LD X0
1  AND X1
2  OUT Y0
3  LD Y0
4  ANI X2
5  OUT M0
6  AND T0
7  OUT Y1

```

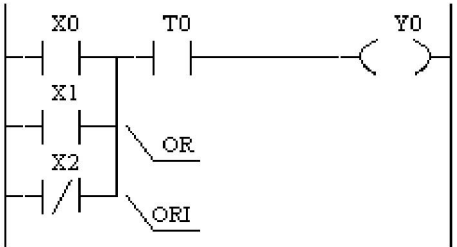
AND、ANI 的使用

- AND、ANI 指令是用于触点的串联连接。串联触点的个数没有限制，该指令可以多次重复使用。但因图形编程环境的限制，因此建议尽量做到一行不超过 10 个触点和一个线圈。

1.4.3 触点并联（OR/ORI）

OR、ORI 指令的功能、操作数等如下：（无操作数 2）

指令码	功能	操作数 1	步数
OR	常开触点并联连接	X, Y, M, T, C, Dx.y, DMx.y	1 或 2
ORI	常闭触点并联连接	X, Y, M, T, C, Dx.y, DMx.y	1 或 2



```

0  LD X0
1  OR X1
2  ORI X2
3  AND T0
4  OUT Y0

```

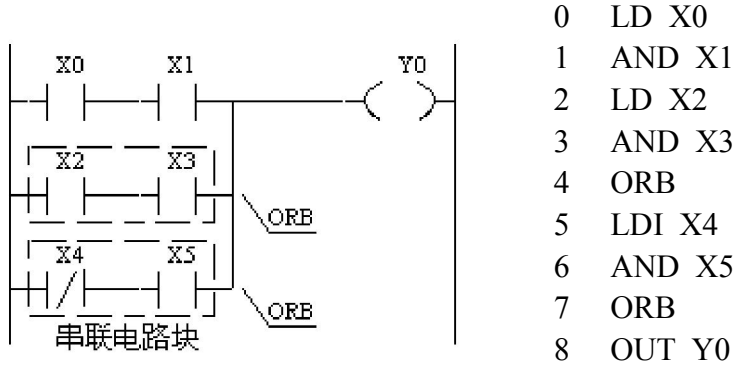
OR、ORI 的使用

- OR、ORI 用作为 1 个触点的并联连接指令。连接 2 个以上的触点串联连接的电路块的并联连接时，要用后述的 ORB 指令。
- OR、ORI 是从该指令的当前步开始，对前面的 LD、LDI 指令并联连接。并联的次数无限制。

1.4.4 串联电路块的并联（ORB）

ORB 指令的功能等如下：（无操作数）

指令码	功能	步数
ORB	串联电路块的并联连接	1



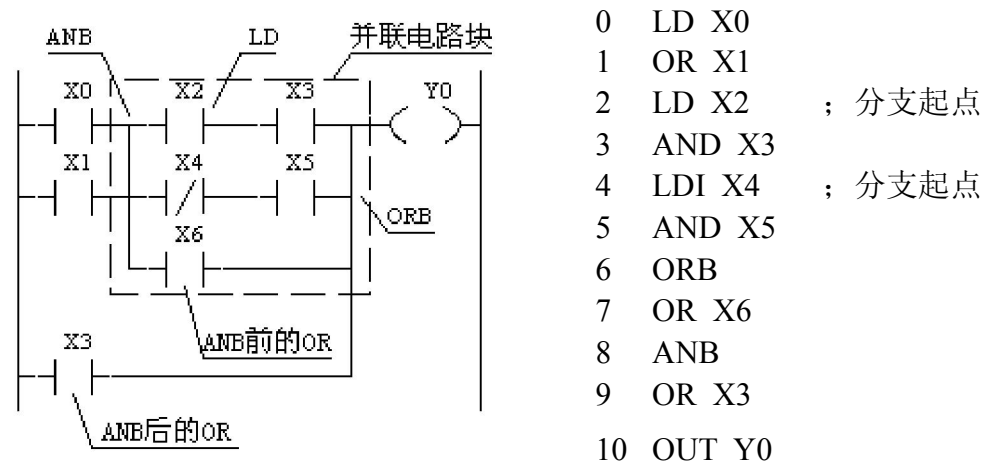
ORB 指令的使用

- 2 个以上的触点串联连接的电路称之为串联电路块。串联电路块并联连接时，分支的开始用 LD、LDI 指令，分支的结束用 ORB 指令。
- ORB 或 ANB 指令连续使用时，重复使用 LD、LDI 指令的次数限制在 8 次以下。

1.4.5 并联电路块的串联（ANB）

ANB 指令的功能等如下：（无操作数）

指令码	功能	步数
ANB	并联电路块之间的串联连接	1



ANB 指令的使用

- 分支电路并联电路块与前面电路串联连接时，使用 ANB 指令。分支的起始点用 LD、LDI 指令，并联电路块结束后，使用 ANB 指令与前面电

路串联。

- ORB 或 ANB 指令连续使用时，重复使用 LD、LDI 指令的次数限制在 8 次以下。

1.4.6 多重输出电路（MPS/MRD/MPP）

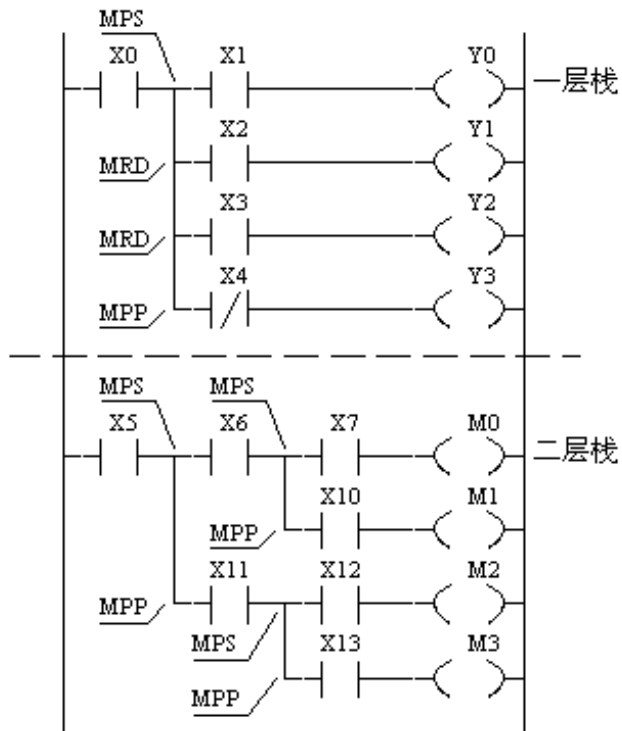
MPS、MRD、MPP 指令的功能等如下：（无操作数）

指令码	功能	步数
MPS	进栈	1
MRD	读栈	1
MPP	出栈	1

这组指令可将联接点先存储，因此可用于接后面的电路。

EasyLad 梯形图中有 8 个存储中间逻辑运算结果的存储区域，被称为逻辑堆栈存储器。

- MPS 指令用于把当时的逻辑运算结果压入逻辑堆栈存储器。
- MRD 指令用于读出逻辑堆栈存储器中最上层的数据作为该指令后的逻辑运算结果，读出时，栈内数据不发生移动。
- MPP 指令用于从逻辑堆栈存储器中向上弹出一个数据。
- MPS 指令与 MPP 指令必须成对使用。



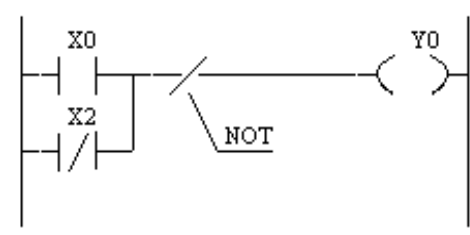
MPS、MRD、MPP 指令的使用

0	LD X0	15	AND X6
1	MPS	16	MPS
2	AND X1	17	AND X7
3	OUT Y0	18	OUT M0
4	MRD	19	MPP
5	AND X2	20	AND X10
6	OUT Y1	21	OUT M1
7	MRD	22	MPP
8	AND X3	23	AND X11
9	OUT Y2	24	MPS
10	MPP	25	AND X12
11	AND X4	26	OUT M2
12	OUT Y3	27	MPP
13	LD X5	28	AND X13
14	MPS	29	OUT M3

1.4.7 取反触点（NOT）

NOT 指令的功能等如下：（无操作数）

指令码	功能	步数
NOT	取反	1



0	LD X0
1	ORI X2
2	NOT
3	OUT Y0

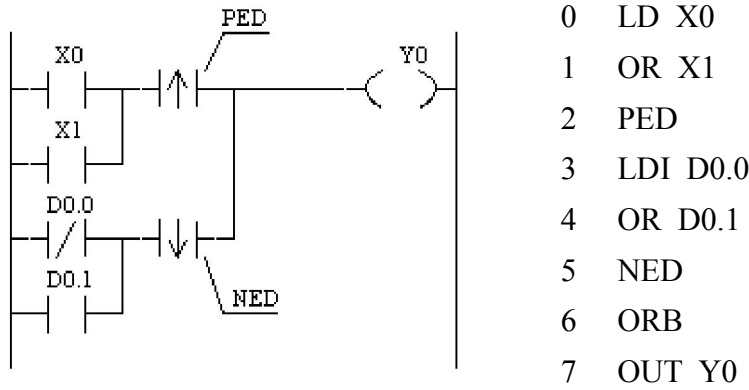
NOT 指令的使用

- NOT 指令为取反触点。当与该触点串联的左边的电路块为 ON（接通）时，经过该触点的逻辑运算结果为 OFF（断开）；当与该触点串联的左边的电路块为 OFF（断开）时，经过该触点的逻辑运算结果为 ON（接通）。
- 该触点左边必须且只能与电路块串联。即该触点在电路中只能处于逻辑与（AND）的位置，而不能处于逻辑取（LD）和逻辑或（OR）的位置。

1.4.8 边缘检测触点（PED/NED）

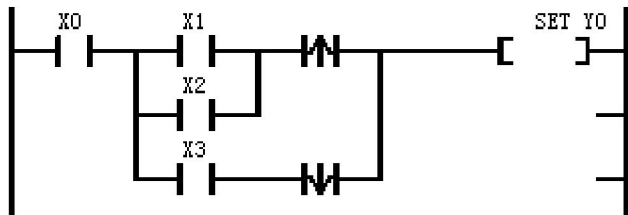
PED、NED 指令的功能等如下：

指令码	功能	步数
PED	正边缘检测	1
NED	负边缘检测	1



PED、NED 指令的使用

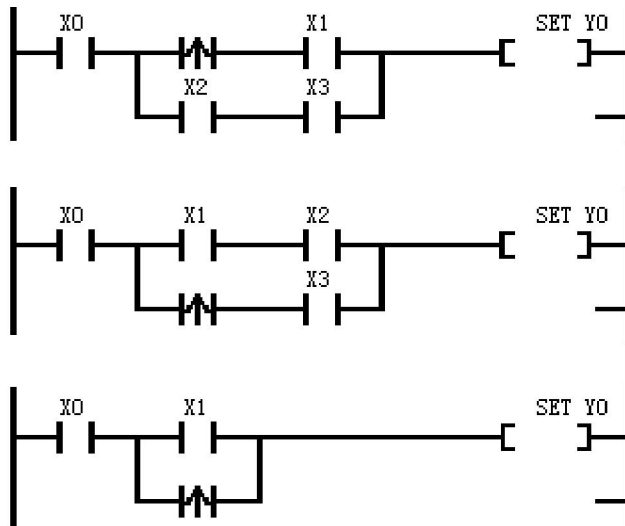
- PED 指令为正边缘检测触点。当且仅当该触点检测到与该触点串联的左边的电路块发生从 OFF（断开）到 ON（接通）的正跳变，也即该触点前的逻辑运算结果在上次扫描为 OFF、本次扫描为 ON 时，才使经过该触点的逻辑运算结果为 ON（接通），在其他情况下，经过该触点的逻辑运算结果为 OFF（断开）。
- NED 指令为负边缘检测触点。当且仅当该触点检测到与该触点串联的左边的电路块发生从 ON（接通）到 OFF（断开）的负跳变，也即该触点前的逻辑运算结果在上次扫描为 ON、本次扫描为 OFF 时，才使经过该触点的逻辑运算结果为 ON（接通），在其他情况下，经过该触点的逻辑运算结果为 OFF（断开）。
- 无操作数边缘检测触点（PED 和 NED）最多可使用 256 个。
- 该触点只检测左边串联的电路块是否发生跳变。其截止位置是左边的电路块连接有与该触点并联的电路为止。如下图：



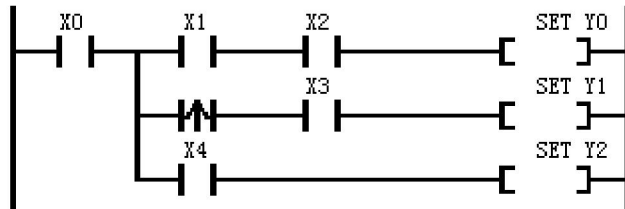
图中，正边缘检测触点只检测 X1 和 X2 的并联结果是否发生正跳变，而

不考虑 X0；负边缘检测触点只检测 X3 是否发生负跳变，也不考虑 X0。

- 该触点左边必须且只能与电路块串联。即该触点在电路中只能处于逻辑与（AND）的位置，而不能处于逻辑取（LD）和逻辑或（OR）的位置：

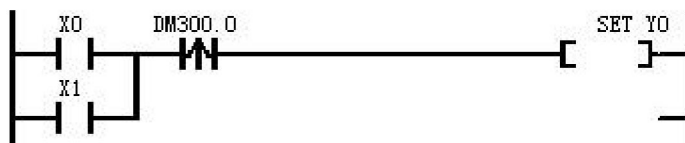


第 1 个图和第 2 个图中的边缘检测触点处于逻辑取（LD）的位置，因而是错误的；第 3 个图中的边缘检测触点处于逻辑或（OR）的位置，因而也是错误的。而在下图中：



边缘检测触点处于逻辑与（AND）的位置，因而是正确的。

- 也可使用带操作数的边缘检测触点（PED 和 NED），操作数为位 **DMx.y** 或位元件间接寻址 **Dx#偏移**，用 DMx.y 或间接寻址位元件来保存上个扫描周期该触点前的逻辑运算结果。带操作数的和无操作数的边缘检测触点动作一样，都是对该触点前的逻辑运算结果进行检测。带操作数的边缘检测触点使用次数不限，只和 DMx.y 的数量有关（每个检测触点都要使用一个不同的 DMx.y）。如：



操作数也可为 DM 字元件，此时由系统自动分配 DMx.y 在该 DM 字元件开始的数据块中。

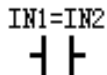
1.4.9 比较触点 (=、<、>、<=、>=、<>)

比较触点有等于(=)、小于(<)、大于(>)、小于等于(<=)、大于等于(>=)、不等于(<>)共 6 类触点。每个触点均可进行逻辑取 (LD)、逻辑与 (AND)、逻辑或 (OR) 等逻辑操作。

参加比较的两个操作数的数据类型应一致，即整型 (字) 只能和整型 (字) 相比较，长整型 (双字) 只能和长整型 (双字) 相比较，浮点型只能和浮点型相比较。

比较触点均使用到特殊辅助继电器 EQ、LT、GT。其各个触点的表示方式和描述如下：

- 等于触点

$IN1=IN2$


若 IN1 等于 IN2 则该触点接通 (ON)，否则该触点断开 (OFF)。

操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～
IN2	常数、D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～

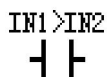
- 小于触点

$IN1<IN2$


若 IN1 小于 IN2 则该触点接通 (ON)，否则该触点断开 (OFF)。

操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～
IN2	常数、D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～


- 大于触点

$IN1>IN2$


若 IN1 大于 IN2 则该触点接通 (ON)，否则该触点断开 (OFF)。


操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～
IN2	常数、D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～

● 小于等于触点

$IN1 \leq IN2$
 若 IN1 小于等于 IN2 则该触点接通 (ON) , 否则该触点断开 (OFF)。


操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～
IN2	常数、D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～

● 大于等于触点

$IN1 \geq IN2$
 若 IN1 大于等于 IN2 则该触点接通 (ON) , 否则该触点断开 (OFF)。

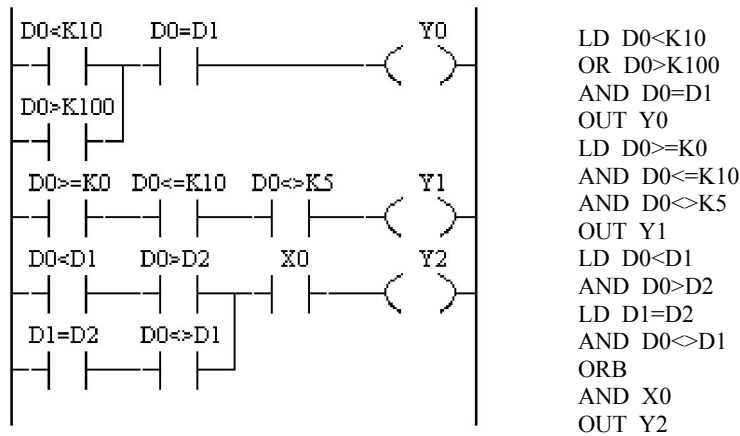
操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～
IN2	常数、D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～

● 不等于触点

$IN1 \neq IN2$
 若 IN1 不等于 IN2 则该触点接通 (ON), 否则该触点断开 (OFF)。

操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～
IN2	常数、D、RM、RY、RX、RC、RT、DM256～、LDM256～、FDM256～

例：

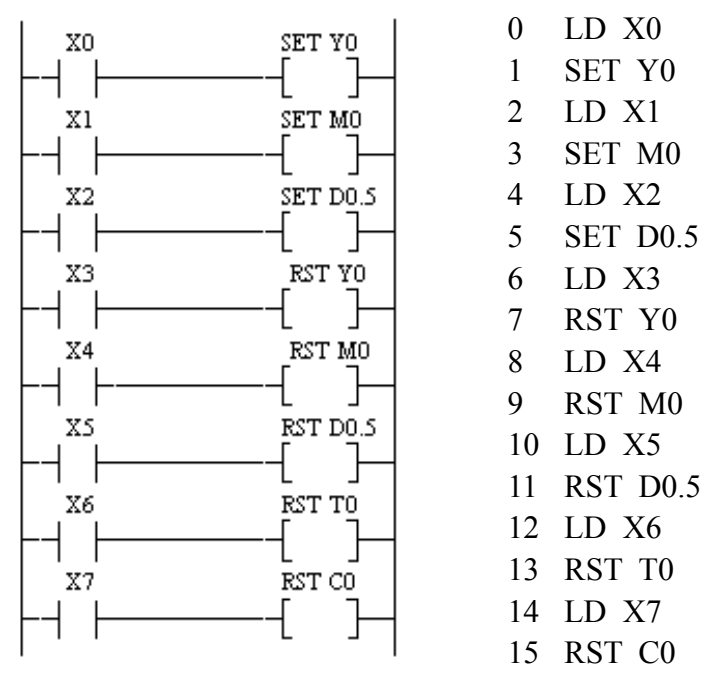


比较触点的使用

1.4.10 置位、复位与取反线圈（SET/RST/INV）

SET、RST、INV 指令的功能、操作数等如下：（无操作数 2）

指令码	功能	操作数 1	步数
SET	使元件置位	Y, M, Dx.y, DMx.y	1 或 2
RST	使元件复位	Y, M, T, C, Dx.y, DMx.y	1 或 2
INV	使元件取反	Y, M, Dx.y, DMx.y	1 或 2



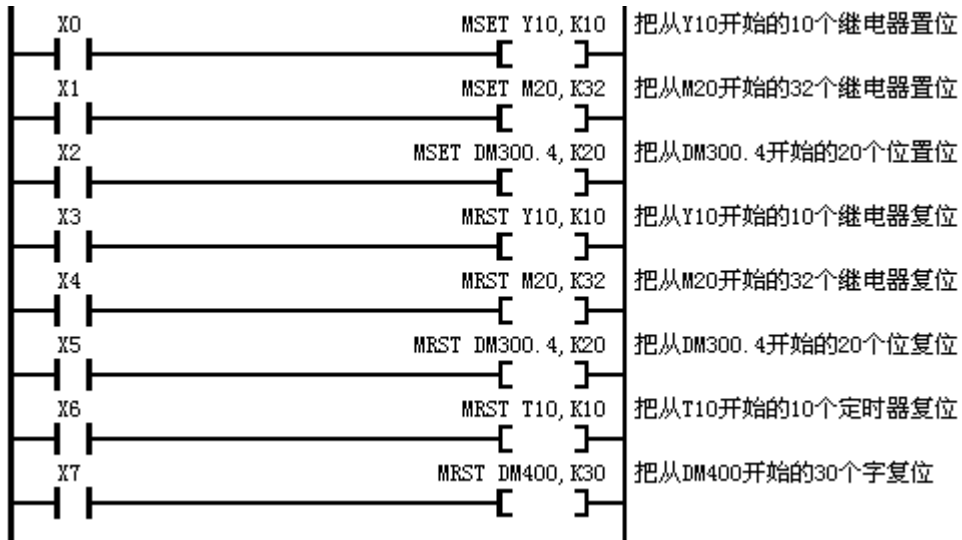
SET、RST 指令的使用

- SET 为置位指令。若该指令被接通，则使所驱动的元件置位；若该指令被断开，则不执行任何操作。
- RST 为复位指令。若该指令被接通，则使所驱动的元件复位；若该指令被断开，则不执行任何操作。
- INV 为位取反指令。若该指令被接通，则使所驱动的元件取反；若该指令被断开，则不执行任何操作。

1.4.11 多点置位和多点复位线圈（MSET/MRST）

MSET、MRST 指令的功能、操作数等如下：

指令码	功能	操作数 1	操作数 2	步数
MSET	使连续的多个元件置位	Y, M, Dx.y, DMx.y	常数	2 或 3
MRST	使连续的多个元件复位	Y, M, Dx.y, DMx.y, T, C, DM	常数	2 或 3



MSET、MRST 指令的使用

- MSET 为多点置位指令。若该指令被接通，则使所驱动的元件开始的多个元件（最多 64 个）置位；若该指令被断开，则不执行任何操作。
- MRST 为多点复位指令。若该指令被接通，则使所驱动的元件开始的多个元件（最多 64 个）复位；若该指令被断开，则不执行任何操作。

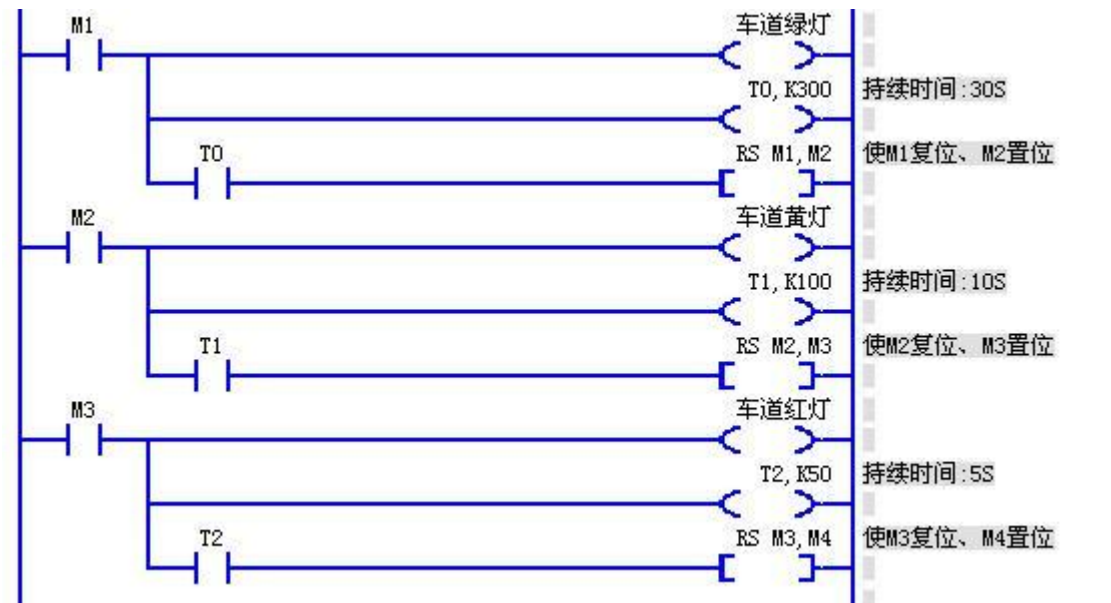
1.4.12 复位置位线圈（RS）

RS 指令的操作数如下：

指令码	操作数 1	操作数 2
RS	Y, M, Dx.y, DMx.y	Y, M, Dx.y, DMx.y

功能：若该指令被接通，则使操作数 1 复位、操作数 2 置位。若该指令被断开，则不被执行。

梯形图例子如下



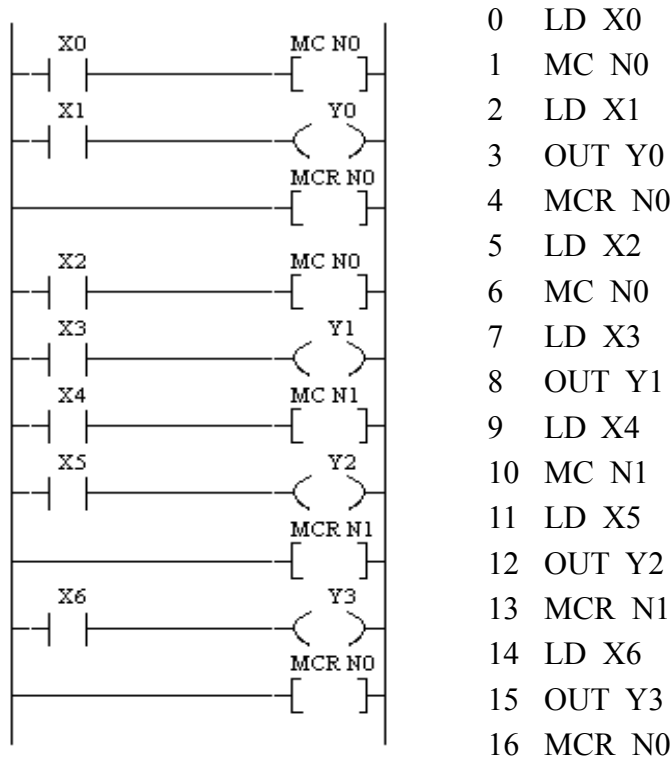
RS 指令的使用

1.4.13 主控线圈（MC/MCR）

MC、MCR 指令的功能、操作数等如下：（无操作数 2）

指令码	功能	操作数 1	步数
MC	主控开始	Nx	1
MCR	主控结束	Nx	1

- MC 为主控指令，用来表示主控电路块的开始。
MCR 为主控复位指令，使主控指令返回母线，用来表示主控电路块的结束。
Nx 为主控编号，x 为 0~254 共 255 个编号。对于同一编号的主控指令的使用次数不限，但 MC Nx 指令必须被编号相同的 MCR Nx 指令返回。
- 若 MC 指令被接通，则 MC 与编号相同的 MCR 之间的指令正常执行；若 MC 指令被断开，则 MC 与编号相同的 MCR 之间的线圈类指令（输出线圈和指令线圈）均被强制断开，其他指令均被跳过不执行。
- 当 MC 指令被断开时，在 MC 指令对之间的边缘检测指令（PED、NED）停止检测并保持其在 MC 指令被断开之前检测的状态。
- MC 指令使用后必须要用 MCR 指令返回母线，否则程序将出错。



MC、MCR 指令的使用

1.4.14 空操作指令（NOP）

NOP 指令的功能等如下：（无操作数）

指令码	功能	步数
NOP	空操作	1

1.4.15 程序结束（END）

END 指令的功能等如下：（无操作数）

指令码	功能	步数
END	程序结束	1

- END 为程序结束指令。当程序扫描到 END 指令时，则进行输入输出处理、刷新警戒时钟等操作，使程序回到第 0 步继续执行。END 后面的程序则不被处理。
- 在一个完整的程序最后必须有 END 指令。

1.5 步进指令

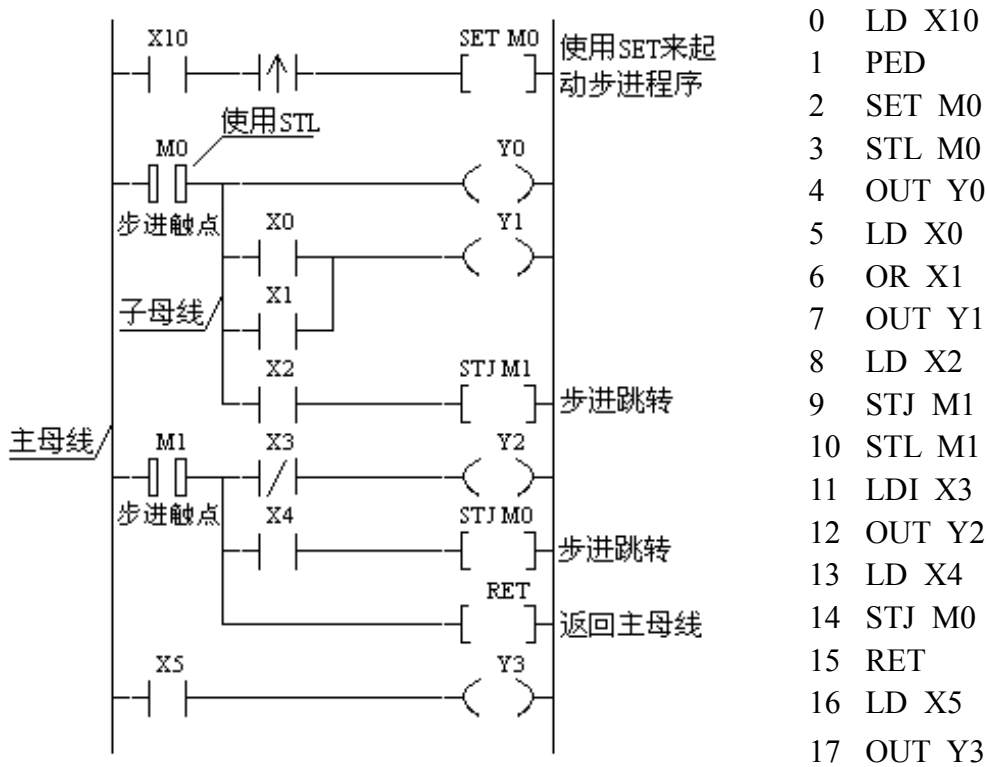
1.5.1 步进梯形图的介绍

要用继电器梯形图编制顺序控制程序需要有些经验，并且所编的复杂程序也难以读懂。

若用步进梯形图来表示，则编程就很方便，并且所编的程序可读性也很强。

步进触点是构成步进梯形图的重要元素，用来表示步进梯形程序中的一个工步。若步进触点闭合，则表示该工步正在执行。步进触点由辅助继电器 M 组成，当辅助继电器 M 由 STL 指令所驱动时，该辅助继电器即为一个步进触点。

下图即为一个步进梯形图。



步进梯形图的例子

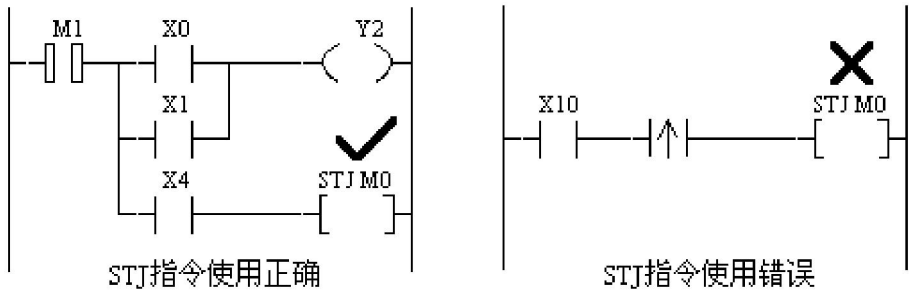
步进指令的功能、操作数等如下：（无操作数 2）

指令码	功能	操作数 1	步数
STL	步进触点加载	Mx	1
STJ	步进跳转	Mx	1
SNEXT	转移到下面的工步	无	1
SLAST	转移到上面的工步	无	1
RET	步进返回主母线	无	1

- STL 为步进触点加载指令，用来加载一个步进触点与左母线或其他步进

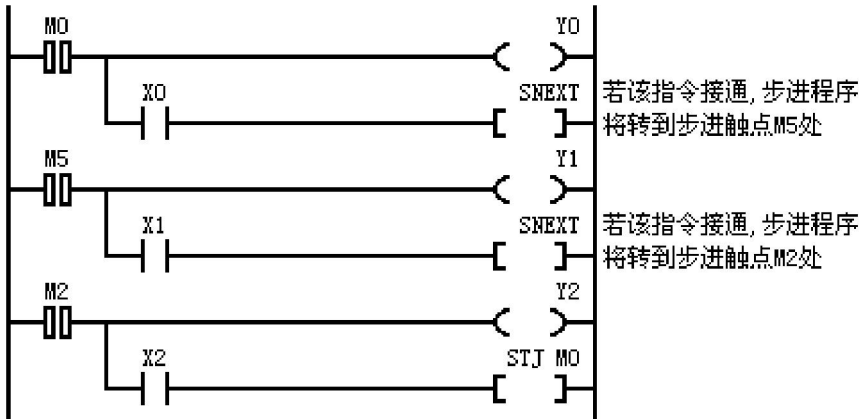
触点连接，表示一个工步的开始。加载后，母线（LD、LDI 点）将移至该触点的右边（除非右边还为步进触点）成为子母线，子母线可以直接连接线圈或通过触点驱动线圈（如上图）。若要返回原来的主母线，则使用 RET（步进返回）指令。

- STJ 为步进跳转指令。用来使步进程序转移到指定的工步运行。当该指令被接通时，将使该指令所驱动的辅助继电器置位，使步进程序转移到该辅助继电器所表示的步进触点处运行，同时本工步的步进触点自动复位；当该指令被断开时将不执行任何操作。该指令必须由步进触点的子母线所驱动。如下图所示：



STJ 指令的正确使用

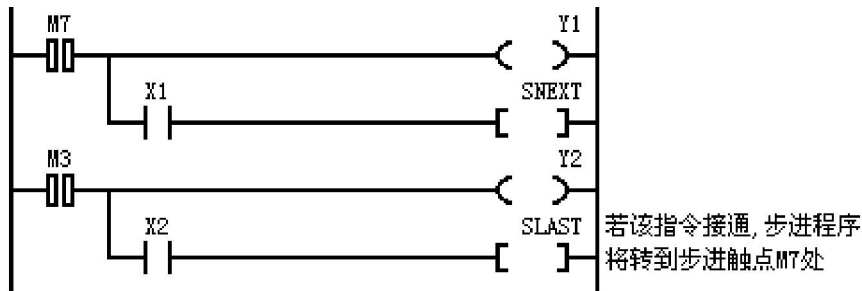
- SNEXT 为转移到下面的工步指令。当该指令被接通时，将使步进程序转移到该指令所在工步的下面的工步处运行，同时本工步的步进触点自动复位。当该指令被断开时将不执行任何操作。该指令必须由步进触点的子母线所驱动。



SNEXT 指令的使用

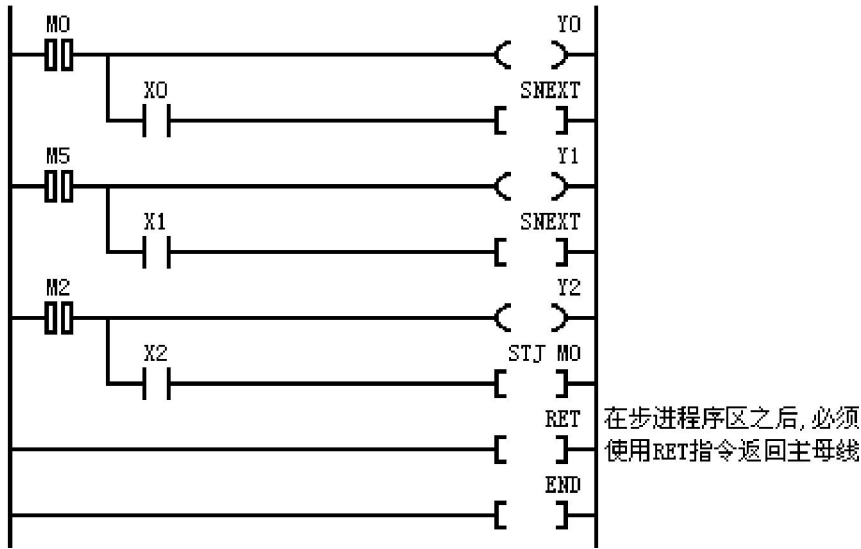
- SLAST 为转移到上面的工步指令。当该指令被接通时，将使步进程序转移到该指令所在工步的上面的工步处运行，同时本工步的步进触点自动复位。当该指令被断开时将不执行任何操作。该指令必须由步进触点的

子母线所驱动。



SLAST 指令的使用

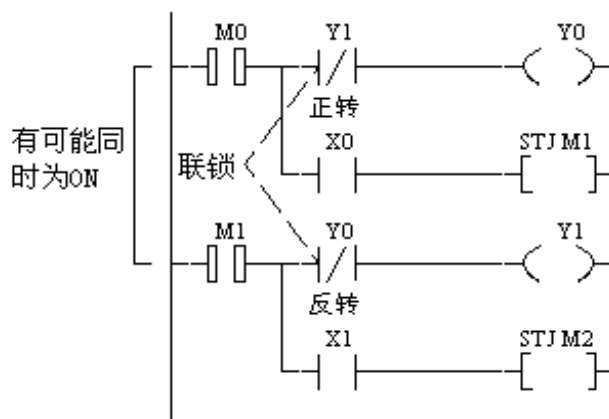
- RET 为步进返回指令。用于使步进程序返回到原来的主母线。在一系列的 STL 指令的最后，必须要使用 RET 指令返回到主母线。



RET 指令的使用

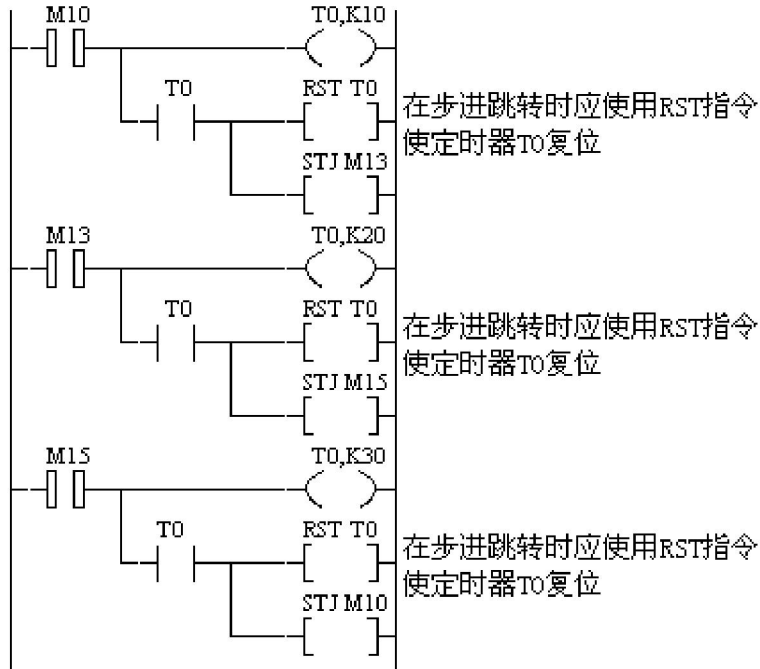
1.5.2 步进梯形图的详细动作

- 当步进触点闭合（为 ON），与此连接的电路就运行。当步进触点断开（为 OFF），与此连接的电路停止运行，在其中的线圈类指令被强制断开一个扫描周期后（触点类指令均被跳过不处理），这部分电路的指令就被跳过不再执行，但还占用程序处理时间。
- 当步进触点断开后，首先与该步进触点直接连接（在其他触点和线圈之前）的上升沿检测触点复位，该上升沿检测触点可用来检测该步进触点的上升沿，该步进触点所连接的电路中的其他边缘检测指令（PED、NE D）停止检测并保持其在步进触点断开之前检测的状态。
- 在步进程序的一个工步内（从 STL 指令开始到下一个 STL 指令或到 RET 指令为止之间），可以使用 MC/MCR、子程序调用（CALL）等程序流控制指令，在这些程序流控制指令之间也可使用步进程序块（一系列相互有联系的 STL 指令和 RET 指令组成）。但这些程序流控制指令若跨步或交叉使用，会使动作复杂化。故建议只能嵌套使用，不要交叉使用。
- 在中断程序中不可使用步进程序。
- 步进转移过程中，在一个扫描周期中两个工步同时为 ON 的情况也可能出现。因此，如下图不能同时为 ON 的一对输出之间（例如同一电动机的正转与反转）必须加上联锁，防止同时为 ON。



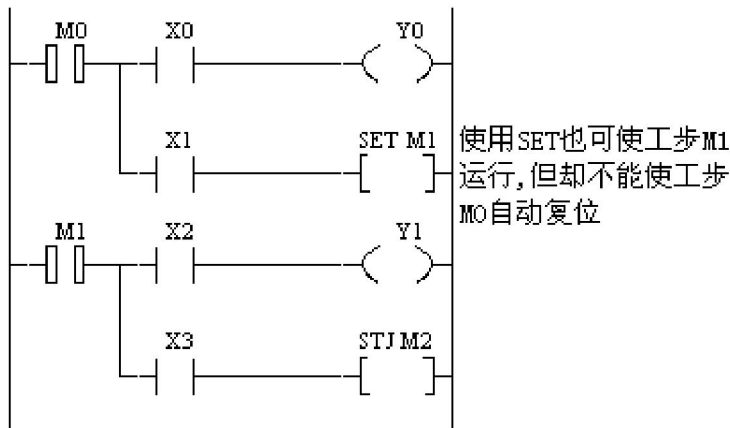
不能同时为 ON 的输出的处理

- 对于不可能同时为 ON 的工步，可以重复使用同一个定时器等线圈（计数器除外）。但若重复使用同一个定时器，则应在步进跳转（执行 STJ 指令）时使用 RST 指令使该定时器复位。



在不同的工步重复使用同一个定时器的处理

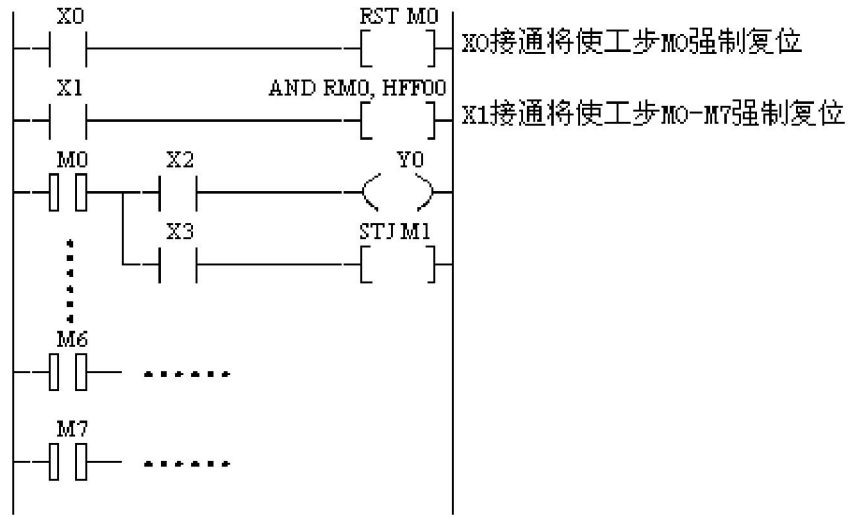
- 若使用 SET 指令使步进触点（M）置位，也可以使该步进触点所表示的工步运行，但却不能使步进触点自动复位。



使用 SET 指令来运行步进程序

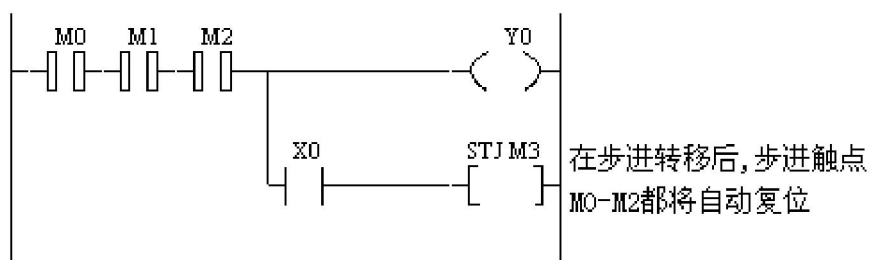
当步进程序中所有工步都停止运行时，必须使用 SET 指令来启动步进程序，使某一工步处于运行状态。

- 若使步进触点复位的指令（如 RST 等）被接通，则可使该步进触点所表示的工步立即强制复位，停止运行，而不必有步进转移动作发生，但必须保证在该步进触点被复位后，该步进触点及其所连接的电路至少再被程序扫描 1 次。



步进触点的强制复位

- 步进触点可以串联，但不能并联。串联时表示并行汇合处理，即当这些所串联的步进触点都为 ON 时，才使所连接的电路运行。步进触点的最大可串联数目为 8 个。当在步进触点串联的电路中使用输出线圈（包括定时器和计数器线圈）时，应注意会同其他电路产生双线圈输出冲突的影响。



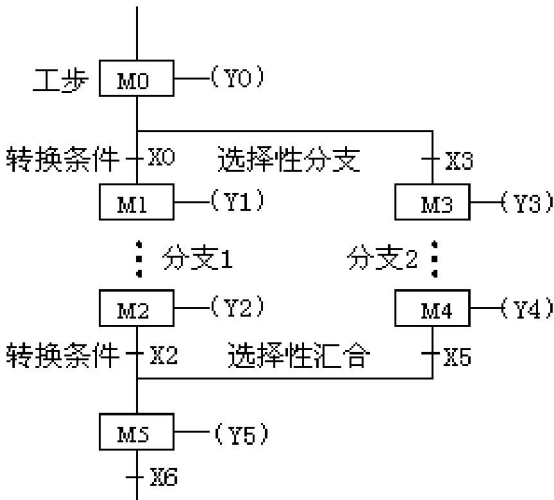
步进触点的串联

- 在整个程序中，可同时最多有 8 个步进触点为 ON，即程序允许最多有 8 条分支在同时运行。若某一步进触点由 ON 变为 OFF 后，但该步进触点及其所连接的电路没再被扫描过，则该步进触点还将继续占用 1 条运行分支。

1.5.3 分支和汇合的编程

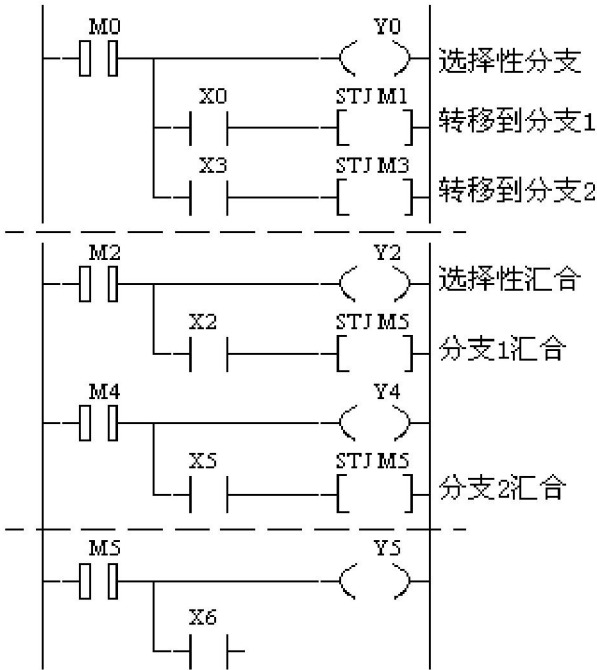
- 选择性分支和选择性汇合

从多个分支流程中选择某一个单支流程，称之为选择性分支。多个分支流程的汇合状态可由其中一个分支进入，而不管其他分支是否完成，称之为选择性汇合。如下图：



选择性分支和汇合

选择性分支和选择性汇合的编程如下：

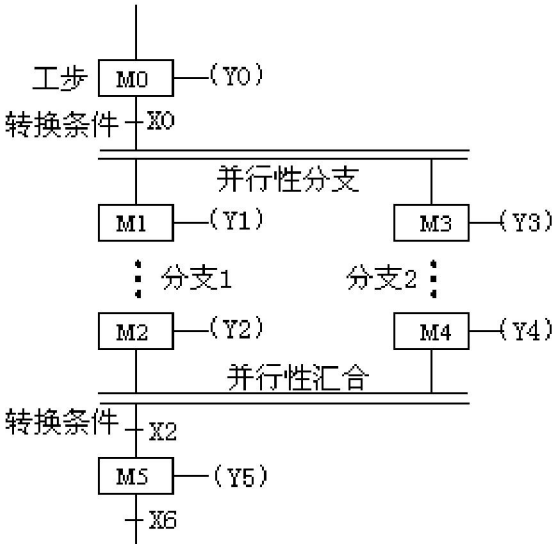


选择性分支和汇合的编程

对各个步进跳转指令分别设置转移条件，可实现选择性分支。使各个工步的步进跳转指令都转移到一个步进触点，可实现选择性汇合。

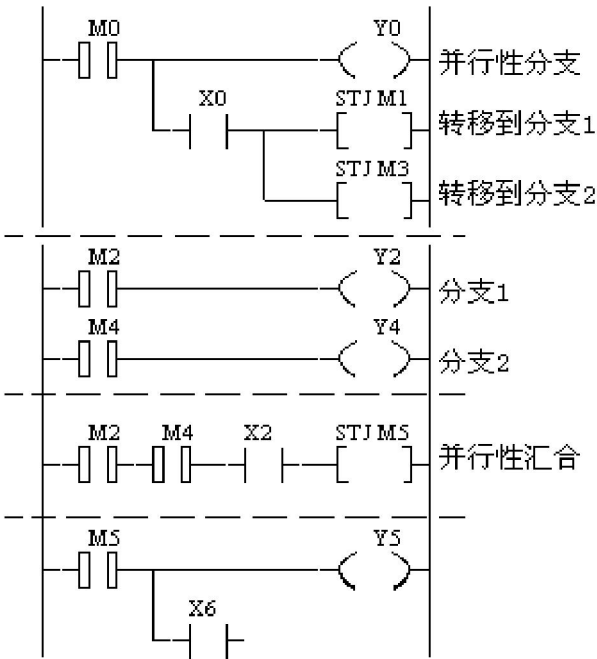
● 并行性分支和并行性汇合

当一个控制状态流分离成多个分支时，所有的分支流程必须同时处理，称之为并行性分支。当多个分支汇合时，必须等到所有的分支都完成后才能汇合到下一个状态，称之为并行性汇合。如下图：



并行性分支和汇合

并行性分支和并行性汇合的编程如下：



并行性分支和汇合的编程

通过对步进跳转指令线圈的并联可实现并行性分支。通过对步进触点的串联可实现并行性汇合。

1.6 功能指令

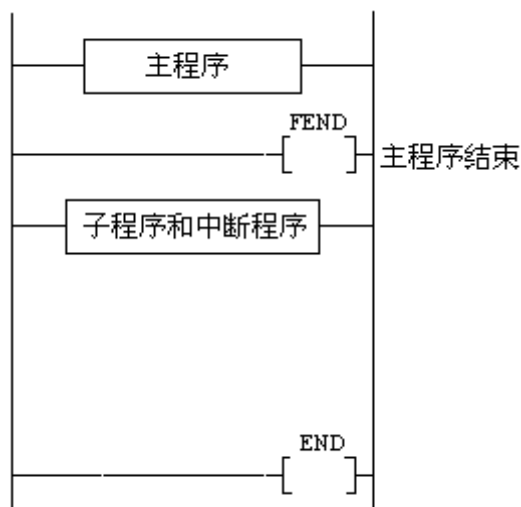
1.6.1 主程序结束指令（FEND 工作/MEND 调试）

- **FEND**

指令步数：1 步。

FEND 指令表示正常工作状态的主程序结束。执行到 FEND 指令时机器进行输出处理、输入处理、警戒时钟刷新，完成以后返回到第 0 步。

子程序和中断程序必须位于 FEND 或 MEND 指令之后。



FEND 指令和程序结构

- **MEND**

指令步数：1 步。

MEND 指令表示调试状态的主程序结束。执行到 MEND 指令时机器进行输出处理、警戒时钟刷新，完成以后返回到第 0 步，但不刷新输入继电器，输入继电器的状态可由鼠标来设置。

在程序调试时可使用 MEND 指令代替 FEND 指令作为主程序结束，此时可用鼠标改变输入继电器的状态来模拟外接输入开关，当程序正式投入运行时，要用 FEND 指令替换掉 MEND 指令。

1.6.2 程序暂停（放置断点）指令（STOP）

● STOP IN

指令步数：1 步（无操作数）或 3 步（有操作数）。

操作数	可使用的元件
IN	M、DMx.y

IN 为可选参数

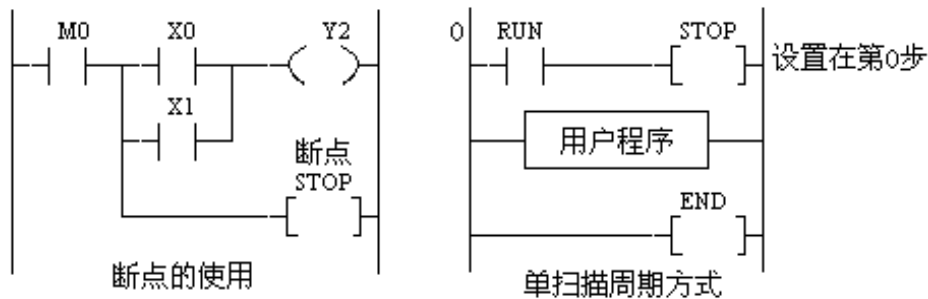
★当无操作数时为不可关闭的断点指令，功能如下：

扫描到该指令时，若该指令被接通，则程序暂停在该指令处，直到点工具栏的“运行”按钮时，才使程序向下运行。

★当有操作数 IN 时为可关闭的断点指令，功能如下：

扫描到该指令时，若该指令被接通且 IN 为 ON，则程序暂停在该指令处，直到点工具栏的“运行”按钮或者把 IN 置为 OFF 时才使程序向下运行。若 IN 为 OFF 时，则即使该指令被接通也不执行程序暂停功能（即该指令失效）。

利用该指令可以在程序调试时放置断点，或让程序以单扫描周期方式运行等功能。



STOP 指令的使用

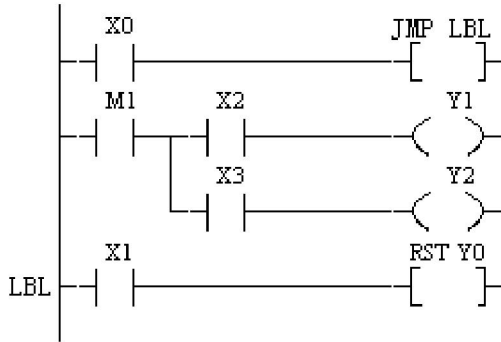
在单扫描周期方式中，用户每点工具栏的“运行”按钮一次，将使 PLC 运行一个扫描周期。

1.6.3 立即跳转指令（JMP）

- **JMP “标号”**

指令步数：2 步。

JMP 为立即跳转指令。该指令被接通，将使程序立即跳到该指令所指定的程序标号处执行，同时刷新警戒时钟；该指令被断开，则程序按顺序执行该指令的下面一条指令。



JMP 指令的使用

当 JMP 指令被接通时，被跳过的程序将不再被扫描，因此，被跳过的程序不占用扫描时间。JMP 指令可以向后跳转，也可以向前跳转，但只能在主程序之中或同一子程序之中（SUBS 和 SRET 之间）跳转。

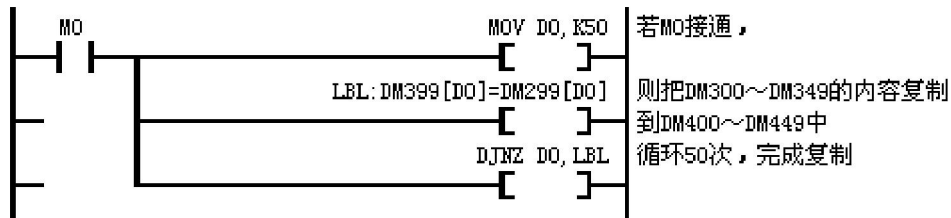
1.6.4 减 1 不为 0 跳转指令（DJNZ）

- **DJNZ OUT, “标号”**

指令步数：2 步或 3 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址

若该指令被接通，则把 OUT 的内容减 1，若减 1 后的结果不为 0，则使程序跳转到该指令所指定的程序标号处执行，若减 1 后的结果为 0，或者该指令被断开，则程序按顺序执行该指令的下面一条指令。



DJNZ 指令的使用

1.6.5 子程序指令（CALL/SUBS/SRET/CRET）

- **CALL “标号”**

指令步数：2 步。

CALL 为子程序调用指令。若该指令被接通，则调用该指令所指定的程序标号处的子程序，子程序返回后，程序按顺序执行该指令下面的指令。

若该指令被断开，则不调用子程序，直接执行该指令下面的指令。

子程序和自定义函数最多允许 8 级嵌套但其个数无限制。

- **SUBS**

指令步数：1 步。

SUBS 为子程序开始指令。一个子程序必须以 SUBS 指令为开始，子程序的入口标号必须位于该指令处。若 CALL 指令调用的标号处的指令不为 SUBS 指令，程序将出错。

在一个函数的内部禁止使用该指令。

- **SRET**

指令步数：1 步。

SRET 为子程序返回指令。执行子程序时，若碰到该指令，程序将返回到调用该子程序的 CALL 指令的下面那条指令处执行。

建议用户在一个子程序中只有一个 SRET 指令，且位于子程序的最后。

在一个函数的内部禁止使用该指令。

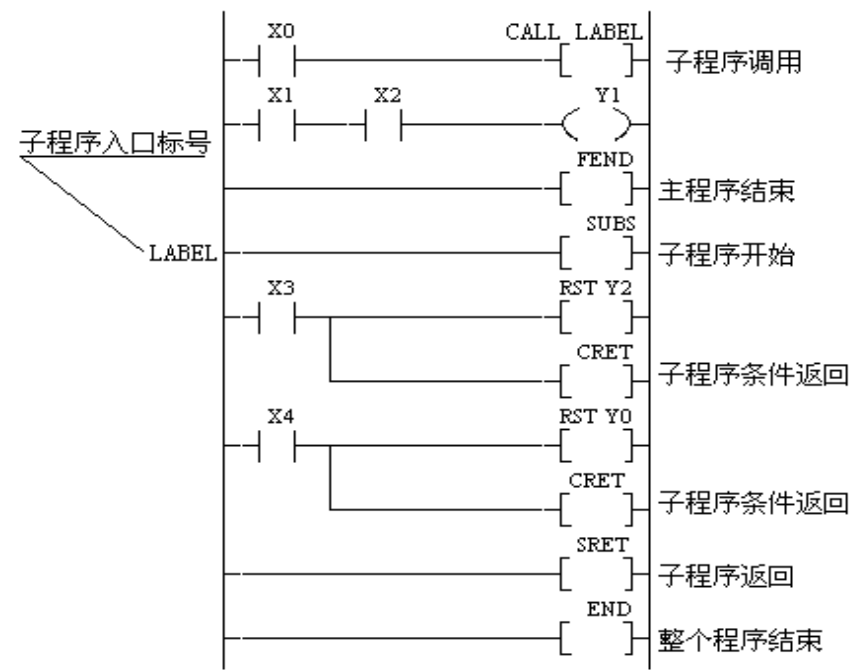
- **CRET**

指令步数：1 步。

CRET 为子程序条件返回指令。若该指令被接通，则执行子程序返回，程序将返回到调用该子程序的 CALL 指令的下面那条指令处执行；若该指令被断开，则程序按顺序执行该指令的下面一条指令。

在一个函数的内部禁止使用该指令。

子程序指令的梯形图例子如下：



CALL、SUBS、SRET、CRET 指令的使用

1.6.6 编程口通讯地址设置指令（CADDR）

● CADDR IN

指令步数：1 步或 2 步。

操作数	可使用的元件
IN	K、D、RM、RY、RX、RC、RT、DM、变址寻址

该指令为通讯地址设置指令。当该指令被接通时，将把 IN 的内容（仅低字节有效）送入 PLC 的通讯地址寄存器中。若该指令被断开，则不执行任何操作。

在 PLC 上电后，将把通讯地址寄存器的内容设置为 1。

1.6.7 中断处理指令 (TIM/INT/IRET/CRETI/EI/DI)

- **TIM Kx**

指令步数：1 步。

该指令为设置定时中断时间间隔指令。Kx 为设置的时间间隔，范围为 0~255，单位为 ms。当 Kx 为 K0 时停止定时，定时中断的定时器及其中断申请被复位，不产生定时中断。当 Kx 为其他值时，则每隔所设定的时间间隔将产生一次中断申请，若定时中断开，则响应中断，执行定时中断服务程序。

该指令被接通时执行上述功能，被断开时不执行。一旦该指令被接通，所设置的时间间隔即有效，即使再被断开，其设置的时间间隔还是有效的，直到另一条“TIM”指令被接通。

- **INT Kx**

指令步数：1 步。

该指令为中断服务程序入口指令。该指令定义了所指定中断源的中断服务程序的入口位置。各个中断源的中断服务程序都必须以该指令为开始。Kx 为中断源的编号。

各个中断源的编号如下（括号内为编号的预定义标识符）：

定时中断源：K0 (I_TIM)

高速计数器 C0 中断源：K1 (I_HC0)

高速计数器 C1 中断源：K2 (I_HC1)

高速计数器 C2 中断源：K3 (I_HC2)

外部输入 X3 的中断源：K4 (I_X3)

外部输入 X4 的中断源：K5 (I_X4)

PTO 脉冲串输出完成中断源：K6 (I_PTO)

外部输入 X5 的中断源：K7 (I_X5)

外部输入 X6 的中断源：K8 (I_X6)

中断服务程序必须位于 FEND 指令之后。

中断服务程序中若使用了 D2~D15、DM256~DM271 并且改变了这些元件的值，则应注意对这些元件的现场保护（可使用 PUSH/POP 指令进

行现场保护)。而 D0、D1 和 EQ、LT、GT、CF、OV、GF0 等特殊辅助继电器由系统自动保护故不需要再保护。

● **IRET**

指令步数：1 步。

该指令为中断返回指令。执行中断服务程序时，若碰到该指令，则程序返回到原来中断处重新执行被中断的程序。CPU 在执行完该指令后，至少需要再执行一条指令才会响应新的中断请求。

建议用户在一个中断服务程序中只有一个 IRET 指令，且位于中断服务程序的最后。

● **CRETI**

指令步数：1 步。

该指令为中断条件返回指令。

若该指令被接通，则执行中断程序返回，程序返回到原来中断处重新执行被中断的程序；

若该指令被断开，则程序按顺序执行该指令的下面一条指令。

CPU 在执行中断程序返回后，至少需要再执行一条指令才会响应新的中断请求。

● **EI Hx**

指令步数：1 步。

该指令为开中断指令。

该指令被接通，则允许 Hx（单字常数）中为 1 的位对应的中断源中断，为 0 的位对应的中断源不受影响（即以前允许中断则还允许，以前禁止中断则还禁止）。

该指令被断开，则不执行任何操作。

只有全局中断位也被打开时才允许各个中断源中断。

Hx 中各位的定义：（位 0—最低位）

位	9	8	7	6	5	4	3	2	1	0
中断源	X6	X5	全局	PTO	X4	X3	C2	C1	C0	定时

● **DI Hx**

指令步数：1 步。

该指令为关中断指令。

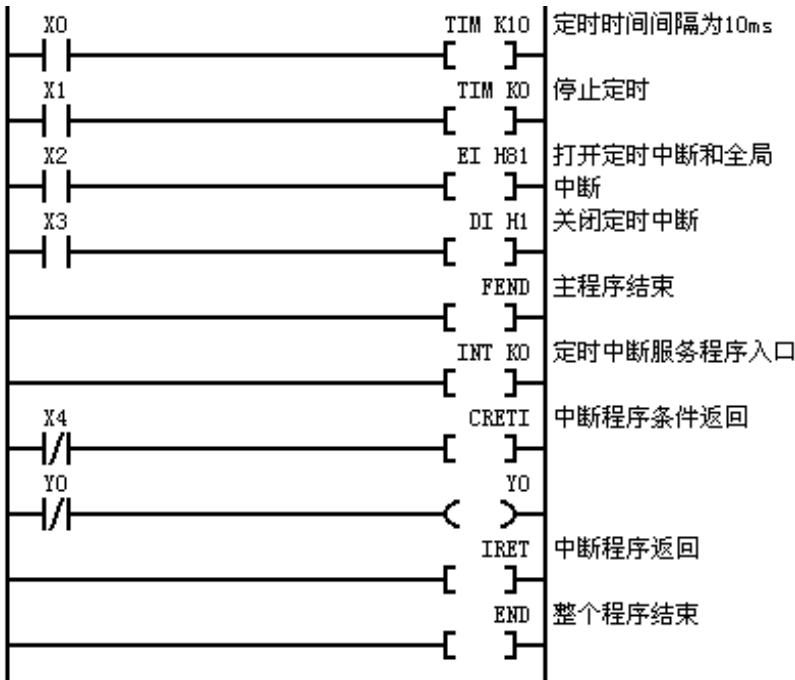
该指令被接通，则禁止 Hx（单字常数）中为 1 的位对应的中断源中断，为 0 的位对应的中断源不受影响（即以前允许中断则还允许，以前禁止中断则还禁止）。

该指令被断开，则不执行任何操作。

若全局中断位被关闭，则将禁止所有中断源中断。

Hx 中各位的定义：（位 0—最低位）

位	9	8	7	6	5	4	3	2	1	0
中断源	X6	X5	全局	PTO	X4	X3	C2	C1	C0	定时



中断处理指令的使用

1.6.8 脉冲输出指令（PTO/ PWM/PLS）

● PTO IN1, IN2

指令步数：1 步。

操作数	可使用的元件
IN1	D、RM、RY、RX、RC
IN2	Y0、Y1

说明：

脉冲串输出设置指令。输出指定数量的脉冲串。

该指令有如下操作：

若该指令被接通：如果 IN1 为 D0，则为 PWM 方式（PWM 指令有效）；
否则为脉冲串输出方式，此时把从 IN1 开始的 7 个寄存器作为 IN2（Y0
或 Y1）的脉冲串设置寄存器，各个寄存器的作用如下：

IN1[0]为 0。

IN1[1]为 0。

IN1[2]为 0。

IN1[3]为 0。

IN1[4]、IN1[5]为脉冲输出总的脉冲个数（32 位）。

IN1[6]为起始脉冲周期值。

每输出 1 个脉冲，将把 IN1[4]、IN1[5]（32 位）的内容减 1，当减到 0
时停止输出脉冲，并把 IN2 的脉冲串输出完成标记（Y0 为 PTO0F，Y1
为 PTO1F）置 ON，若允许中断，则发出中断申请。

当脉冲串输出完成后，IN1[4]、IN1[5]、IN1[6]的内容都发生变化，若要
再输出脉冲串，必须重新设置这些寄存器。

在脉冲输出过程中，若把 IN1[4]、IN1[5]设置为 0，则立即停止输出脉冲，
同时把 IN2 的脉冲串输出完成标记（Y0 为 PTO0F，Y1 为 PTO1F）置 O
N，若允许中断，则发出中断申请。

当无脉冲串输出时，执行该指令后，则立即开始输出脉冲（IN1[4]、IN1
[5]必须不为 0）。

脉冲周期的计算为：**周期 = 值 × 0.5us**

若该指令被断开，则不执行任何操作。

当程序开始运行时，系统将默认为 PWM 方式。在程序运行中，可使用该指令设置为 PWM 方式或脉冲串输出方式。

注：在某通道正在输出脉冲串的过程中，不可再执行该通道的 PTO 指令，直到该通道的脉冲串输出完后才可再执行。

周期设置值 = 2000000 / 频率

● PWM IN1, IN2

指令步数：1 步或 2 步。

操作数	可使用的元件
IN1	K、D、RM、RY、RX、RC、RT、DM、变址寻址
IN2	Y0、Y1

说明：

PWM 脉冲宽度设置指令。

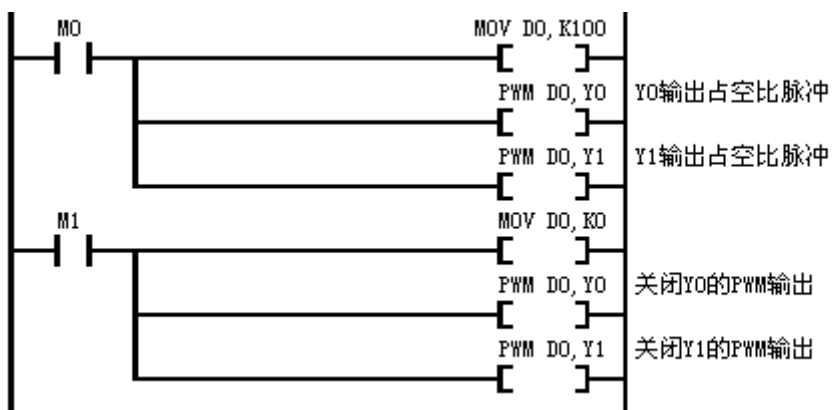
若该指令被接通，则把 IN1 的值送给 IN2 的 PWM 脉宽设定值寄存器，如果 IN1 的值为 0，则使 IN2 的 PWM 停止，但不影响 PWM 的周期定时器；如果 IN1 的值为负数，则使 IN2 的 PWM 停止，并使 PWM 的周期定时器复位。

若该指令被断开，则不执行任何操作。

PWM 脉宽（ON 的宽度）为： $\text{IN1 值} \times \text{PWM 时钟周期} \times 2$

若 PWM 时钟为系统时钟时，PWM 时钟周期为 0.25us。

注：当 PWM 的周期改变后，脉宽也可能被改变，因此必须重新设置脉宽。



脉宽调制输出指令的应用

● PLS IN1, IN2

指令步数：1 步或 2 步。

操作数	可使用的元件
IN1	K、D、RM、RY、RX、RC、RT、DM、变址寻址
IN2	Y0、Y1

说明：

PWM 脉冲周期设置指令。

若该指令被接通，则把 IN1 的低字节值送给 PWM 的周期设定值寄存器，IN1 的低字节值的范围为 1~128。IN1 的高字节指定 PWM 的时钟源，

为 0 则 PWM 时钟为系统时钟，PWM 时钟周期为 0.25us；不为 0 则 PWM 时钟周期为 **IN1 高字节值**×0.25us，此时 IN1 高字节值范围为 1~32。当程序开始运行时，PWM 周期的默认值为 128，时钟为系统时钟。Y0 和 Y1 的 PWM 时钟源要设置的相同。

若 Y0 和 Y1 其中一个用于 PTO 输出时，另一个可以用于 PWM 输出，但是其时钟只能为 PTO 时钟。

若该指令被断开，则不执行任何操作。

PWM 周期为：**IN1 低字节值** × 256 × PWM 时钟周期 × 2

注：若要使 PWM（PTO）输出有效，则 IN2 对应的映像寄存器中的位必须为 0，如果为 1，则使对应的端口固定为 ON 而屏蔽掉 PWM（PTO）的输出。

1.6.9 高速计数器及其高速输出指令（HCN/HCR/HCO）

- **HCN Cx, IN**

指令步数：2 步。

操作数	可使用的元件
Cx	C0、C1、C2
IN	常数、D、RM、RY、RX、RC、RT、DM

说明：

该指令为 16 位高速计数器线圈驱动指令。Cx 为指定的高速计数器（C0~C2），IN 为高速计数器的设定值（0~32767）。

若该指令被接通，则允许指定的高速计数器计数。若该指令被断开，不执行任何操作。当该指令被接通时，若所指定的高速计数器还没有装入设定值，则把 IN 的内容作为设定值装入该高速计数器的当前值寄存器。

- **HCR Cx**

指令步数：1 步。

该指令为 16 位高速计数器复位指令。Cx 为指定的高速计数器（C0~C2）。若该指令被接通，则指定的高速计数器被复位，高速计数器的触点被断开，当前值寄存器被清 0，高速计数器禁止计数及脉冲测量。若该指令被断开，则不执行任何操作。

- **HCO Cx, IN**

指令步数：2 步。

操作数	可使用的元件
Cx	C0、C1
IN	K、H、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

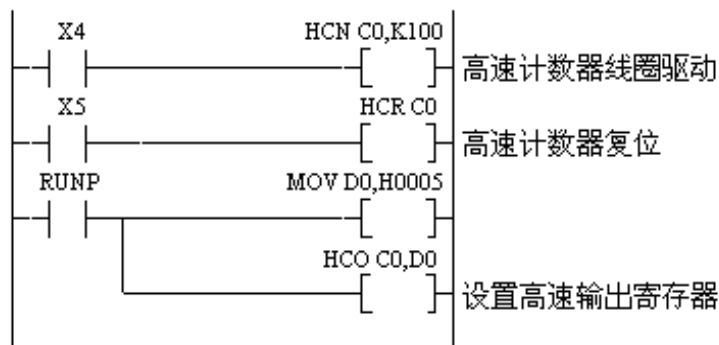
该指令为高速计数器的高速输出设置指令。Cx 为指定的高速计数器（C0~C1），IN 为指定的字元件。

YF0H 系列 PLC 为每个高速计数器分别设置了一个高速输出寄存器（单字节）。当某一高速计数器计数到预置值时，若其高速输出被允许（由特殊辅助继电器控制），则将其高速输出寄存器中的内容翻转相对应的输出继电器（Y0~Y7），高速输出寄存器中为 1 的位对应的输出继电器要取反，为 0 的位对应的输出继电器则不改变状态。高速输出寄存器的最低位

对应于 Y0，最高位对应于 Y7，其他依次类推。

例如：若高速输出寄存器的内容为 H03，则表示将把 Y0,Y1 取反，而 Y2~Y7 状态保持不变；若为 H9B，则表示将把 Y0,Y1,Y3,Y4,Y7 取反，而 Y2,Y5,Y6 状态保持不变。

若该指令被接通，则把 IN 的内容（仅低字节有效）送入所指定的高速计数器的高速输出寄存器中，此后，即使该指令再被断开，则高速输出寄存器中的内容仍有效。若该指令被断开，则不执行任何操作。



高速计数器和高速输出指令的使用

1.6.10 脉冲宽度、周期、频率测量指令（PWD/HDT/HDF）

- **PWD Cx**

指令步数：1 步。

该指令为脉冲宽度测量指令。用来测量指定的高速计数器（C0、C1）的输入端（X0、X1）上的脉冲宽度。

当该指令被断开时，将使所指定的高速计数器复位并禁止测量。若该指令被接通，将启动测量，用 1ms 的时钟来测量一个完整的脉冲宽度（由 OFF→ON 的正跳变到 ON→OFF 的负跳变之间的时间），当完成测量后，高速计数器的触点被接通，高速计数器当前值寄存器的值即为脉冲宽度（1ms 为单位），同时，停止继续测量，直到高速计数器被复位为止。测量脉冲宽度的范围为 0~32.767S。

- **HDT Cx**

指令步数：1 步。

该指令为脉冲周期测量指令。用来测量指定的高速计数器（C0、C1）的输入端（X0、X1）上的脉冲周期。

当该指令被断开时，将使所指定的高速计数器复位并禁止测量。若该指

令被接通，将启动测量，用 1ms 的时钟来测量一个完整的脉冲周期（由 ON→OFF 的负跳变到下一个 ON→OFF 的负跳变之间的时间），当完成测量后，高速计数器的触点被接通，高速计数器当前值寄存器的值即为脉冲周期（1ms 为单位），同时，停止继续测量，直到高速计数器被复位为止。测量脉冲周期的范围为 0~32.767S。

● HDF Cx

指令步数：1 步。

该指令为脉冲频率测量指令。用来测量指定的高速计数器（C0~C2）的输入端（X0~X2）上的脉冲频率。

（1）高速 C0、C1 测频率说明

① 高速 C0、C1 单相只测频率

当该指令被断开时，将使所指定的高速计数器复位并禁止测量。若该指令被接通，将启动测量，用由特殊继电器 DF2M、DF2M1 设定的闸门时间来测量频率。高速 C0、C1 完成测量后，其触点被接通，其当前值寄存器的值即为该闸门时间内的脉冲个数，同时，停止继续测量，直到高速计数器被复位为止。单相只测频率时不能进行高速计数。

② 高速 C0、C1 进行 32 位计数并测频率

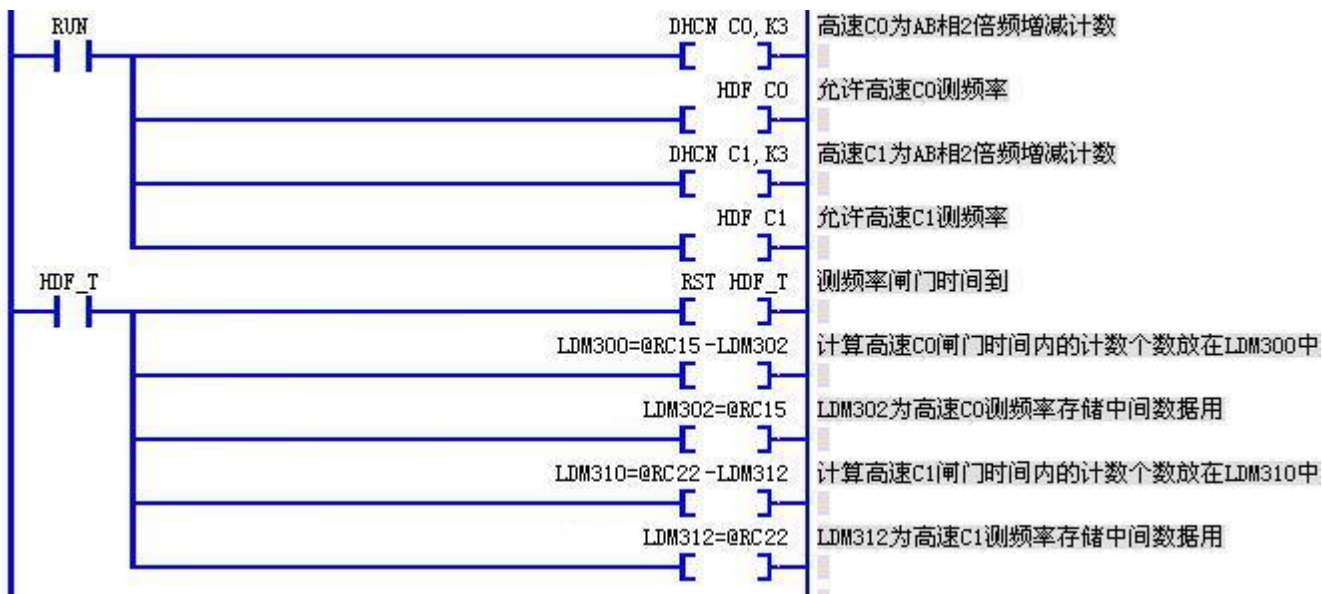
高速 C0、C1 在进行 32 位计数的同时测频率。高速 C0、C1 必须工作于 32 位 2 倍频或 4 倍频计数模式（单相或 AB 相），且不能到预置值自动复位。**在测频率时用户不能设定高速计数器的当前值。**

若该指令被接通，将允许测量。若该指令被断开，将禁止测量。

测频率时每隔一个闸门时间将刷新一次测量值（该闸门时间内的计数个数），该闸门时间由特殊继电器 DF2M、DF2M1 设定（00 为 500ms，01 为 1s，10 为 50ms，11 为 100ms）或者由指令“SFRWR 闸门时间值，HDF_TV”设定（闸门时间值单位为 10ms，范围 0~255）。每隔一个闸门时间特殊继电器 HDF_T 都会被接通，此时可以读取测量值，HDF_T 由用户来复位。

用户程序的扫描周期不能大于测频率的闸门时间，否则测量结果可能会不正确。

32 位高速计数并测频率例子如下：



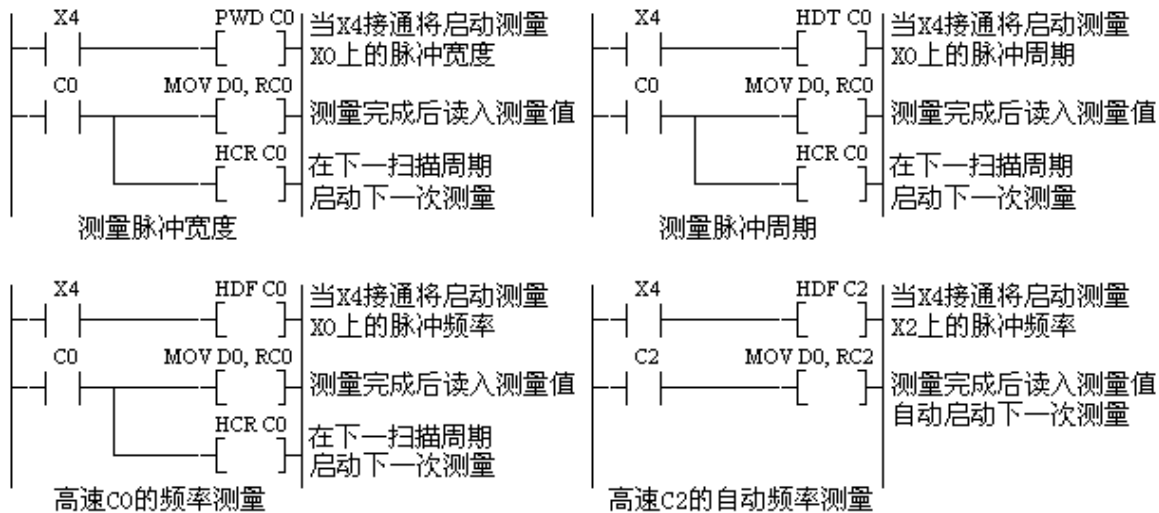
(2) 高速 C2 测频率说明

高速 C2 为自动测量，被接通后每隔一个闸门时间将刷新一次频率测量值，其映像寄存器 RC2 的值即为该闸门时间内的脉冲个数。

对于使用高速 C2 测频率，有以下几点需注意：

- ① 高速 C2 被启动测频率后(该指令被接通),当第 1 个测量结果完成后, C2 的触点被接通,在以后的自动测量过程中,该触点一直保持接通,直至高速 C2 被复位为止。若第一个测量结果还没有完成,则 C2 的触点处于断开状态。
- ② HC2F 也可用来判断测量结果是否完成。在每次测量结果完成（即刷新频率测量值）时 HC2F 都会被接通,HC2F 由用户来复位,而不必复位高速计数器。
- ③ DF2M、DF2M1 为测频率闸门时间控制继电器, HDF_TV 为设置测频率闸门时间值寄存器（使用 SFRWR 指令）,若闸门时间在 C2 处于自动测量过程中改变状态,则不能保证在改变状态后的第 1 个测量结果是正确的,除非在状态改变后立即使高速 C2 至少被复位一次。
- ④ 若 HCR C2 指令被接通,则高速 C2 复位,且禁止测量,直至执行 HDF C2 指令且该指令被接通才启动测量频率。

脉冲宽度、周期、频率测量的例子如下：



单相脉冲测量指令的使用

1.6.11 I/O 刷新指令（REF/REFL/REFH）

- **REF OUT**

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、变址寻址

该指令为输入输出继电器字刷新指令。若操作数 OUT 位于 X 继电器区，则当该指令被接通时，将从输入端口读入 OUT 所对应的输入继电器的最新输入信息。若 OUT 位于 Y 继电器区，则当该指令被接通时，将把 OUT 中的内容输出到所对应的输出继电器的输出端口上。若该指令被断开或 OUT 位于其他区，则不执行任何操作。

- **REFL OUT**

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、变址寻址

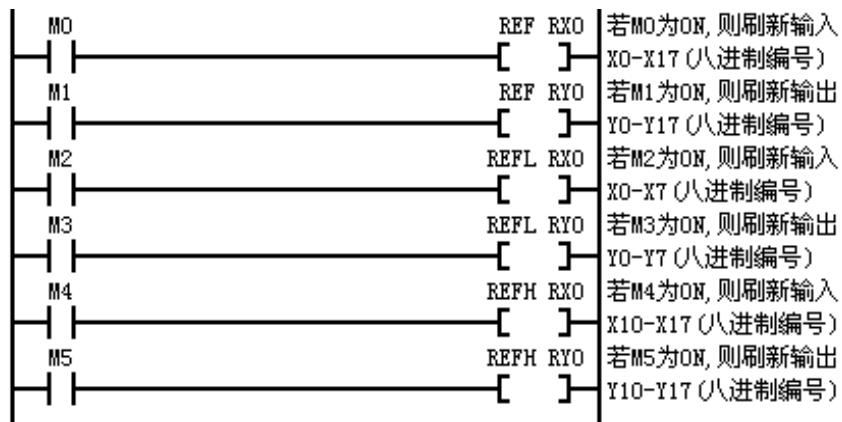
该指令为输入输出继电器低字节刷新指令。若操作数 OUT 位于 X 继电器区，则当该指令被接通时，将从输入端口读入 OUT 的低字节所对应的输入继电器的最新输入信息。若 OUT 位于 Y 继电器区，则当该指令被接通时，将把 OUT 的低字节中的内容输出到所对应的输出继电器的输出端口上。若该指令被断开或 OUT 位于其他区，则不执行任何操作。

● **REFH OUT**

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、变址寻址

该指令为输入输出继电器高字节刷新指令。若操作数 OUT 位于 X 继电器区，则当该指令被接通时，将从输入端口读入 OUT 的高字节所对应的输入继电器的最新输入信息。若 OUT 位于 Y 继电器区，则当该指令被接通时，将把 OUT 的高字节中的内容输出到所对应的输出继电器的输出端口上。若该指令被断开或 OUT 位于其他区，则不执行任何操作。



I/O 刷新指令的使用

1.6.12 查表和表格数据定义指令 (LDTAB/DW)

● **LDTAB OUT, 表格标号[D0]**

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT

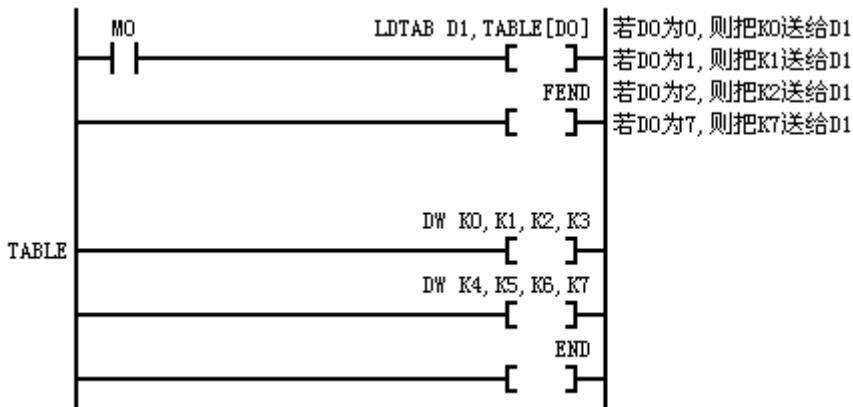
若该指令被接通，则以指定的表格标号所在的位置为基址加上 D0 中的偏移量作为新的位置（以字为单位），把该位置处的内容送给 OUT。若该指令被断开，则不执行任何操作。

● **DW 数据 1, 数据 2, …, 数据 n**

指令步数：n 步（若数据为字符串，则字符串中的每个字符占用 1 步）。

该指令用于定义表格中的数据。数据若为十进制整数 K，如果在-32768～32767 之间并且无后缀 L 则占用 1 步，之外或者有后缀 L 则占用 2 步；若为浮点数 K 则占用 2 步；若为十六进制数 H，如果小于 8 位则占用 1

步，如果为 8 位则占用 2 步；若为字符串，则该字符串中的每个字符占用 1 步。



LDTAB 和 DW 指令的使用

1.6.13 非易失性数据存储读写指令（RAMRD/RAMWR）

- **RAMRD OUT, IN1, N**

指令步数：2～4 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT
N	常数、D、RM、RY、RX、RC、RT

注：当 N 为寄存器时，IN 也必须为寄存器。

说明：

N 为可选参数，取值范围为 1～16，省略表示为 1。

若该指令被接通，则从非易失性数据存储中读入 N 个字的数据块放入从 OUT 开始的 N 个字中，IN 为该非易失性数据存储块的开始地址。

若该指令被断开，则不执行任何操作。

- **RAMWR IN1, IN2, N**

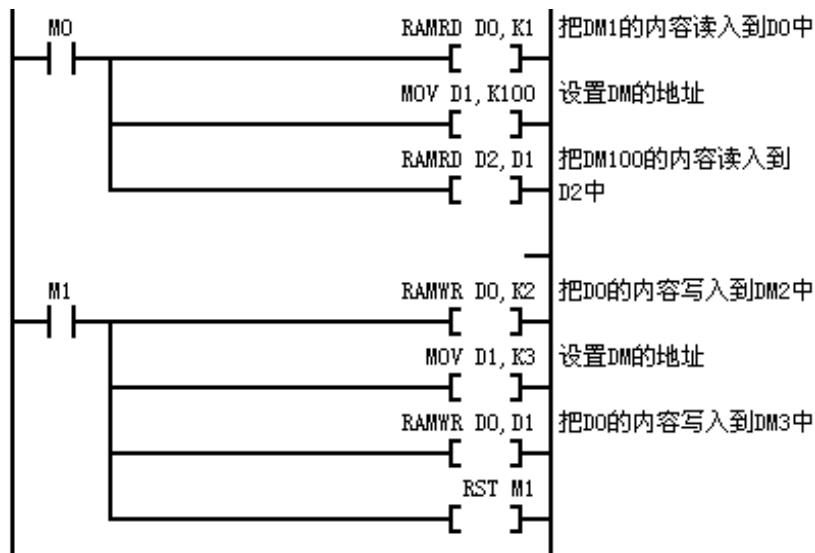
指令步数：2～4 步。

操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM、变址寻址
IN2	常数、D、RM、RY、RX、RC、RT
N	常数、D、RM、RY、RX、RC、RT

注：当 N 为寄存器时，IN2 也必须为寄存器。

说明：

N 为可选参数，取值范围 YF0A 为 1~8、YF0H 为 1~16，省略表示为 1。
 若该指令被接通，则把从 IN1 开始的 N 个字的内容写入到非易失性数据存储块中，IN2 为该非易失性数据存储块的开始地址（当 N>1 时，对于 YF0A 该地址必须是 8 的倍数，对于 YF0H 该地址必须是 16 的倍数）。
 若该指令被断开，则不执行任何操作。



RAMRD/RAMWR 指令的使用

1.6.14 进栈和出栈指令（PUSH/POP）

- **PUSH IN, Kx**

指令步数：2 步。

操作数	可使用的元件
IN	D、RM、RY、RX、RC、RT、DM256~DM383
Kx	常数（K0~K15）

说明：

若该指令被接通，则把从 IN 开始的 Kx+1 个元件的内容按后进先出原则压入 PLC 的数据堆栈空间，即 IN[0]先进栈，然后 IN[1]进栈，…，最后 IN[x]进栈。若该指令被断开，则不执行任何操作。

若 IN 为字元件，则每次进栈的数据为 2 个字节。

● POP OUT, Kx

指令步数：2 步。

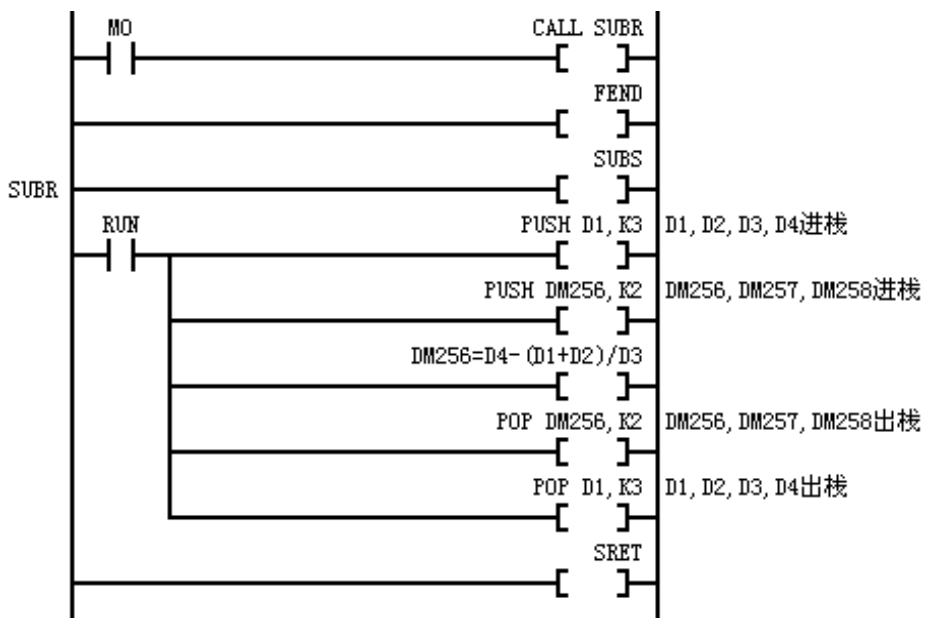
操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM256~DM383
Kx	常数（K0~K15）

说明：

若该指令被接通，则按后进先出原则从 PLC 的数据堆栈空间中弹出 Kx +1 个数据送入到从 OUT 开始的 Kx+1 个元件中，即先弹出 1 个数据送给 OUT[x]，然后再弹出 1 个数据送给 OUT[x-1]，…，最后再弹出 1 个数据送给 OUT[0]。若该指令被断开，则不执行任何操作。

若 OUT 为字元件，则每次弹出的数据为 2 个字节。

注：在一个函数的内部禁止使用这两条指令。



PUSH/POP 指令的使用

1.6.15 数据传送指令（MOV/LMOV/HMOV）

● MOV OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址、RHC2
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址、RHC2

说明：

若该指令被接通，则把 IN 的内容传送到 OUT 中。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=H1234，IN 为 D1=H4321，则执行该指令后 D0=H4321，D1=H4321。

● LMOV OUT, IN

指令步数：3 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 IN 低字节的内容传送到 OUT 低字节中。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=H1234，IN 为 D1=H4321，则执行该指令后 D0=H1221，D1=H4321。

● HMOV OUT, IN

指令步数：3 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 IN 高字节的内容传送到 OUT 高字节中。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=H1234，IN 为 D1=H4321，则执行该指令后 D0=H4334，D1=H4321。

1.6.16 字交换指令（XCH）

- **XCH OUT, IN**

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址

说明：

若该指令被接通，则把 IN 的内容与 OUT 的内容相互交换。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=K123，IN 为 D1=K321，则执行该指令后 D0=K321，D1=K123。

1.6.17 半字交换指令（LXCH/HXCH/SWAP）

- **LXCH OUT, IN**

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 IN 的低字节的内容与 OUT 的低字节的内容相互交换，而 IN 和 OUT 的高字节保持不变。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=H1234，IN 为 D1=H4321，则执行该指令后 D0=H1221，D1=H4334。

- **HXCH OUT, IN**

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 IN 的高字节的内容与 OUT 的高字节的内容相互交换，而 IN 和 OUT 的低字节保持不变。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=H1234，IN 为 D1=H4321，则执行该指令后 D0=H4334，D1=H1221。

● SWAP OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 的高字节与低字节的内容相互交换。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=H1234，则执行该指令后 D0=H3412。

1.6.18 加法指令（ADD）

● ADD OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 OUT 的内容与 IN 的内容相加，其结果放在 OUT 中。若该指令被断开，则不执行任何操作。

若运算结果最高位有进位输出，则进位标记 CF 为 ON，否则 CF 为 OFF。

若运算结果大于 32767 或小于 -32768（代数运算），则溢出标记 OV 为 ON。

例：

若 OUT 为 D0=K5，IN 为 K-8，则执行该指令后 D0=K-3，CF=OFF。

若 OUT 为 D0=K5，IN 为 D1=K-4，则执行该指令后 D0=K1，CF=ON。

若 OUT 为 D0=K30000，IN 为 D1=K4000，则执行该指令后 D0=K-31536，CF=OFF，OV=ON。

1.6.19 减法指令（SUB）

● SUB OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 OUT 的内容减去 IN 的内容，其结果放在 OUT 中。若该指令被断开，则不执行任何操作。

若运算结果最高位有借位输出，则进位标记 CF 为 ON，否则 CF 为 OFF。

若运算结果大于 32767 或小于-32768（代数运算），则溢出标记 OV 为 ON。

例：

若 OUT 为 D0=K5，IN 为 K-8，则执行该指令后 D0=K13，CF=ON。

若 OUT 为 D0=K-5，IN 为 D1=K4，则执行该指令后 D0=K-9，CF=OFF。

若 OUT 为 D0=K30000，IN 为 D1=K-4000，则执行该指令后 D0=K-31536，CF=ON，OV=ON。

1.6.20 乘法指令（MUL）

● MUL OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 OUT 的内容（有符号整数）与 IN 的内容（有符号整数）相乘，其结果为 32 位有符号整数，高 16 位（包括符号位）放在 OUT 中，低 16 位放在 OUT[1]中。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D1=K5, IN 为 D0=K-8, 则执行该指令后 D1=K-1 (高 16 位, 十六进制为 HFFFF), D2=K-40 (低 16 位, 十六进制为 HFFD8)。

1.6.21 除法指令 (DIV)

- **DIV OUT, IN**

指令步数: 2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明:

OUT 为 32 位被除数的高 16 位, OUT[1] 为 32 位被除数的低 16 位, IN 为 16 位除数。

若该指令被接通, 则把 OUT、OUT[1] 组成的 32 位数 (有符号整数) 除以 IN 的内容 (有符号整数), 其结果商放在 OUT 中, 余数放在 OUT[1] 中。若该指令被断开, 则不执行任何操作。

若运算结果溢出 (商的绝对值 > 32767) 或除数为 0, 则溢出标记 OV 为 ON, 同时 OUT、OUT[1]、IN 保持运算前的值不变。

例:

若 OUT、OUT[1] 分别为 D1=HFFFE、D2=H1DC0 (对应的 32 位数为 K-123456), IN 为 D0=K123, 则执行该指令后 D1=K-1003 (商), D2=K-87 (余数)。

1.6.22 脉冲滤波定时器指令（TPF）

● TPF Timer, PV, Sel

指令步数：3 步

操作数	可使用的元件
Timer	T384～T4349、变址寻址
PV	常数、D、RM、RY、RX、RC、RT、DM
Sel	0.01S、0.1S、1MS、无该参数

说明：

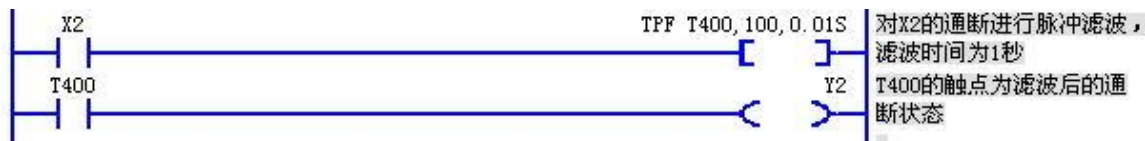
脉冲滤波定时器驱动指令

Timer 为指定的内部定时器(T384～T4349), PV 为该定时器的设定值(0～8000)。Sel 为可选参数，用于选择定时器的时基，若无该参数则时基由刷新 TDEC 指令周期决定。

功能：

对该指令的通断状态进行脉冲滤波，小于该定时器设定时间的通或断的窄脉冲被滤掉，该定时器触点即为滤波后的通断状态。

梯形图例子如下：



1.6.23 毫秒延时器指令（TMS）

● TMS Timer, PV

指令步数：3 步

操作数	可使用的元件
Timer	T384～T4349、变址寻址
PV	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

毫秒延时器驱动指令

Timer 为指定的内部定时器(T384～T4349), PV 为延时时间设定值(单位：毫秒，范围:0～16383)。

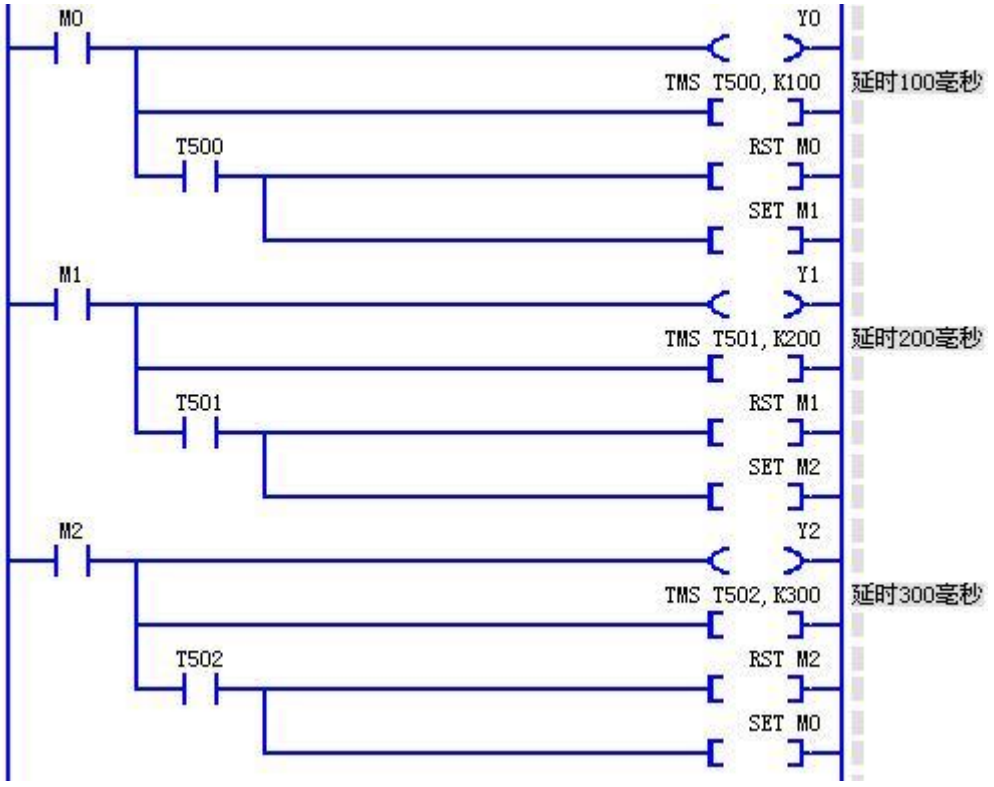
功能：

进行毫秒时间单位的延时。若该指令断开，则定时器触点复位并停止延

时；若该指令接通则开始延时，延时时间到后定时器触点接通并保持到该指令断开为止。

注：用于毫秒定时器的定时器只有触点，没有当前值寄存器。

梯形图例子如下：



1.6.24 可重触发脉冲定时器指令（TPR）

- TPR Timer, PV, CLK

指令步数：3 步

操作数	可使用的元件
Timer	T384~T4349、变址寻址
PV	常数、D、RM、RY、RX、RC、RT、DM
CLK	0.01S、0.1S、1MS、1S

说明：

脉冲滤波定时器驱动指令

Timer 为指定的内部定时器(T384~T4349)，PV 为该定时器的设定值（1MS 时基范围为 0~16383，其他时基范围为 0~8000）。CLK 为定时器的时基选择（1S，0.1S，0.01S，1MS）。

功能：

若指令输入发生上升沿跳变，则定时器触点接通，同时开始定时，定时时间到后定时器触点复位。在定时过程中，若指令输入又发生上升沿跳变，则会重新开始定时，而指令输入的其他状态都不会影响定时和触点。在定时器复位后(上电或执行定时器复位指令)，首次扫描到该指令时若指令输入为 ON，则也会启动脉冲定时。

例如：TPR T500, 1000, 1MS

1.6.25 增 1 和减 1 指令 (INC/DEC)

● INC OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、!LDM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 的内容加 1，不影响任何标记。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=K5，则执行该指令后 D0=K6。

若 OUT 为 D0=K-5，则执行该指令后 D0=K-4。

● DEC OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、!LDM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 的内容减 1，不影响任何标记。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=K5，则执行该指令后 D0=K4。

若 OUT 为 D0=K-5，则执行该指令后 D0=K-6。

1.6.26 数据移位指令（RLC/RRC/ROL/ROR/SHLA/SHRA/LSR）

● RLC OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 的内容和进位标记 CF 一起向左环移一位，OUT 的位 15 移入 CF，CF 移入 OUT 的位 0。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB															LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0

CF 为 ON

则执行该指令后

D0 的内容如下：

MSB															LSB
1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	1

CF 为 OFF

● RRC OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 的内容和进位标记 CF 一起向右环移一位，OUT 的位 0 移入 CF，CF 移入 OUT 的位 15。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB															LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0

CF 为 ON

则执行该指令后

D0 的内容如下：

MSB																LSB
	1	0	1	1	1	0	1	0	0	1	1	0	0	0	1	0

CF 为 OFF

● ROL OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为自循环左移指令，IN 为移位次数（字：0~15，双字：0~31）。

若该指令被接通，则由操作数 OUT 自成一个封闭的左移环移位 IN 次，每次 OUT 的最高位都移入位 0 中。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB																LSB
	0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0

IN 为 3

则执行该指令后

D0 的内容如下：

MSB																LSB
	1	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1

● ROR OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为自循环右移指令，IN 为移位次数（字：0~15，双字：0~31）。

若该指令被接通，则由操作数 OUT 自成一个封闭的右移环移位 IN 次，每次 OUT 的位 0 都移入最高位中。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB																LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	

IN 为 3

则执行该指令后

D0 的内容如下：

MSB																LSB
1	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	

● SHLA OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为算术左移指令，IN 为移位次数（字：0~15，双字：0~31）。

若该指令被接通，则把 OUT 中的内容向左移动 IN 位，其左端移出的位被丢弃，并在其右端补以相应位数的“0”。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB																LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	

IN 为 3

则执行该指令后

D0 的内容如下：

MSB																LSB
1	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	

● SHRA OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为算术右移指令，IN 为移位次数（字：0~15，双字：0~31）。

若该指令被接通，则把 OUT 中的内容向右移动 IN 位，其右端移出的位被丢弃，若原来数据的最高位为“0”，则在其左端补以相应位数的“0”，若原来数据的最高位为“1”，则在其左端补以相应位数的“1”（即保持原来的符号不变）。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB																LSB
1	0	1	1	0	1	0	0	1	1	0	0	0	1	0	0	

IN 为 3

则执行该指令后

D0 的内容如下：

MSB																LSB
1	1	1	1	0	1	1	0	1	0	0	1	1	0	0	0	

● LSR OUT, IN

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为逻辑右移指令，IN 为移位次数（字：0~15，双字：0~31）。

若该指令被接通，则把 OUT 中的内容向右移动 IN 位，其右端移出的位被丢弃，其左端补以相应位数的“0”。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB															LSB
1	0	1	1	0	1	0	0	1	1	0	0	0	1	0	0

IN 为 3

则执行该指令后

D0 的内容如下：

MSB															LSB
0	0	0	1	0	1	1	0	1	0	0	1	1	0	0	0

1.6.27 逻辑与指令（AND）

- **AND OUT, IN**

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 OUT 的内容与 IN 的内容之间以位为基础进行逻辑与操作，其结果放在 OUT 中。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB															LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0

IN 为 D1 的内容如下：

MSB															LSB
1	0	1	1	0	0	0	1	1	1	0	1	0	1	1	0

则执行该指令后

D0 的内容如下：

MSB															LSB
0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	0

1.6.28 逻辑或指令（OR）

- **OR OUT, IN**

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 OUT 的内容与 IN 的内容之间以位为基础进行逻辑或操作，其结果放在 OUT 中。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB															LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0

IN 为 D1 的内容如下：

MSB															LSB
1	0	1	1	0	0	0	1	1	1	0	1	0	1	1	0

则执行该指令后

D0 的内容如下：

MSB															LSB
1	1	1	1	0	1	0	1	1	1	0	1	0	1	1	0

1.6.29 逻辑异或指令（XOR）

- **XOR OUT, IN**

指令步数：2 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 OUT 的内容与 IN 的内容之间以位为基础进行逻辑异或操作，其结果放在 OUT 中。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB																LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	

IN 为 D1 的内容如下：

MSB																LSB
1	0	1	1	0	0	0	1	1	1	0	1	0	1	1	0	

则执行该指令后

D0 的内容如下：

MSB																LSB
1	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	

1.6.30 数据取反指令（CPL）

- **CPL OUT**

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 的内容按位进行逻辑取反。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB																LSB
0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0	

则执行该指令后

D0 的内容如下：

MSB																LSB
1	0	0	0	1	0	1	1	0	0	1	1	1	0	1	1	

1.6.31 数据转换指令（BCD/BIN/NEG/ABS/ROT）

● BCD OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 中的二进制数转换为 BCD 数后放入 OUT 中。BCD 数的范围为 0000-9999，若转换数值小于 0 或大于 9999，则为溢出，溢出标记 OV 为 ON，否则 OV 为 OFF。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=K1234，则执行该指令后 D0=H1234，OV=OFF。

若 OUT 为 D0=K12345，则执行该指令后 D0=HEEEE，OV=ON。

● BIN OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 中的 BCD 数转换为二进制数后放入 OUT 中。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0=H1234，则执行该指令后 D0=K1234。

● NEG OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 中以原码表示的二进制数转换为系统所默认的二进制补码数后放入 OUT 中。若该指令被断开，则不执行任何

操作。

例：

OUT 为 D0 的内容如下：（原码数：-1220）

MSB																LSB
	1	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0

则执行该指令后

D0 的内容如下：（补码数：-1220）

MSB																LSB
	1	1	1	1	1	0	1	1	0	0	1	1	1	1	0	0

● ABS OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把操作数 OUT 中的二进制补码数取绝对值后放入 OUT 中。若原数值为-32768 则为溢出，溢出标记 OV 为 ON，否则 OV 为 OFF；若原数值为负数，则进位标记 CF 为 ON，否则 CF 为 OFF。若该指令被断开，则不执行任何操作。

例：

若 IN 为 D0=K5，则执行该指令后 D0=K5，CF=OFF，OV=OFF。

若 IN 为 D0=K-5，则执行该指令后 D0=K5，CF=ON，OV=OFF。

● ROT OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为数据反转指令。若该指令被接通，则操作数 OUT 中的各个位将进行以下交换：

位 15↔位 0，位 14↔位 1，位 13↔位 2，位 12↔位 3，

位 11↔位 4，位 10↔位 5，位 9↔位 6，位 8↔位 7。

即 OUT 的内容发生了以下转换（最高位变成最低位，……，最低位变成最高位）：

转换前：MSB LSB

x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

转换后：LSB MSB

若该指令被断开，则不执行任何操作。

例：

OUT 为 D0 的内容如下：

MSB LSB

0	1	1	1	0	1	0	0	1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

则执行该指令后

D0 的内容如下：

MSB LSB

0	0	1	0	0	0	1	1	0	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1.6.32 译码与编码指令（DECO/ENCO）

● DECO OUT, IN

指令步数：2 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则根据 IN 的低 4 位所表示的位号置 OUT 中相应的位为 1，其他位置 0。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0，IN 为 D1=K4

则执行该指令后

D0 中的内容如下：

MSB																LSB
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

● ENCO OUT, IN

指令步数：2 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则将 IN 中的最低有效位（值为 1）的位号（0-15）写入到 OUT 中，若 IN 中的各个位全为 0，则 OUT 为 16。若该指令被断开，则不执行任何操作。

例：

OUT 为 D0

IN 为 D1 的内容如下：

MSB																LSB
0	1	1	1	0	1	0	0	1	1	0	0	1	0	0	0	0

则执行该指令后，D0 中的内容为 K3

1.6.33 数据分离与组合指令（SPLIT/UNITE）

● SPLIT OUT, IN

指令步数：2 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 IN 中的内容以半字节为基础进行分离后传送到 OUT、OUT[1]、OUT[2]、OUT[3]中。其中 IN 的位 15～位 12 的值存入 OUT 中，IN 的位 11～位 8 的值存入 OUT[1]中，IN 的位 7～位 4 的值存入 OUT[2]中，IN 的位 3～位 0 的值存入 OUT[3]中。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D1，IN 为 D0=H5678，则执行该指令后 D1=H5，D2=H6，D3=H7，D4=H8。

● UNITE OUT, IN

指令步数：2 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

若该指令被接通，则把 IN、IN[1]、IN[2]、IN[3]的低 4 位的值组合为一个 16 位的值后传送到 OUT 中。其中 IN 的低 4 位为组合值的位 15～位 12，IN[1]的低 4 位为组合值的位 11～位 8，IN[2]的低 4 位为组合值的位 7～位 4，IN[3]的低 4 位为组合值的位 3～位 0。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0，IN 为 D1，D1=H5，D2=H6，D3=H7，D4=H8
则执行该指令后 D0=H5678。

1.6.34 ASCII 码转换指令（HTA/ATH）

● HTA OUT, IN

指令步数：2 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

十六进制数转换为 ASCII 码数指令。

若该指令被接通，则把 IN 中的内容以半字节为基础转换为 ASCII 码后传送到 OUT[0]、OUT[1]、OUT[2]、OUT[3]中。其中 IN 的位 15～位 12 的 ASCII 码存入 OUT[0]中，IN 的位 11～位 8 的 ASCII 码存入 OUT[1]中，IN 的位 7～位 4 的 ASCII 码存入 OUT[2]中，IN 的位 3～位 0 的 ASCII 码存入 OUT[3]中。

若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D1，IN 为 D0=H5678，则执行该指令后 D1=H35，D2=H36，D3=H37，D4=H38。

● ATH OUT, IN

指令步数：2 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

ASCII 码转换为十六进制数指令。

若该指令被接通，则把 IN[0]、IN[1]、IN[2]、IN[3]中存的 ASCII 码组合为一个 16 位的值后传送到 OUT 中。其中 IN[0]中的 ASCII 码为组合值的位 15～位 12，IN[1]中的 ASCII 码为组合值的位 11～位 8，IN[2]中的 ASCII 码为组合值的位 7～位 4，IN[3]中的 ASCII 码为组合值的位 3～位 0。

若 ASCII 码不为 0～9 或 A～F 范围内的字符则置溢出继电器 OV 为 1，若转换正确则溢出继电器 OV 为 0。

若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0，IN 为 D1，D1=H35，D2=H36，D3=H37，D4=H38

则执行该指令后 D0=H5678。

1.6.35 字和字节转换指令（WTOB/BTOW）

● WTOB OUT, IN

指令步数：3 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

字分离为字节指令。

若该指令被接通，则把 IN 的内容(字)分离为两个字节，低字节传送到 OUT[0]的低字节中，高字节传送到 OUT[1]的低字节中，OUT[0]和 OUT[1]的高字节都清 0。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0，IN 为 D2，D2=H1234，则执行该指令后 D0=H34，D1=H12。

● BTOW OUT, IN

指令步数：3 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

字节组合为字指令。

若该指令被接通，则把 IN[0]的低字节和 IN[1]的低字节组合为一个字传送到 OUT 中，其中 IN[0]的低字节为字的低字节，IN[1]的低字节为字的高字节。若该指令被断开，则不执行任何操作。

例：

若 OUT 为 D0，IN 为 D1，D1=H1234，D2=H5678，则执行该指令后 D0=H7834。

1.6.36 数据块操作指令（BMOV/ BRMOV/ BXMOV/FILL/BL0D/BCMP）

● BMOV OUT, IN, N

指令步数：4 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址
N	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

数据块传送指令

若该指令被接通，则把从 IN 开始的 N 个字传送到从 OUT 开始的 N 个字中。从数据块起始处开始传送。N 的取值范围为 0～64。

若该指令被断开，则不执行任何操作。

例：

BMOV DM300, DM400, K10 则当该指令被接通时，把从 DM400 开始的 10 个字的内容传送到从 DM300 开始的 10 个字中。

● BRMOV OUT, IN, N

指令步数：4 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址
N	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

数据块从右边开始传送指令

若该指令被接通，则把从 IN 开始的 N 个字传送到从 OUT 开始的 N 个字中。从数据块结尾处开始传送。N 的取值范围为 0～64。

若该指令被断开，则不执行任何操作。

例：

BRMOV DM300, DM400, K10 则当该指令被接通时，把从 DM400 开始的 10 个字的内容传送到从 DM300 开始的 10 个字中。

● **BXMOV OUT, IN, N**

指令步数：4 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	D、RM、RY、RX、RC、RT、DM、变址寻址
N	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

数据块高低字节交换传送指令

若该指令被接通，则把从 IN 开始的 N 个字的高低字节交换后传送到从 OUT 开始的 N 个字中。从数据块起始处开始传送。N 的取值范围为 0～64。

若该指令被断开，则不执行任何操作。

例：

BXMOV DM300, DM400, K10 则当该指令被接通时，把从 DM400 开始的 10 个字的高低字节交换后传送到从 DM300 开始的 10 个字中。

● **FILL OUT, IN, N**

指令步数：4 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	常数、D、RM、RY、RX、RC、RT、DM、变址寻址
N	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

数据块填充指令

若该指令被接通，则用 IN 的值填充到从 OUT 开始的 N 个字中。N 的取值范围为 0～64。

若该指令被断开，则不执行任何操作。

例：

FILL DM300, K1234, K10 则当该指令被接通时，把从 DM300 开始的 10 个字都设置为 1234 中。

● BLOD OUT, IN, N

指令步数：4 步

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址
IN	数据表、字符串、\$表格标号
N	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

数据块加载指令

若该指令被接通，则从 IN 指定的字符串或数据表中加载 N 个字到从 OUT 开始的 N 个字存储器中。N 的取值范围为 0~64。

若该指令被断开，则不执行任何操作。

数据表格式：把多个常数用{ }括起来，各个常数之间用逗号分隔。如下

{K1, K2, K3, K4, K5, K6, K7, K8}

{K1, H2EA, H0AAEE, K4, K5, K6, H1234, K8}

{‘a’, ‘b’, ‘c’, ‘d’, K5, ‘我’, K7, K8, K0}

{K1, K2, K3, “abcdefg 你好”, K0} 注：当数据表作为字符串使用时，必须要加入字符串结束标记 K0。

在数据表格式中，数据若为十进制整数 K 则如果在 -32768~32767 之间并且无后缀 L 则占用 1 步，之外或者有后缀 L 则占用 2 步；若为浮点数 K 则占用 2 步；若为十六进制数 H，如果小于 8 位则占用 1 步，如果为 8 位则占用 2 步；若为字符串，则该字符串中的每个字符占用 1 步。

例：

BLOD DM300,{K0,K1,K2,K3,K4,K5,K6,K7,K8,K9},K10 表示把从 DM300 开始的 10 个字存储器依次送入 K0~K9，即执行后 DM300 为 0、DM301 为 1、DM302 为 2、……、DM309 为 9。

BLOD DM300, “abcdefg”, K7 表示把从 DM300 开始的 7 个字存储器依次送入字符 a~g，即执行后 DM300 为’a’、DM301 为’b’、DM302 为’c’、……、DM306 为’g’。

BLOD DM300,\$TBL,K10 表示从表格 TBL（由 DW 指令所定义）中加载 10 个字的数据到 DM300 开始的 10 个字存储器中。

● BCMP IN1, IN2, N

指令步数：4 步

操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM、变址寻址
IN2	D、RM、RY、RX、RC、RT、DM、变址寻址
N	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

数据块比较指令

若该指令被接通，则把从 IN1 开始的 N 个字与从 IN2 开始的 N 个字进行比较，若全部相同，则特殊继电器 CF(M202)为 ON，否则 CF(M202)为 OFF。N 的取值范围为 0~64。

若该指令被断开，则不执行任何操作。

例：

BCMP DM300, DM400, K10 则当该指令被接通时，若 DM300~DM309 与 DM400~DM409 全部相同（10 个字），则特殊继电器 CF(M202)为 ON，不是全部相同，则特殊继电器 CF(M202)为 OFF。

1.6.37 数据保存指令（DSAVE）

- **DSAVE IN1, IN2, N**

指令步数：4 步

操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、DM
IN2	常数
N	常数

说明：

数据保存指令

在梯形图程序运行前，按 IN2 中指定的起始地址从非易失存储器中读入 N 个字的数据放入 IN1 中(上电加载)。在梯形图程序运行中，若该指令断开，则不执行任何操作，若该指令接通，则把从 IN1 开始的 N 个字的内容写入到 IN2 所指定的非易失存储器块中。

IN2 中的值（非易失存储器块的起始地址）必须为 16 的倍数。

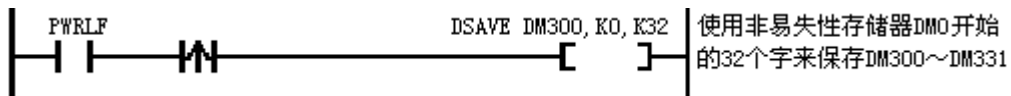
第 1 条 DSAVE 指令的 IN2 必须为 K0。

该指令可用于易失性存储器的掉电保持。

注：数据保存也可使用编程软件的掉电保持配置，当使用了编程软件的掉电保持配置后，不要再使用该指令。

例：

要使易失性存储器 DM300～DM331 掉电保持。其梯形图例子（PWRLF 为电源掉电标记）：



DSAVE 指令使用例子

1.6.38 扫描时间测量指令（SCANT）

● SCANT OUT1,OUT2

指令步数：4 步

操作数	可使用的元件
OUT1	DM
OUT2	DM

说明：

扫描时间测量指令

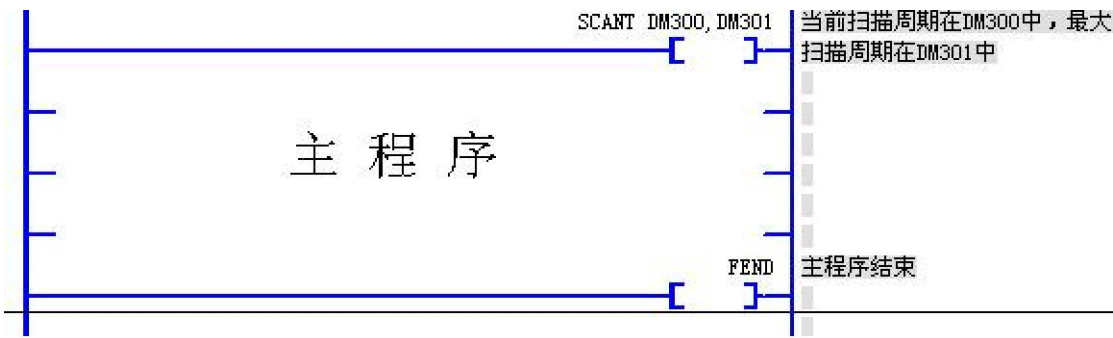
测量程序的扫描时间，并把当前扫描时间放在 OUT1 中，最大扫描时间放在 OUT2 中，时间单位为 ms。

注：该指令为无条件执行指令，可直接连接于左母线。

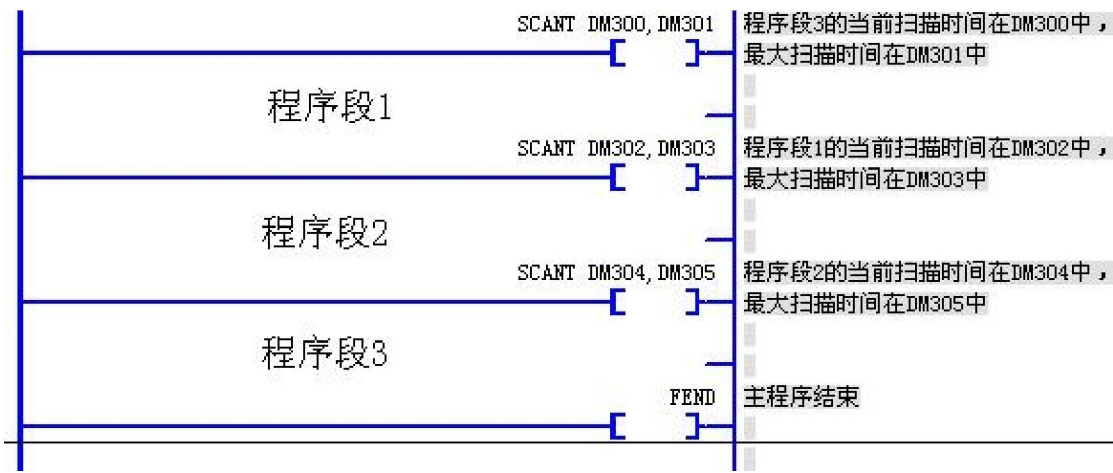
该指令可用来测量程序的扫描周期或某段程序的扫描时间。

例：

测量整个程序的扫描周期：



测量某个程序段的扫描时间：



1.6.39 SPI 接口指令（SSET/SCLR/SOUT/SIO/SOUB/SIOB/SPAO）

- **SSET Sx**

指令步数：1 步。

该指令为 SPI 接口控制线置“1”指令。若该指令被接通，则立即使指定的 SPI 接口控制线（S0 或 S1 或 S2）置“1”（高电平）。若该指令被断开，则不执行任何操作。

- **SCLR Sx**

指令步数：1 步。

该指令为 SPI 接口控制线清“0”指令。若该指令被接通，则立即使指定的 SPI 接口控制线（S0 或 S1 或 S2）清“0”（低电平）。若该指令被断开，则不执行任何操作。

- **SOUT OUT**

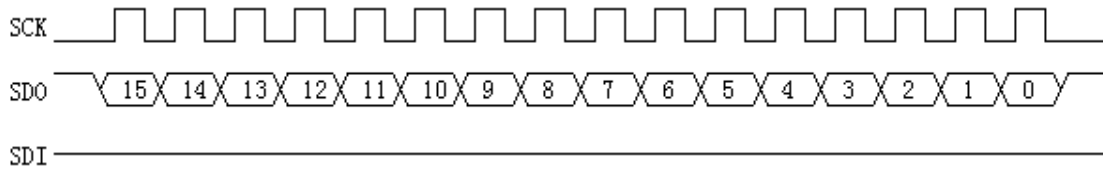
指令步数：1 步。

操作数	可使用的元件
OUT	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为 SPI 接口字移位输出指令。若该指令被接通，则把操作数 OUT 的内容从 SPI 接口的 SDO 引脚移位输出(一个字的数据)，高位在先，低位在后，但不接收数据，即 OUT 的内容保持不变。若该指令被断开，则不执行任何操作。

串行数据输出（SDO）和串行移位时钟（SCK）的时序关系如下：



SDO 和 SCK 的字输出时序图

- **SIO OUT**

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

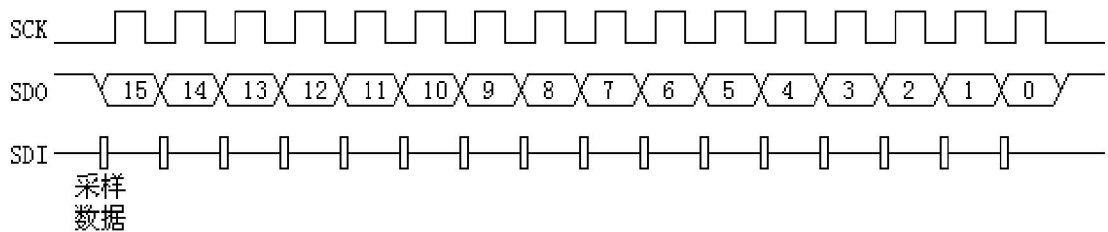
该指令为 SPI 接口字移位输入输出指令。若该指令被接通，则从 SPI 接口的 SDI 引脚移位输入一个字的串行数据放入 OUT 中，同时把 OUT 的原内容从 SPI 接口的 SDO 引脚移位输出，输入输出均是高位在先，低位在后。若该指令被断开，则不执行任何操作。

特殊辅助继电器 SPIM0、SPIM1 决定输入采样数据的方式：

SPIM1	SPIM0	输入采样数据和移位时钟的相位关系
0	0	方式 0：输入数据开始采样发生在时钟跳变前，且以后的采样均在时钟低电平。
0	1	方式 1：输入数据开始采样发生在时钟第 1 次上跳变后，且以后的采样均在时钟高电平。
1	0	方式 2：输入数据开始采样发生在时钟第 1 次上跳变和下跳变后，且以后的采样均在时钟低电平。

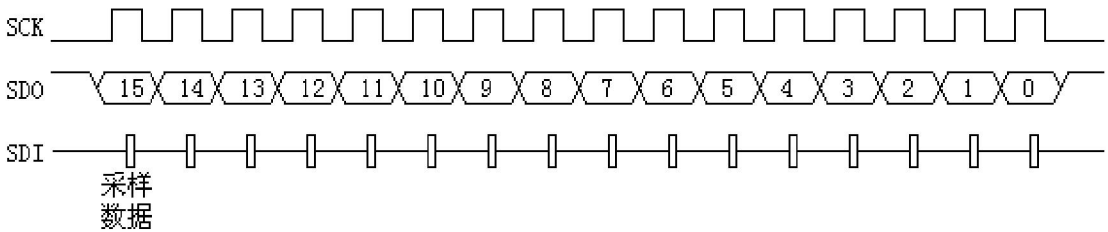
串行数据输入（SDI）、串行数据输出（SDO）和串行移位时钟（SCK）的时序关系如下：

SPIM1=0、SPIM0=0:



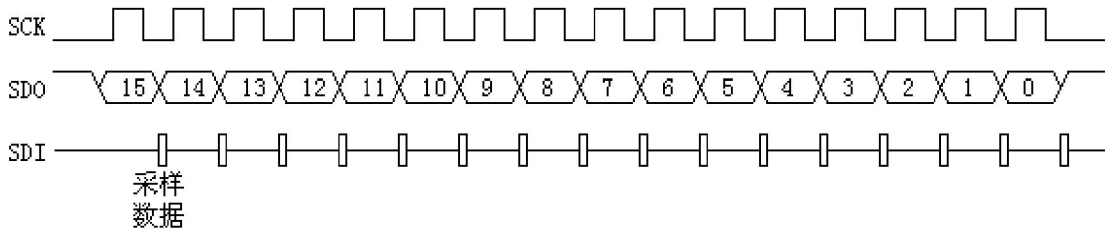
方式 0 的字输入输出时序图

SPIM1=0、SPIM0=1:



方式 1 的字输入输出时序图

SPIM1=1、SPIM0=0:



方式 2 的字输入输出时序图

● SOUB OUT

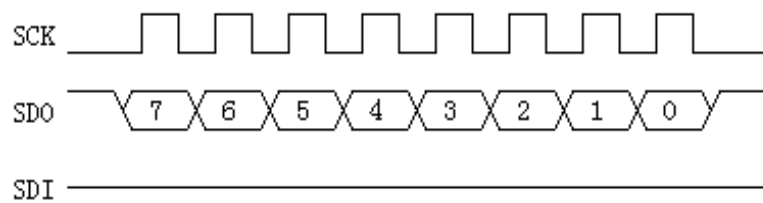
指令步数：1 步。

操作数	可使用的元件
OUT	常数、D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

该指令为 SPI 接口字节移位输出指令。若该指令被接通，则把 OUT 的低字节的内容从 SPI 接口的 SDO 引脚移位输出(一个字节的的数据)，高位在先，低位在后，但不接收数据，即 OUT 的内容保持不变。若该指令被断开，则不执行任何操作。

串行数据输出（SDO）和串行移位时钟（SCK）的时序关系如下：



SDO 和 SCK 的字节输出时序图

● SIOB OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、变址寻址

说明：

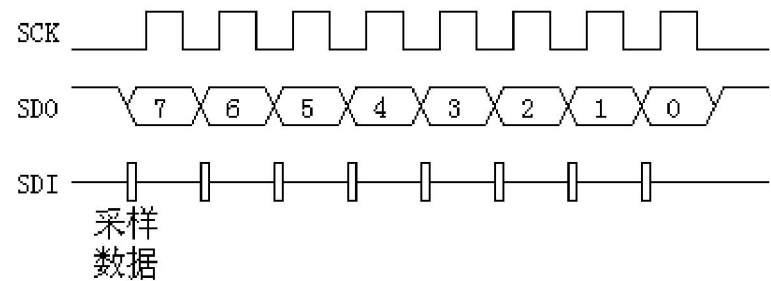
该指令为 SPI 接口字节移位输入输出指令。若该指令被接通，则从 SPI 接口的 SDI 引脚移位输入一个字节的串行数据放入 OUT 的低字节中，同时把 OUT 的低字节的原内容从 SPI 接口的 SDO 引脚移位输出，而 OUT 的高字节的内容保持不变，输入输出均是高位在先，低位在后。若该指令被断开，则不执行任何操作。

特殊辅助继电器 SPIM0、SPIM1 决定输入采样数据的方式：

SPIM1	SPIM0	输入采样数据和移位时钟的相位关系
0	0	方式 0：输入数据开始采样发生在时钟跳变前，且以后的采样均在时钟低电平。
0	1	方式 1：输入数据开始采样发生在时钟第 1 次上跳变后，且以后的采样均在时钟高电平。
1	0	方式 2：输入数据开始采样发生在时钟第 1 次上跳变和下跳变后，且以后的采样均在时钟低电平。

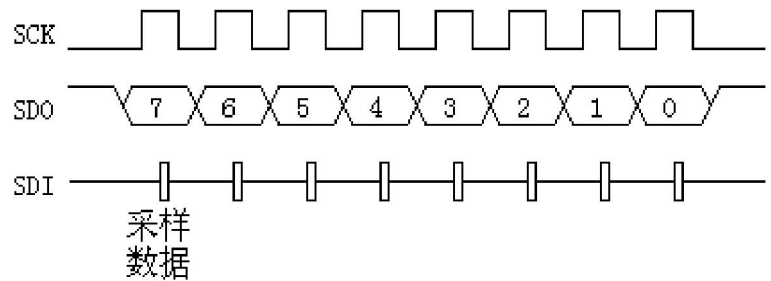
串行数据输入（SDI）、串行数据输出（SDO）和串行移位时钟（SCK）的时序关系如下：

SPIM1=0、SPIM0=0:



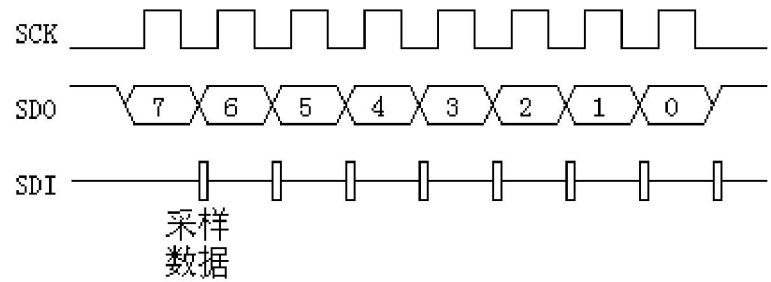
方式 0 的字节输入输出时序图

SPIM1=0、SPIM0=1:



方式 1 的字节输入输出时序图

SPIM1=1、SPIM0=0:



方式 2 的字节输入输出时序图

● SPAO Kx

指令步数：1 步。

该指令为串行外设地址输出指令。Kx 为外设地址（单字节）。

在基于 SPI 接口的多外围设备扩展中，可使用 SPI 接口的控制线 S2 作为设备的地址/数据控制信号。若 S2 为“1”（高电平），则 SPI 接口输出的为设备的地址信息，用来选通设备；若 S2 为“0”（低电平），则 SPI

接口输入、输出的为被选通设备的数据（命令）信息。

该指令即为完成上述的多外设扩展方式而设置的。

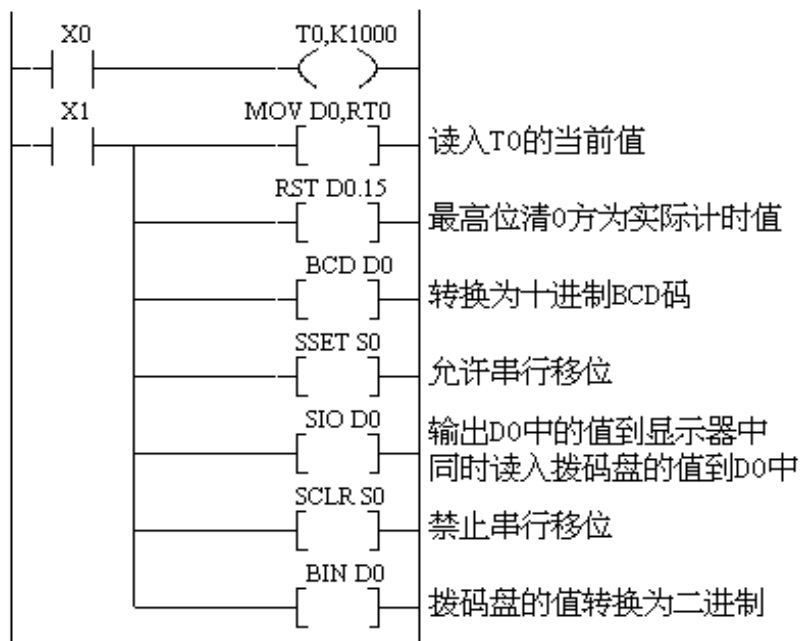
若该指令被接通，则执行以下动作：

- ① SPI 接口的控制线 S2 置“1”（高电平）。
- ② 把常数 Kx（1 个字节）作为地址信息从 SPI 接口移位输出，高位在先，低位在后。
- ③ SPI 接口的控制线 S2 清“0”（低电平）。

若该指令被断开，则不执行任何操作。

SPI 接口的应用实例：

下面为用 SPI 接口扩展 4 位拨码盘输入和 4 位 LED 数码显示的实例。该实例具有电路简单，成本极低，外形和尺寸可由用户自由设计等优点。



SPI 接口的应用实例梯形图

1.6.40 IIC 接口指令（IICSTART/IICSTOP/IICW/IICRACK/IICRACK OUT）

● IICSTART

指令步数：1 步。

该指令为 IIC 接口产生开始信号指令。若该指令被接通，则立即产生一个 IIC 开始信号。若该指令被断开，则不执行任何操作。

● IICSTOP

指令步数：1 步。

该指令为 IIC 接口产生停止信号指令。若该指令被接通，则立即产生一个 IIC 停止信号。若该指令被断开，则不执行任何操作。

● IICW IN

指令步数：1 步。

操作数	可使用的元件
IN	D

说明：

该指令为 IIC 接口写数据指令。若该指令被接通，则把数据寄存器 D 中低字节数据写给从机，并把从机应答信号的状态放在进位继电器 CF 中。若该指令被断开，则不执行任何操作。

● IICRACK OUT

指令步数：1 步。

操作数	可使用的元件
OUT	D

说明：

该指令为 IIC 接口应答读数据指令。若该指令被接通，则从从机中读 1 字节数据放在数据寄存器 D 的低字节中并产生 1 个应答(ACK)信号，数据寄存器 D 的高字节保持不变。若该指令被断开，则不执行任何操作。

● IICRACK OUT

指令步数：1 步。

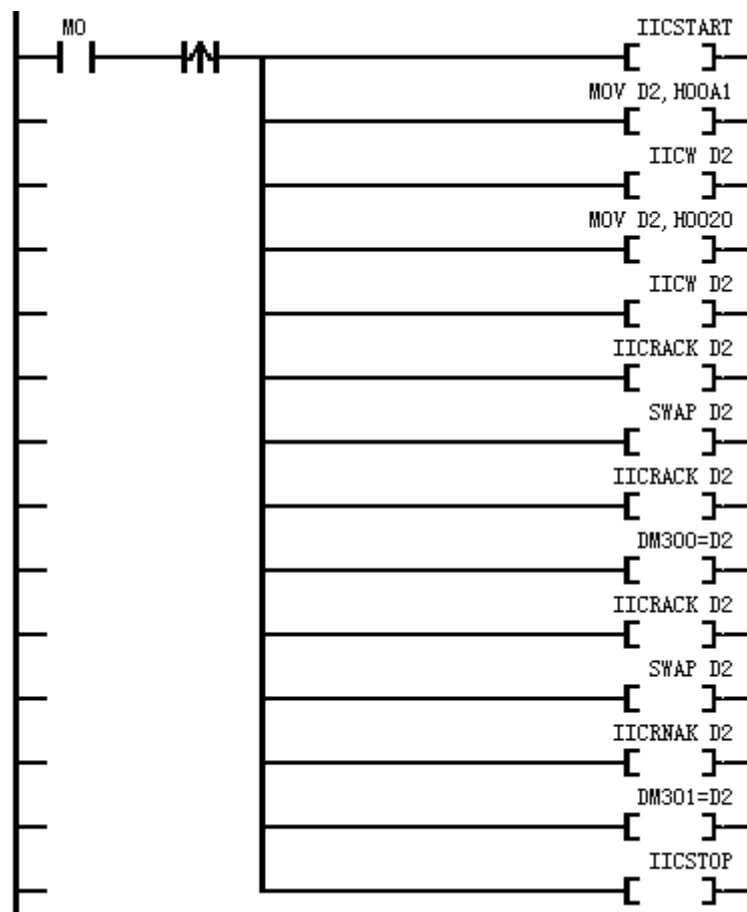
操作数	可使用的元件
OUT	D

说明：

该指令为 IIC 接口无应答读数据指令。若该指令被接通，则从从机中读 1 字节数据放在数据寄存器 D 的低字节中并产生 1 个无应答(NAK)信号，数据寄存器 D 的高字节保持不变。若该指令被断开，则不执行任何操作。

注：该 IIC 接口不支持时钟同步功能。时钟速率为 400KHz。

IIC 接口指令的梯形图例子如下:



1.7 宏指令

宏指令是为了增加程序的可读性、可维护性和可兼容性或者用来增强程序的功能等而设置的一类指令。它们在编译时不产生代码，或者由若干个内部指令组成。例如：在程序设计过程中，对于经常使用的一些常数，如果将它们直接写到程序中去，一旦常数的数值发生变化，就必须逐个找出程序中的所有对应的常数，逐一进行修改，这样必然会降低程序的可维护性和可靠性。因此为了便于对整个程序进行修改维护，或为了提高程序的可读性，可采用预定义指令“EQU”来使用一些有一定意义的字符串定义常数或变量，这些“EQU”指令可集中放在程序的开始处，或专门形成一个文件通过文件包含指令“INCLUDE”来调用，当需要修改程序中的某个常数，可以不必修改整个程序，而只要修改一下相应常数的预定义指令即可，大大提高了编程效率。

1.7.1 预定义指令（EQU/AS）

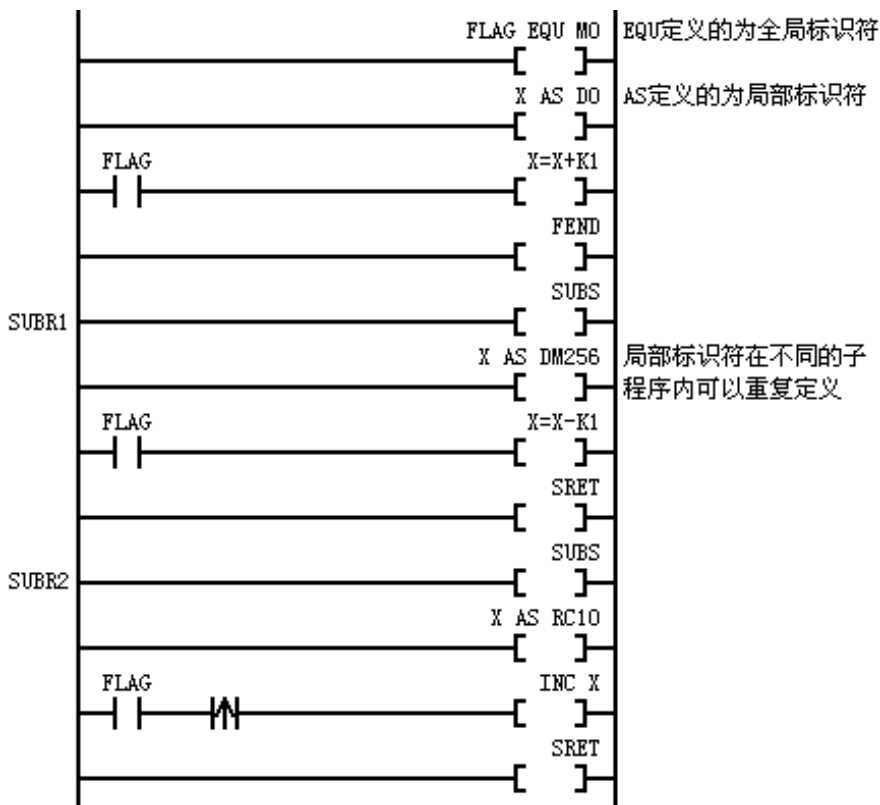
EQU/AS 指令的作用是用一个指定的标识符来定义一个常数或变量（元件），然后在程序中就可以使用该标识符来代替所定义的常数或变量。指令格式如下：

标识符 EQU 常数或变量

标识符 AS 常数或变量

EQU 和 AS 的区别是：EQU 定义的标识符为全局标识符，在整个程序中都有效，并且不可以重复定义；AS 定义的标识符为局部标识符，只在其所在的子程序或函数或主程序内有效，即在主程序内用 AS 定义的标识符只在主程序内有效，在某一子程序或函数内用 AS 定义的标识符只在该子程序或函数内有效。在同一个子程序或函数或主程序内用 AS 定义的标识符不可以重复定义，但在不同的子程序或函数内可以重复定义即用 AS 定义的标识符可以相同。

标识符应先定义，后使用。



EQU/AS 指令的使用

1.7.2 预定义文件包含指令（INCLUDE）

INCLUDE 指令的作用是加载一个由预定义指令(EQU/AS)组成的文本文件。用户可以把一些预定义指令放在一个文本文件中，在程序中可通过 INCLUDE 指令把这些预定义指令包含到程序中。该文本文件必须符合语句表的格式。

INCLUDE 指令的格式如下：

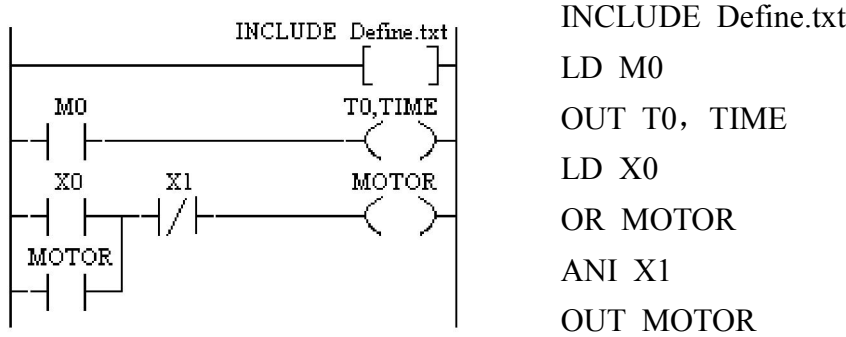
INCLUDE 文件名

文件名不能带有路径，其路径为该指令所在程序的路径，即 INCLUDE 指令包含的文件和 INCLUDE 指令所在的程序必须在同一个文件夹中。

例如：文本文件 Define.TXT 的内容如下：

```
;YF0A 的预定义文件
TIME EQU K10    ;定时时间
MOTOR EQU Y0    ;控制电机
```

在程序中被包含如下：

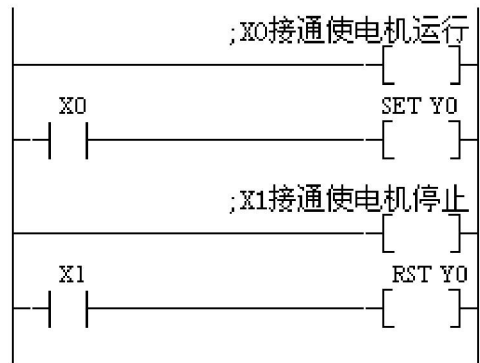


INCLUDE 指令的使用

1.7.3 注释指令 (;)

用户可以使用指令线圈在程序中放置注释。其方法是使用注释指令：分号“;”+注释内容。其格式如下：

;注释内容



注释指令的使用

★注：在一个指令或表达式前面加分号“;”，则使该指令或表达式被注释掉，即该指令或表达式无效，该方法在程序调试时特别有用。

1.7.4 程序连接指令（LINK）

LINK 指令用于在梯形图中连接一个由语句表组成的程序文件。其格式如下：

LINK 文件名

文件名不能带有路径，其路径为该指令所在程序的路径，即 LINK 指令连接的文件和 LINK 指令所在的程序必须在同一个文件夹中。

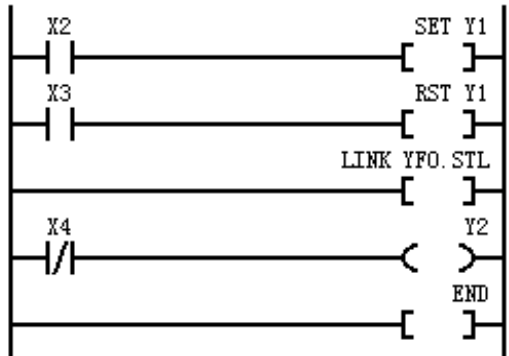
程序在编译时，将在该指令处插入该文件中的程序来一起编译。被连接的文件中不许再有 LINK 指令。

例如：

语句表文件 YF0.STL 的内容如下：

<pre> ;LINK 指令举例 LD X0 OR Y0 ANI X1 OUT Y0 </pre>

某梯形图如下：



则该梯形图等同于下面的程序：

```

LD X2
SET Y1
LD X3
RST Y1
LD X0      ;LINK 指令的内容（开始）
OR Y0
ANI X1
OUT Y0     ;LINK 指令的内容（结束）
LDI X4
OUT Y2
END
    
```

1.7.5 取标号的步数 (\$)

该符号的作用是取出某一标号在程序中所处的步数（地址）作为某一指令或表达式的常数操作数。其格式为：

\$标号

例如若标号 TABLE 在程序中所处的步数为第 500 步，则指令：

MOV D0, \$TABLE 等同于指令： MOV D0, K500

注：标号 TABLE 为系统保留标号，为程序中的所有表格标号的公共参考点，所有的表格标号都可以以该标号的地址为参考地址。

1.7.6 取元件的编号或变量的地址 (#)

该符号的作用是把某一元件的编号（地址）作为某一指令或表达式的常数操作数。其格式为：

#元件（或标识符）

例如

指令：MOV D0, #DM256 等同于指令：MOV D0, K256

指令：RAMRD D0, #DM100 等同于指令：RAMRD D0, K100

程序：

VAR1 EQU DM300

MOV D0, #VAR1 ； 取变量 VAR1 的地址（D0 = 300）

DM0[D0] = 0 ； 等同于 DM300 = 0

DM1[D0] = 1 ； 等同于 DM301 = 1

DM2[D0] = 2 ； 等同于 DM302 = 2

DM3[D0] = 3 ； 等同于 DM303 = 3

注 1：元件或标识符可以是偏移量为常数的变址寻址格式，例如#DM300[10]、#VAR1[5]、#DM8[#VAR1]，得到的地址常数为基址+偏移量，但不能是偏移量为寄存器的变址寻址格式，例如#DM300[D0]是错误的，得不到正确的地址常数，此时应使用表达式“#DM300+D0”计算地址。

注 2：该符号也可取 DM 存储器的高字节或低字节的字节地址，取高字节的字节地址格式为#DMHx，取低字节的字节地址格式为#DMLx，x 为 DM 存储器

编号，高字节字节地址为 $2 \times x + 1$ ，低字节字节地址为 $2 \times x$ ，例如#DMH300，#DML500，#DMH5[#VAR1]都是正确的格式，而#DMH5[D0]是错误的，此时应使用表达式“ $2 \times D0 + \#DMH5$ ”计算字节地址。

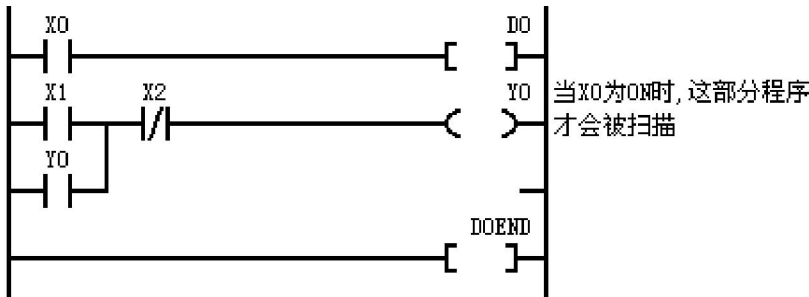
1.8 结构化的程序流控制指令

1.8.1 DO/DOEND 指令

若 DO 指令被接通，则 DO 和与之配对的 DOEND 之间的指令将被扫描；若 DO 指令被断开，则 DO 和与之配对的 DOEND 之间的指令将跳过不被扫描。

若 DO 指令被接通，则执行 DOEND 指令时使逻辑运算结果为 ON；若 DO 指令被断开，则执行 DOEND 指令时使逻辑运算结果为 OFF。

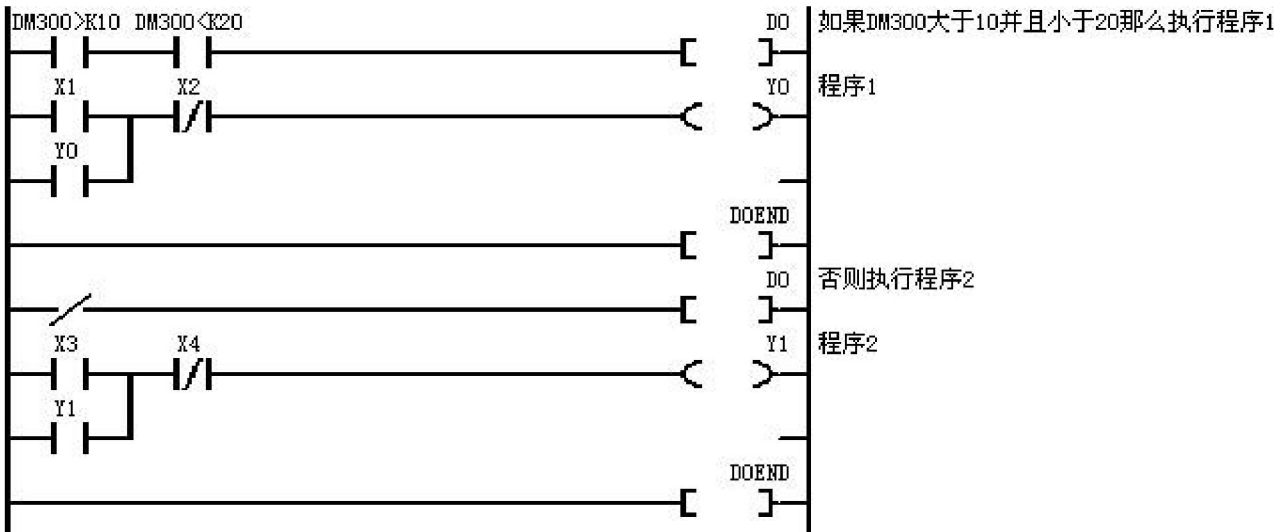
DO/DOEND 指令可以嵌套，且使用次数不限，但应成对使用。



DO/DOEND 指令的使用

注：若 DOEND 指令没有直接连接于左母线，则 DOEND 指令和其配对的 DO 指令必须直接连接于同一垂直连线。

如果条件成立那么执行程序 1、否则执行程序 2 的程序结构例子：



IF ... THEN ... ELSE ... 程序结构例子

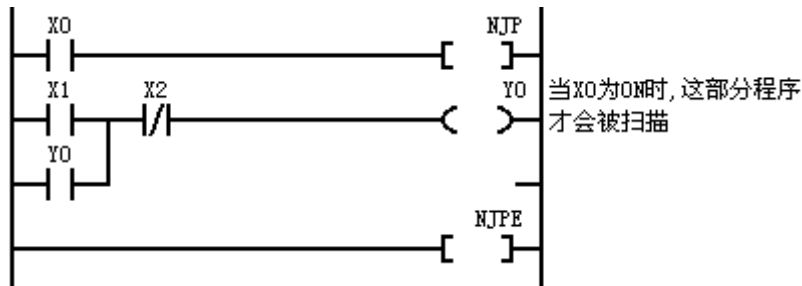
1.8.2 NJP/NJPE 指令

若 NJP 指令被接通，则 NJP 和与之配对的 NJPE 之间的指令将被扫描；若 NJP 指令被断开，则 NJP 和与之配对的 NJPE 之间的指令将跳过不被扫描。

NJP 和 NJPE 之间的指令的程序步数不能超过 255 步。

NJP/NJPE 指令和 DO/DOEND 指令相比，NJP/NJPE 指令执行速度快、占用程序步数少（占用 1 步）。

NJP/NJPE 指令可以嵌套，且使用次数不限，但应成对使用。



NJP/NJPE 指令的使用

注：若 NJPE 指令没有直接连接于左母线，则 NJPE 指令和其配对的 NJP 指令必须直接连接于同一垂直连线。

1.8.3 BLOCK/BREAK/BEND 指令

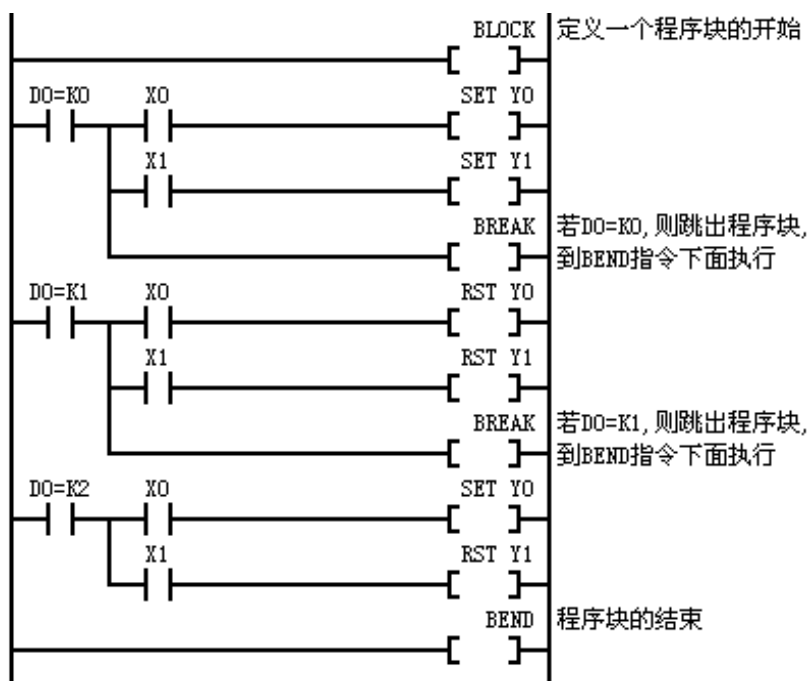
BLOCK 为程序块开始指令。

BEND 为程序块结束指令。

BREAK 为程序块中止指令。若该指令被接通，将立刻中止该指令所在的程序块的扫描，跳到 BEND 指令下面的指令处执行。该指令必须用在 BLOCK 和 BEND 指令之间。

用 BLOCK 和 BEND 指令可以把一些程序定义为一个程序块，然后可以使用 BREAK 指令来跳出该程序块。

BLOCK/BEND 指令可以嵌套，但应成对使用。

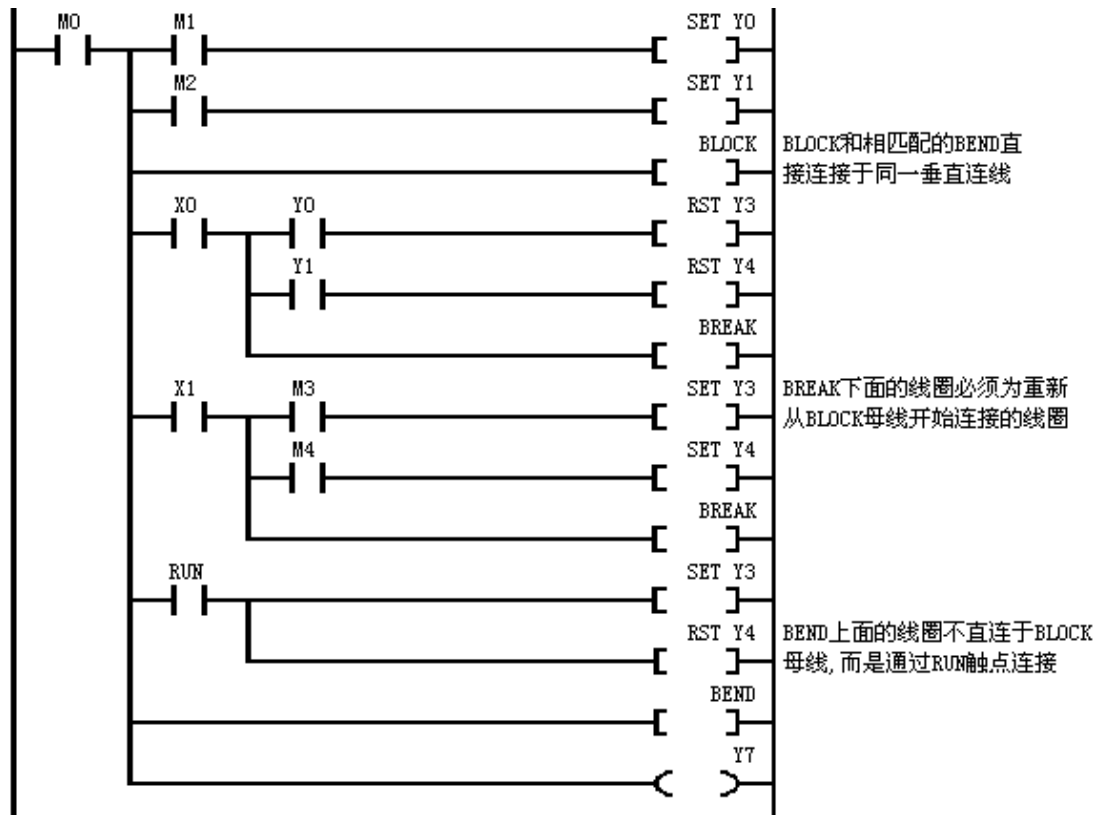


BLOCK/BREAK/BEND 指令的使用

BLOCK 和 BEND 指令也可以不直接连接于左母线，但此时应注意执行 BREAK 指令后对 MPS（逻辑运算结果进栈）/MPP（逻辑运算结果出栈）指令的影响。具体来说，应注意以下几点：

- (1) BLOCK 和相匹配的 BEND 必须直接连接于同一垂直连线（以下简称 BLOCK 母线）。
- (2) BREAK 下面的线圈必须为重新从 BLOCK 母线开始连接的线圈。
- (3) BEND 上面的线圈不要直接连接于 BLOCK 母线，必要时可通过 RUN 辅助继电器的触点连接。

下面即是没有直接连接于左母线的 BLOCK/BEND 指令的梯形图例子：



不直接连接于左母线的 BLOCK/BEND 指令的结构

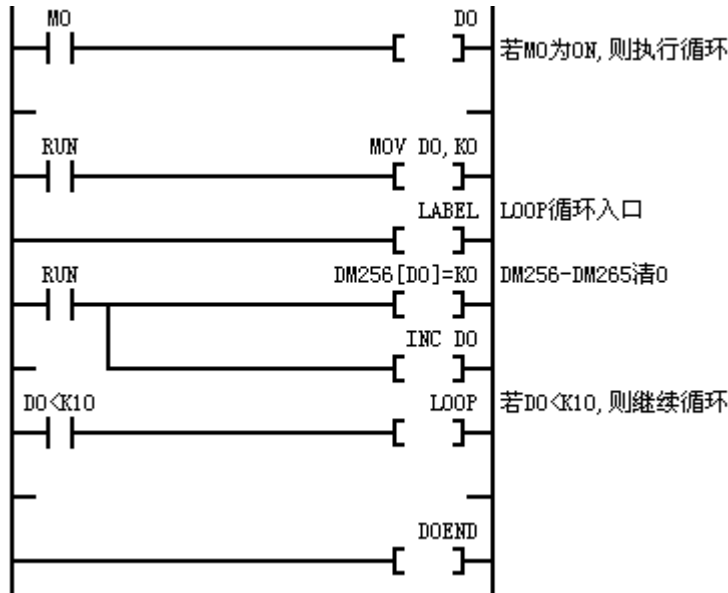
1.8.4 LABEL/EXITLOOP/LOOP 指令

LABEL 指令为 LOOP 指令的循环入口标签。该指令必须直接与左母线连接，否则将产生保护性错误。

EXITLOOP 指令为退出 LOOP 循环指令。若该指令被接通，将立刻退出 LOOP 循环，跳到 LOOP 指令下面的指令处执行。该指令必须用在 LABEL 和 LOOP 指令之间。

LOOP 指令为条件循环指令。若该指令被接通，则使程序跳到与之匹配的 LABEL 指令处循环执行，直到该指令被断开为止。LOOP 指令下面的指令，应为从左母线开始的指令，否则应注意执行 EXITLOOP 指令后对 MPS（逻辑运算结果进栈）/MPP（逻辑运算结果出栈）指令的影响。

LABEL/LOOP 指令可以嵌套，但应成对使用。



LABEL/LOOP 指令的使用

1.8.5 FOR/EXITFOR/NEXT 指令

FOR/NEXT 循环指令有两种格式：

第 1 种：FOR IN/NEXT OUT 格式

● FOR IN

操作数	可使用的元件
IN	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址

说明：

FOR 指令为 FOR/NEXT 循环入口指令。

IN 为循环计数器变量，IN 不能为常数，该指令前 IN 中的值为循环次数。

FOR 指令的循环计数器变量和相配对的 NEXT 指令的循环计数器变量要一致。

注：该格式下 FOR 指令可以直接与左母线连接。若没有直接与左母线连接，则和相配对的 NEXT 指令都要直接连接于同一垂直连线，并且若 FOR 指令上面连接该垂直线的所有分支是直连该垂直线，则 NEXT 指令下面连接该垂直线的所有分支也必须是直连该垂直线或者无连接该垂直线的分支；若 FOR 指令上面连接该垂直线的任一分支不是直连该垂直线（经过触点连接），则 NEXT 指令下面也必须有连接该垂直线的分支且不是直连该垂直线（经过触点连接）。

● NEXT OUT

操作数	可使用的元件
OUT	D、RM、RY、RX、RC、RT、DM、LDM、变址寻址

说明：

NEXT 指令为 FOR/NEXT 循环计数指令。

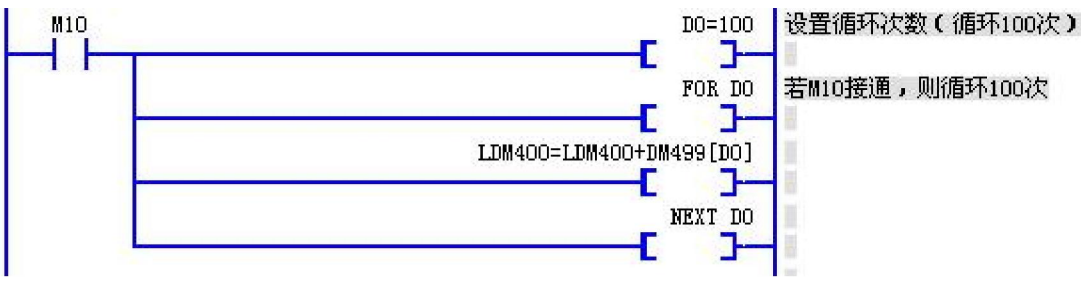
OUT 为循环计数器变量，不能为常数。

NEXT 指令的循环计数器变量和相配对的 FOR 指令的循环计数器变量要一致。

若该指令被接通，则把 OUT 的内容减 1，若减 1 后的结果不为 0，则使程序跳转到该指令相配对的循环入口 FOR 指令处执行，若减 1 后的结果为 0，或者该指令被断开，则程序按顺序执行该指令的下面一条指令。

注：该格式下 NEXT 指令为条件执行，不可以直接与左母线连接。

第 1 种 FOR 指令格式梯形图例子：



第 2 种：FOR IN1，IN2/NEXT 格式

● FOR IN1，IN2

操作数	可使用的元件
IN1	D、RM、RY、RX、RC、RT、变址寻址
IN2	常数、D、RM、RY、RX、RC、RT、变址寻址

根据 IN1 和 IN2 的值来决定是否循环扫描 FOR 和 NEXT 之间的指令，若该指令被接通且 $IN1 \leq IN2$ ，则 FOR 和 NEXT 之间的指令被循环扫描，直到 $IN1 > IN2$ 为止；否则 FOR 和 NEXT 之间的指令跳过不被扫描。

FOR 指令下面的指令，应为从左母线开始的指令。

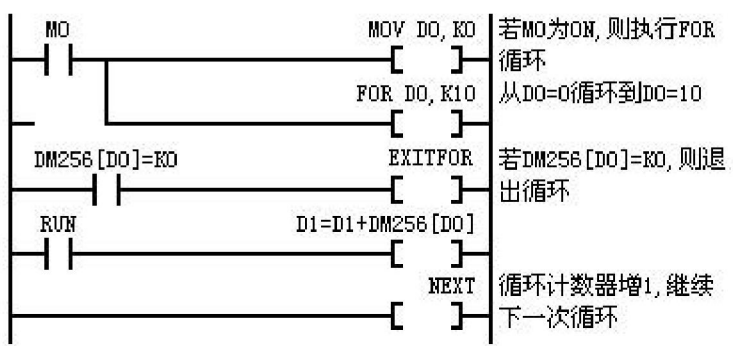
● NEXT

使与之匹配的 FOR 指令中的 IN1 的值增 1，并跳到该 FOR 指令处执行。该指令必须直接与左母线连接，否则将产生错误。

● EXITFOR

若该指令被接通，将立刻退出 FOR 循环，跳到 NEXT 指令下面的指令处执行，跳到的指令应为从左母线开始的指令或触点，否则应注意执行 EXITFOR 指令后对 MPS/MPP 指令的影响。该指令可以用在两种格式的 FOR 和 NEXT 指令之间。

第 2 种 FOR 指令格式梯形图例子：



FOR/NEXT 指令可以嵌套，但应成对使用。

1.9 变量定义、表达式和函数

1.9.1 数据类型和存储类型

1、数据类型

一个变量可以按其数据类型来进行定义。可定义的数据类型如下：

(1) BOOL 布尔类型

用来表示变量为“TRUE/FALSE”、“YES/NO”、“ON/OFF”等开关量信息的一种数据类型，其长度为 1 个位，且总是被分配在 M 存储区（辅助继电器区）。

(2) INT 整型

用来表示变量为不带小数点的数字（整数），其长度为 1 个字（16 位），所能表示的数值范围是-32768~32767，字的最高位表示该数据的符号：“0”表示正数，“1”表示负数。被声明为 INT 类型的变量总是被分配在从 DM384 开始的 DM 数据存储器中。

(3) LONG（或 DINT） 长整型

用来表示变量为不带小数点的数字（整数），其长度为 2 个字（32 位），所能表示的数值范围是-2147483648~2147483647。被声明为 LONG 类型的变量总是被分配在从 LDM384 开始的 DM 数据存储器中，且每个变量占用两个连续的 DM 单元，编号小的单元存放高 16 位并且其地址（编号）即为变量的地址。

(4) FLOAT（或 REAL） 浮点型

用来表示变量为带小数点的数字（实数），为 IEEE-754 标准单精度浮点格式。被声明为 FLOAT 类型的变量总是被分配在从 FDM384 开始的 DM 数据存储器中，且每个变量占用两个连续的 DM 单元。

(5) BIT 位类型

用来表示把变量分配在从 DM384 开始的 DM 存储器中的某个位。

2、存储类型

一个变量也可以按其存储类型来进行声明。可声明的存储类型如下：

(1) X 类型

用来表示把变量分配在 X 继电器区，表明该变量为 1 个 X 继电器。

(2) Y 类型

用来表示把变量分配在 Y 继电器区，表明该变量为 1 个 Y 继电器。

(3) M 类型

用来表示把变量分配在 M 继电器区，表明该变量为 1 个 M 继电器。

(4) T_100、T_10 类型

用来表示把变量分配在 T0~T23 的定时器区，表明该变量为 1 个时基为 100 ms 或 10ms 的定时器。

(5) T_200 类型

用来表示把变量分配在 T384~以后的定时器区，表明该变量为 1 个时基为 1 秒或 100ms 或 10ms 或自定义时基的定时器。

(6) C_15 类型

用来表示把变量分配在计数器区 C3~C23 中，表明该变量为 1 个 15 位的计数器，其计数范围为 1~32767。

(7) C_14 类型

用来表示把变量分配在计数器区 C384~以后中，表明该变量为 1 个 14 位的计数器，其计数范围为 1~16383。

(8) RT 类型

用来表示把变量分配在定时器的当前值寄存器区 RT0~RT23 中，表明该变量可以作为 1 个 15 位的无符号整数，其数值范围为 0~32767。

(9) RC 类型

用来表示把变量分配在计数器的当前值寄存器区 RC3~RC23 中，表明该变量可以作为 1 个 INT 整型变量，其数值范围为-32768~32767。

(10) DM 类型

用来表示把变量分配在从 DM384 开始的 DM 数据存储器中（字），表明该变量为 1 个 INT 整型变量，其数值范围为-32768~32767。

(11) LDM、!LDM 类型

用来表示把变量分配在从 LDM384 开始的 DM 数据存储器中（双字），表明该变量为 1 个 LONG 长整型变量，其数值范围为-2147483648~2147483647，并且该变量占用两个连续的 DM 单元，LDM 类型为编号小的 DM 单元存放高 16 位，!LDM 类型为编号小的 DM 单元存放低 16 位。

(12) FDM 类型

用来表示把变量分配在从 FDM384 开始的 DM 数据存储器中（浮点），表明该变量为 1 个 IEEE-754 标准单精度浮点型变量，且该变量占用两个连续的 DM 单元。

(13) NOV 类型

用来表示把变量分配在非易失性数据存储器区（DM0～DM255），表明该变量用来存储 1 个非易失性的整型（字）数据。

(14) DMx.y 类型

用来表示把变量分配在从 DM384 开始的 DM 存储器中的某个位。

1.9.2 使用 EQU/AS 定义变量

使用 EQU 可以定义全局变量，使用 AS 可以定义静态局部变量。其格式如下：

变量名 EQU 直接地址或数据类型或存储类型

变量名 AS 直接地址或数据类型或存储类型

或者多个变量定义：

变量名 1, 变量名 2, …, 变量名 n EQU 数据类型或存储类型

变量名 1, 变量名 2, …, 变量名 n AS 数据类型或存储类型

或者连续编号变量定义：

字符串 m-n EQU 直接地址或数据类型或存储类型

字符串 m-n AS 直接地址或数据类型或存储类型

其中字符串的最后一个字符不为数字，m、n 为非负的整数数字编号。相当于定义了字符串 m 到字符串 n 之间连续编号的多个变量。

例如

按直接地址定义方式：

IN1 EQU X0 ; 定义 1 个全局变量

FLAG1 AS M10 ; 定义 1 个静态局部变量

连续编号变量定义方式：

Var10-19 EQU DM500 ; 定义变量 Var10 在 DM500、Var11 在 DM501、Var12 在 DM502、……、Var19 在 DM509。

按数据类型定义方式：

FLAG1 EQU BOOL ; 定义 1 个全局变量

VAR1, VAR2, VAR3 AS INT ; 定义 3 个静态局部变量

按存储类型定义方式:

ALM1 EQU Y ; 定义 1 个全局变量

TIMER EQU T_100 ; 定义 1 个全局变量

CNT1 AS C_15 ; 定义 1 个静态局部变量

VAR1 AS DM ; 定义 1 个静态局部变量

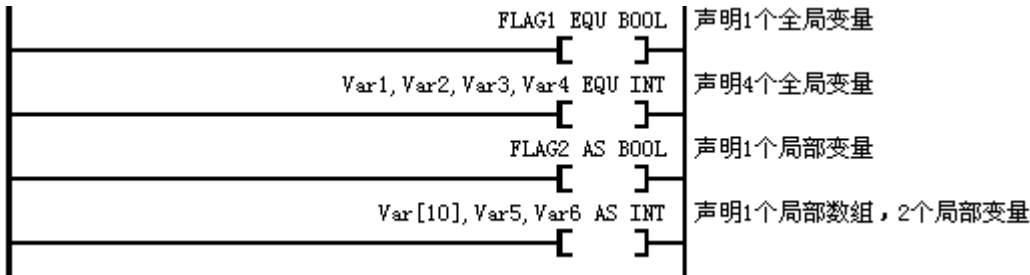
全局变量是用 EQU 宏指令定义的变量，其有效作用范围是从它定义的位置开始到整个程序结束。如果全局变量定义在一个程序文件的开始处，则在整个程序文件范围内都可以使用它。在整个程序中，全局变量名必须是唯一的。

静态局部变量是用 AS 宏指令定义的变量，如果是在子程序或函数或主程序内定义的，只能在定义它的那个程序或函数或主程序内访问，如果是在子程序或函数外（不在主程序内）定义的，在定义它后的所有子程序或函数内都可访问，但不可与这些子程序或函数内的局部变量重名。在程序的整个运行期间静态局部变量存储位置固定不变，即退出定义它的那个子程序或函数之后虽不能进行访问，但变量的值仍然保持。不同的子程序或函数或主程序可以使用相同的局部变量名，由于它们的作用范围不同，不会相互干扰。

局部变量可以与全局变量同名，但在这种情况下局部变量的优先级较高，而同名的全局变量在定义该局部变量的子程序或函数或主程序内被暂时屏蔽。

注：在全局符号表和程序中，若同时使用按直接地址定义变量和按数据类型（或存储类型）定义变量，则应把按直接地址定义变量放在最前面，并且在按数据类型（或存储类型）定义变量的后面（包括程序中）不要再使用元件名和直接地址。

变量定义在梯形图中的例子如下：



梯形图中的变量定义

1.9.3 数组

数组是一组有序数据的集合，数组中的每一个数据都属于同一个数据类型。数组中的各个元素可以用数组名和下标来唯一地确定。在 YF0H 系列 PLC 中可以定义一维、二维和三维数组，并且数组必须先定义，然后才能使用。

1、一维数组的定义形式如下：

数组名[常量] EQU 元件名或数据类型或存储类型

数组名[常量] AS 元件名或数据类型或存储类型

“数组名”是整个数组的标识符，其地址即为存储数组的一系列单元的起始地址也即数组的地址，使用“#数组名”即可得到该地址的值。可以把该地址的值传递给某个函数，然后在该函数中即可对该数组进行操作。

“常量”说明了该数组的长度，即该数组中的元素个数。

每个数组都具有两个只读属性：

数组名.LEN 表示数组的长度，即该数组中的元素个数。

数组名.UB 表示数组的最大下标，而数组的下标总是从 0 开始的，因此数组的最大下标总是等于数组的长度-1。

在程序中，通过访问这两个只读属性即可知道一个数组的长度和最大下标。

还有一个可读写属性：

数组名.Rx 为数组的第 x 个（从 0 开始）元素所对应的 PLC 内部元件，x 为常数。例如“数组名.R5”表示数组的第 5 个元素，该属性可用在比较触点、指令和表达式中进行读写。

下面是几个定义数组的例子：

按元件名定义方式：

BIT[5] EQU M10 ；定义全局数组 BIT，它具有 5 个元素（M10～M14）。

#BIT 为 10，BIT.LEN 为 5，BIT.UB 为 4。

VR[8] AS DM300 ；定义局部数组 VR，它具有 8 个元素（DM300～

DM307）。#VR 为 300，VR.LEN 为 8，VR.UB 为 7。

按数据类型定义方式：

BIT[5] EQU BOOL ；定义 BOOL 型全局数组 BIT，它具有 5 个元素。

VAR[10] EQU INT ；定义 INT 型全局数组 VAR，它具有 10 个元素。

N[15] AS LONG ; 定义 LONG 型静态局部数组 N，它具有 15 个元素，
且要占用 30 个 DM 存储单元。

按存储类型定义方式：

BIT[50] EQU DMx.y ; 定义位型（DM 存储器中的位）全局数组 BIT。

CNT[4] EQU C_15 ; 定义位于 15 位计数器区的全局数组 CNT。

VAR[10] AS LDM ; 定义位于 DM 区的 LONG 长整型（双字）局部数组。

N[15] AS DM ; 定义位于 DM 区的 INT 整型静态局部数组。

使用变址寻址方式或变址函数#数组名(元素索引)来读写数组中的每一个元素。位于不同存储区域的数组要遵从各自存储区域的变址寻址方式。当使用变址寻址方式时，若数组为 LONG 长整型或 FLOAT 浮点型数组，则变址寻址的偏移量必须为 0、2、4、6……等偶数才能正确地访问数组中的每一个元素。

例如：

使用变址函数来读写一维数组（VAR 为长整型数组）：

LDM500 = #VAR(2) + #VAR(5)

#VAR(3) = LDM500 + #VAR(5)

使用数组属性高效率读写数组固定位置的元素（VAR 为长整型数组）：

LDM500 = VAR.R2 + VAR.R5

VAR.R3 = LDM500 + VAR.R5

使用变址寻址方式来读写一维数组：

若数组 VAR 为 INT 整型数组，则

VAR[0] 访问数组中的第 0 个元素，

VAR[1] 访问数组中的第 1 个元素，

VAR[2] 访问数组中的第 2 个元素，

若数组 VAR 为 LONG 长整型或 FLOAT 浮点型数组，则

VAR[0] 访问数组中的第 0 个元素，

VAR[2] 访问数组中的第 1 个元素，

VAR[4] 访问数组中的第 2 个元素，

若数组的地址在数据寄存器 D0（或 D1）中，则也可按下方式来访问数组中的每一个元素（首先要知道数组位于哪个存储区域）：

假设数组 VAR 为 INT 整型数组（位于 DM 区域），数据寄存器 D0 中存有该数组的地址（D0 = #VAR），则

DM0[D0] 访问数组中的第 0 个元素，

DM1[D0] 访问数组中的第 1 个元素，

DM2[D0] 访问数组中的第 2 个元素，

而若该数组为 LONG 长整型数组，则

LDM0[D0] 访问数组中的第 0 个元素，

LDM2[D0] 访问数组中的第 1 个元素，

LDM4[D0] 访问数组中的第 2 个元素，

而若该数组为 FLOAT 浮点型数组，则

FDM0[D0] 访问数组中的第 0 个元素，

FDM2[D0] 访问数组中的第 1 个元素，

FDM4[D0] 访问数组中的第 2 个元素，

2、二维数组的定义形式如下：

数组名[行数][列数] EQU 元件名或数据类型或存储类型

数组名[行数][列数] AS 元件名或数据类型或存储类型

“数组名”是整个数组的标识符，其地址即为存储数组的一系列单元的起始地址也即数组的地址，使用“#数组名”即可得到该地址的值。可以把该地址的值传递给某个函数，然后在该函数中即可对该数组进行操作。

“行数”和“列数”必须为常数，说明了该数组的总行数和总列数。数组的行或列下标都是从 0 开始，到总行数-1 或总列数-1 为止。

每个数组都具有 4 个只读属性：

数组名.LEN 为数组的总长度，即该数组中的元素总个数。

数组名.UB 为数组的总长度-1。

数组名.Rows 为数组的总行数。

数组名.Cols 为数组的总列数。

还有两个可读写属性（可用在比较触点、指令和表达式中进行读写）：

数组名.Rx 为二维数组的第 x 行（从 0 开始）的首个元素所对应的 PLC 内

部元件，x 为常数。例如“数组名.R5”表示二维数组的第 5 行的首个元素。

数组名.Rx.Cy 为二维数组的第 x 行第 y 列（均从 0 开始）元素所对应的 PLC 内部元件，x、y 为常数。例如“数组名.R5.C8”表示二维数组的第 5 行第 8 列元素。

下面是几个定义二维数组的例子：

VAR[7][8] EQU DM500 ;定义 7 行 8 列单字二维数组从 DM500 开始

DAT[9][5] EQU LDM600 ;定义 9 行 5 列双字二维数组从 LDM600 开始

CNT[15][6] EQU INT ;定义 15 行 6 列整型（字）二维数组

NUM[5][10] EQU FLOAT ;定义 5 行 10 列浮点型二维数组

可使用“#数组名(行, 列)”格式变址函数读写二维数组，例如：

LDM1000 = #DAT(3,2) + #VAR(6,7)

#DAT(3,2) = LDM1000 + #VAR(6,7)

或者使用二维数组的行列属性高效率读写数组固定位置的元素，例如：

LDM1000 = DAT.R3.C2 + VAR.R6.C7

DAT.R3.C2 = LDM1000 + VAR.R6.C7

也可使用取数组单元地址函数得到二维数组某行的存储首地址，然后再使用变址寻址方式读写该行各个列的数据。

3、三维数组的定义形式如下：

数组名[行数][列数][第三维元素数] EQU 元件名或数据类型或存储类型

数组名[行数][列数][第三维元素数] AS 元件名或数据类型或存储类型

“数组名”是整个数组的标识符，其地址即为存储数组的一系列单元的起始地址也即数组的地址，使用“#数组名”即可得到该地址的值。可以把该地址的值传递给某个函数，然后在该函数中即可对该数组进行操作。

“行数”和“列数”和“第三维元素数”必须为常数，说明了该数组的总行数和总列数和第三维总元素数。数组的行或列或第三维下标都是从 0 开始，到总行数-1 或总列数-1 或第三维总元素数-1 为止。

每个数组都具有 5 个只读属性：

数组名.LEN 为数组的总长度，即该数组中的元素总个数。

数组名.UB 为数组的总长度-1。

数组名.Rows 为数组的总行数。

数组名.Cols 为数组的总列数。

数组名.Thrs 为数组的第三维总元素数。

还有 3 个可读写属性（可用在比较触点、指令和表达式中进行读写）：

数组名.Rx 为三维数组的第 x 行（从 0 开始）的首个元素所对应的 PLC 内部元件，x 为常数。例如“数组名.R5”表示三维数组的第 5 行的首个元素。

数组名.Rx.Cy 为三维数组的第 x 行第 y 列（均从 0 开始）的首个元素所对应的 PLC 内部元件，x、y 为常数。例如“数组名.R5.C8”表示三维数组的第 5 行第 8 列的首个元素。

数组名.Rx.Cy.Tz 为三维数组的第 x 行第 y 列第 z 个（均从 0 开始）元素所对应的 PLC 内部元件，x、y、z 为常数。例如“数组名.R5.C8.T6”表示三维数组的第 5 行第 8 列第 6 个元素。

下面是几个定义三维数组的例子：

`VAR[7][8][9] EQU DM500` ;定义单字三维数组从 DM500 开始

`DAT[9][5][6] EQU LDM600` ;定义双字三维数组从 LDM600 开始

`CNT[15][6][7] EQU INT` ;定义整型（字）三维数组

`NUM[5][10][8] EQU FLOAT` ;定义浮点型三维数组

可使用“**#数组名(行, 列, 第三维索引)**”格式变址函数读写三维数组，例如：

`LDM1000 = #DAT(3,2,4) + #VAR(6,7,5)`

`#DAT(3,2,4) = LDM1000 + #VAR(6,7,5)`

或者使用三维数组的行列属性高效率读写数组固定位置的元素，例如：

`LDM1000 = DAT.R3.C2.T4 + VAR.R6.C7.T5`

`DAT.R3.C2.T4 = LDM1000 + VAR.R6.C7.T5`

也可使用取数组单元地址函数得到三维数组某行某列的存储首地址，然后再使用变址寻址方式读写该行列中各个元素的数据。

4、取数组的单元（元素）存储地址

若取数组固定单元（行列值都为常数）的存储地址，可对数组的行列属性使

用取元件地址符号“#”来获取，例如：#DAT.R5（取数组第 5 行的存储地址），#DAT.R3.C2（取数组第 3 行第 2 列的存储地址）

若取数组可变单元（行或列值为变量）的存储地址，可使用取数组单元地址函数来获取，函数格式为：

%数组名(行) ;取一维或二维或三维数组某行的存储地址

%数组名(行, 列) ;取二维或三维数组某行某列的存储地址

%数组名(行, 列, 第三维索引) ;取三维数组某个元素的存储地址

函数参数中的行、列、第三维索引可以是常数、变量或表达式。

例如：

%DAT(DM500);取 DAT 数组中行为 DM500 的存储地址

%DAT(D0,D1);取 DAT 数组中行为 D0、列为 D1 的存储地址

5、连续编号数组的定义形式如下：

字符串[常量]m-n EQU 元件名或数据类型或存储类型

字符串[常量]m-n AS 元件名或数据类型或存储类型

字符串[行数][列数]m-n EQU 元件名或数据类型或存储类型

字符串[行数][列数]m-n AS 元件名或数据类型或存储类型

其中 m、n 为非负的整数数字编号。相当于定义了字符串 m[常量]到字符串 n [常量]之间连续编号的多个数组。

例如：

VP[5]1-4 相当于定义了 VP1[5]、VP2[5]、VP3[5]、VP4[5]四个数组。

DAT[6][8]1-5 相当于定义了 DAT1[6][8]、DAT2[6][8]、DAT3[6][8]、DAT4[6][8]、DAT5[6][8]五个数组。

注：位元件数组只能使用数组的行列属性来访问，而不能使用变址函数和取数组单元地址函数来访问。

1.9.4 结构体

结构体就是用来定义具有一些属性成员的标识符，其属性成员的数据类型或存储类型可以不相同。

结构体的定义格式如下:

STRUCT 标识符 1, 标识符 2, 标识符 3, ……., 标识符 n

成员 1 EQU[AS] 类型, 类型, 类型,, 类型

成员 2 EQU[AS] 类型, 类型, 类型,, 类型

成员 3 EQU[AS] 类型, 类型, 类型, …… , 类型

● ● ● ● ● ●

ENDSTRUCT

其“类型”的数目必须与标识符的数目相同，并且按顺序分别对应于各个标识符，表示对应的成员的数据类型或存储类型。“类型”可以为数据类型或存储类型或直接地址，也可以为常数，当为常数时表示为只读属性。

当结构体定义后，即可对其中的成员进行引用，引用格式为：

结构体标识符,成员名

下面是结构体的定义例子:

STRUCT AV1, AV2, AV3, AV4

GAIN EQU K100, K150, K201, K321 ; 定义增益属性

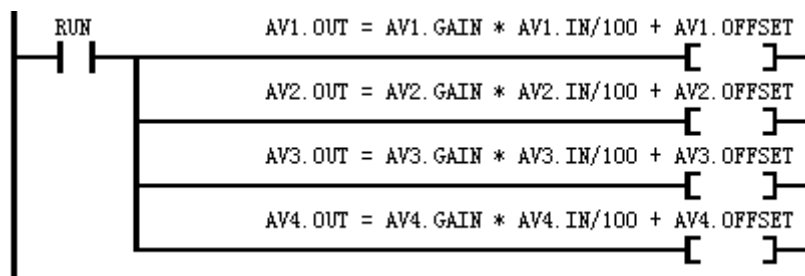
OFFSET EQU K0, K-2, K5, K10 ; 定义偏移属性

IN EQU INT, INT, INT, INT ; 定义存储修正前输入

OUT EQU INT, INT, INT, INT ; 定义存储修正后结果

ENDSTRUCT

其梯形图例子如下:



属性成员也可使用 **DM** 存储器区类型+地址偏移的符号定义形式（使用前缀

%) 对数据块进行数据结构定义，例如：

```
STRUCT G           ;结构名
  LEN EQU 6         ;常数，定义数据块长度（字数）
  CMD EQU %DM0      ;字，地址偏移 0，前缀%表示类型+地址偏移
  STOP EQU %DM1.0   ;位，地址偏移 1，前缀%表示类型+地址偏移
  RUN EQU %DM1.1     ;位，地址偏移 1，前缀%表示类型+地址偏移
  Xe EQU %LDM2       ;双字，地址偏移 2，前缀%表示类型+地址偏移
  Ye EQU %LDM4       ;双字，地址偏移 4，前缀%表示类型+地址偏移
ENDSTRUCT
```

或者使用适合全局符号表的数据结构定义格式：

```
G.                 ;结构名
LEN EQU 6          ;常数，定义数据块长度（字数）
CMD EQU %DM0       ;字，地址偏移 0，前缀%表示类型+地址偏移
STOP EQU %DM1.0    ;位，地址偏移 1，前缀%表示类型+地址偏移
RUN EQU %DM1.1     ;位，地址偏移 1，前缀%表示类型+地址偏移
Xe EQU %LDM2       ;双字，地址偏移 2，前缀%表示类型+地址偏移
Ye EQU %LDM4       ;双字，地址偏移 4，前缀%表示类型+地址偏移
                  ;碰到空行表示该数据结构定义结束
```

在该格式中，结构名、各个属性成员之间不能有空行，碰到空行表示该数据结构定义结束。

对类型+地址偏移的结构体可使用变址寻址来访问数据块，其格式如下：

结构名.成员名[数据块地址]

结构名.成员名@数据块名

为常数的结构体成员可直接使用“结构名.成员名”，而不应使用变址寻址。

例如定义一个 DM 数据块数组 SV[G.LEN]，则可使用变址格式 G.CMD@SV、G.STOP@SV、G.RUN@SV、G.Xe@SV、G.Ye@SV 来对数据块进行使用。若数据块地址在 D0 中，则可使用变址格式 G.CMD[D0]、G.STOP[D0]、G.RUN[D0]、G.Xe[D0]、G.Ye[D0]来对数据块进行使用。

类型+地址偏移的结构体在全局符号表中的定义例子如下：

	符号名	类型或地址	描述
1	G.		数据结构名
2	LEN	6	常数，定义数据块长度（字数）
3	CMD	%DM0	字，地址偏移0，前缀%表示类型+地址偏移
4	STOP	%DM1.0	位，地址偏移1，位0，前缀%表示类型+地址偏移
5	RUN	%DM1.1	位，地址偏移1，位1，前缀%表示类型+地址偏移
6	Xe	%LDM2	双字，地址偏移2，前缀%表示类型+地址偏移
7	Ye	%LDM4	双字，地址偏移4，前缀%表示类型+地址偏移
8			空行表示该数据结构定义结束
9	SV[G.LEN]	DM500	定义一个DM数据块数组

也可直接使用变量定义形式定义数据结构，上面结构可使用如下形式定义：

G.LEN EQU 6 ; 常数，定义数据块长度（字数）

G.CMD EQU %DM0 ; 字，地址偏移 0，前缀%表示类型+地址偏移

G.STOP EQU %DM1.0 ; 位，地址偏移 1，前缀%表示类型+地址偏移

G.RUN EQU %DM1.1 ; 位，地址偏移 1，前缀%表示类型+地址偏移

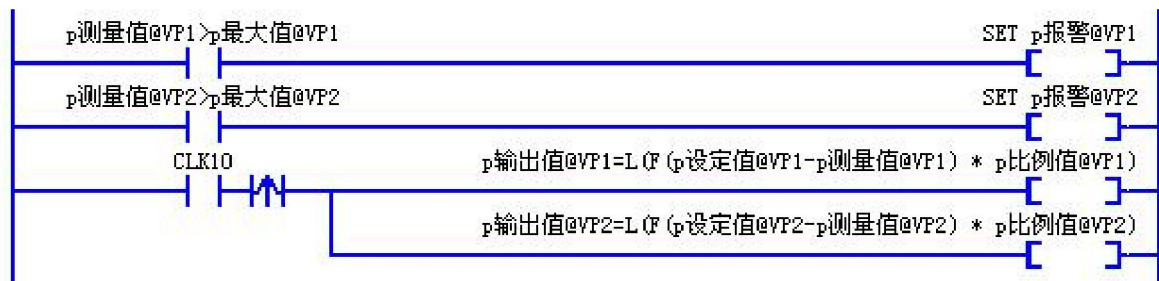
G.Xe EQU %LDM2 ; 双字，地址偏移 2，前缀%表示类型+地址偏移

G.Ye EQU %LDM4 ; 双字，地址偏移 4，前缀%表示类型+地址偏移

直接使用变量定义形式定义数据结构在全局符号表中的定义例子如下：

	符号名	类型或地址	描述
1	p测量值	%DM0	数据块数据结构定义 (数据类型和地址偏移定义)
2	p设定值	%DM1	
3	p输出值	%DM2	
4	p最小值	%DM3	
5	p最大值	%DM4	
6	p运行	%DM5.0	
7	p报警	%DM5.1	
8	p自动	%DM5.2	
9	p比例值	%FDM6	
10			
11	VP[8]1-4	DM1000	连续编号数组定义，定义VP1[8]、VP2[8]、VP3[8]、VP4[8]共4个数组

数据结构在梯形图中的使用例子如下：



对于多级数据结构嵌套，可使用多级变址寻址来访问各个属性成员，其格式如下：

最底级结构名.成员名@上一级结构名.成员名……[数据块地址]

最底级结构名.成员名@上一级结构名.成员名……@数据块名

例如在 CV 结构中嵌套有 G 结构，DT 为定义的为 CV 结构的数据块名，可以这样 G.Xe@CV.G@DT 来访问 DT 数据块中的 G.Xe 属性成员。若 DT 数据块的地址在 D0 中，可以这样 G.Xe@CV.G[D0]来访问 DT 数据块中的 G.Xe 属性成员。

※ 在数据结构定义中如何处理字节类型：

在 DM 存储器区类型+地址偏移的数据结构定义形式中，类型也可为高字节或低字节（地址偏移总是按字来），高字节为%DMHx，低字节为%DMLx，x 为字地址偏移，例如：

BYTE1 EQU %DML5 ;字节，地址偏移 5 的低字节

BYTE2 EQU %DMH5 ;字节，地址偏移 5 的高字节

BYTE3 EQU %DML6 ;字节，地址偏移 6 的低字节

可使用 RB 函数（DM 存储器区按字节地址读字节）和 WB 函数（DM 存储器区按字节地址写字节）来处理这些字节数据，例如对于 DM 数据块数组 SV：

读 BYTE1 属性可使用函数 RB(#BYTE1[#SV])，写 BYTE1 属性可使用函数 WB(#BYTE1[#SV1], 数值)来实现。

读 BYTE2 属性可使用函数 RB(#BYTE2[#SV])，写 BYTE2 属性可使用函数 WB(#BYTE2[#SV1], 数值)来实现。

读 BYTE3 属性可使用函数 RB(#BYTE3[#SV])，写 BYTE3 属性可使用函数 WB(#BYTE3[#SV1], 数值)来实现。

若数据块地址在 D0 中，则：

读 BYTE1 属性可使用函数 RB(2*D0+#BYTE1)，写 BYTE1 属性可使用函数 WB(2*D0+#BYTE1, 数值)来实现。

读 BYTE2 属性可使用函数 RB(2*D0+#BYTE2)，写 BYTE2 属性可使用函数 WB(2*D0+#BYTE2, 数值)来实现。

读 BYTE3 属性可使用函数 RB(2*D0+#BYTE3)，写 BYTE3 属性可使用函数 WB(2*D0+#BYTE3, 数值)来实现。

※ 如何读写数据结构中定义的数组：

可使用数组的行列属性“数组名.行列属性[数据块地址]”或者“数组名.行列属性@数据块名”格式高效率读写数据块数据结构中定义的数组固定位置的元素数值。

可使用“#数组名[数据块地址](数组各维索引)”或者“#数组名@数据块名(数组各维索引)”格式变址函数读写数据块数据结构中定义的数组可变位置的元素数值。

可使用“%数组名[数据块地址](数组各维索引)”或者“%数组名@数据块名(数组各维索引)”格式取地址函数获取数据块数据结构中定义的数组可变位置的存储地址。

例如数据块数据结构定义如下：

PG. ;结构名

LEN EQU 50 ;数据块长度（字数）

ARR[5][8] EQU %DM0 ;5 行 8 列数组，地址偏移 0

定义一个 DM 数据块数组 SV[PG.LEN]，要读写其中的 PG. ARR 数组：

读写固定位置的元素：

LDM1000 = PG. ARR. R3. C2@SV + PG. ARR. R2. C5@SV

PG. ARR. R3. C2@SV = LDM1000 + PG. ARR. R1. C6@SV

读写可变位置的元素（行列位置在 D2、D3 中）：

LDM1000 = #PG. ARR@SV(D2,D3) + #PG. ARR@SV(D2,D3+1)

#PG. ARR@SV(D2,D3+2) = LDM1000 + #PG. ARR@SV(D2+1,D3)

若 SV 数据块的地址在 D0 中，要读写其中的 PG. ARR 数组：

读写固定位置的元素：

LDM1000 = PG. ARR. R3. C2[D0] + PG. ARR. R2. C5[D0]

PG. ARR. R3. C2[D0] = LDM1000 + PG. ARR. R1. C6[D0]

读写可变位置的元素（行列位置在 D2、D3 中）：

LDM1000 = #PG. ARR[D0](D2,D3) + #PG. ARR[D0](D2,D3+1)

#PG. ARR[D0](D2,D3+2) = LDM1000 + #PG. ARR[D0](D2+1,D3)

※ 如何定义结构体数组：

使用二维数组来定义一维结构体数组，使用三维数组来定义二维结构体数组，数组的数据类型为字（INT 或 WORD）：

一维结构体数组定义：

数组名[行数][结构体长度] EQU INT

数组名[行数][结构体长度] AS INT

二维结构体数组定义：

数组名[行数][列数][结构体长度] EQU INT

数组名[行数][列数][结构体长度] AS INT

读写数组中的某个结构体时，可先获取该结构体的存储地址，再使用变址寻址格式来读写该结构体中的各个成员。

例如一维结构体数组定义：VG[10][G.LEN] EQU INT，定义 10 个 G 结构体数组 VG，读写第 5 个结构体中的成员：G.CMD@VG.R5、G.STOP@VG.R5、G.RUN@VG.R5、G.Xe@VG.R5、G.Ye@VG.R5。读写第 x 个结构体中的成员：先获取第 x 个结构体的存储地址 D0=%VG(x)，再使用 G.CMD[D0]、G.STOP[D0]、G.RUN[D0]、G.Xe[D0]、G.Ye[D0]。

1.9.5 使用全局符号表来定义全局变量

用户也可以使用全局符号表来定义全局变量和数组，即可把所有的全局变量放到全局符合表中，使变量的定义变得表格化，从而使变量的定义、编辑、维护更直观，更方便。

全局符号表的样式图如下：



	符号名	类型或地址	描述
1	KEY	INT	按键状态字
2	Enter	KEY.0	进入数据设定或切换设定项目键
3	Exit	KEY.1	退出数据设定键
4	IncKey	KEY.2	设定数据增1键
5	DecKey	KEY.3	设定数据减1键
6	UpKey	KEY.6	上翻页键
7	DownKey	KEY.7	下翻页键
8			
9	TMR1	T_200	开机画面保持时间定时器
10			
11	Screen	INT	屏编号
12	PageEn	BOOL	页面显示使能，为ON则显示页面中的字符常量，显示完后被复位
13	ModEnW	INT	数据修改使能字
14			
15	AI[4]	INT	声明1个整型数组
16			
17	SetVar1	INT	设定变量1
18	SetVar2	INT	设定变量2
19	SetVar3	INT	设定变量3
20	SetVar4	INT	设定变量4
21	SetVar5	INT	设定变量5
22	SetVar6	INT	设定变量6
23	SetVar7	INT	设定变量7
24	SetVar8	INT	设定变量8

全局符号表样式图

在梯形图编程软件 EasyLad 中，点击“工具”菜单中的“全局符号定义”菜单项即可打开全局符号表。

在全局符号表中：

“符号名”：用户要定义的全局变量（符号）的名称。可以是单个符号名格式例如 Var1，或者符号名数组格式例如 AI[4]，或者连续编号符号名格式例如 Var11-20（等同于定义了 10 个符号名 Var11、Var12、Var13、……、Var20）。

“类型或地址”：该符号的数据类型或存储类型或直接地址，也可为常数。当为常数时，表示该符号为常量。

“描述”：该变量（符号）的注释说明。在全局符号表中，变量的“描述”的内容可在“符号参照”中显示出来。

“插入”：在当前的位置插入一个新的符号。

“删除”：删除当前位置的符号或所选中的一块符号。

“查找”：在全局符号表中查找相匹配的符号名（只要符号名的前几个字符包含要查找的字符串即为相匹配）。

“确定”：保存全局符号表（若修改后还没保存过），同时隐藏全局符号表，回到梯形图编辑界面。

“关闭”：关闭全局符号表窗口。

1.9.6 表达式及其运算符

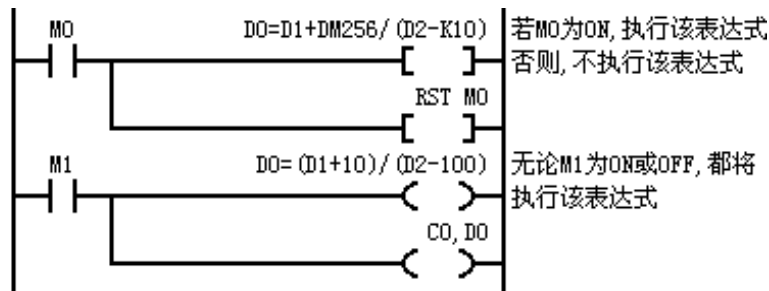
运算符就是完成某种特定运算的符号。表达式则是由运算符及变量或常数或函数所组成的（或者只是一个函数）具有特定含义的一个式子。

例如： $D0 = D1 + DM256 / (D2 - K10)$ ， $D0 = (D1 + K10) / (D2 - K100)$ ， $NOVWR(D0, 10)$ 都是合法的表达式。

在梯形图中，可使用指令线圈或输出线圈来放置一个表达式。

若使用指令线圈，该线圈被接通，则执行该表达式；该线圈被断开，则不执行该表达式。

若使用输出线圈，无论该线圈被接通或被断开都执行该表达式。但要使用 1 个逻辑堆栈存储器位，即先把该线圈前的逻辑运算结果压入逻辑堆栈存储器（若表达式中包含有函数调用，则并把该逻辑堆栈存储器的内容复制给函数中的逻辑堆栈存储器），待表达式执行完后再弹出。**注意当该线圈位于主控电路块或步进电路块中时，若电路块被断开则不执行该表达式。**



表达式的程序例

在表达式中，对于十进制常数，可以省去其前缀“K”。例如上述表达式也可以为： $D0 = D1 + DM256 / (D2 - 10)$ ， $D0 = (D1 + 10) / (D2 - 100)$ 。

1、赋值运算符

赋值运算符的符号为“=”，其作用是将一个表达式的值赋给一个变量（元件）。其格式如下：

变量 = 表达式

也可以使用多个“=”来进行多重赋值，其格式如下：

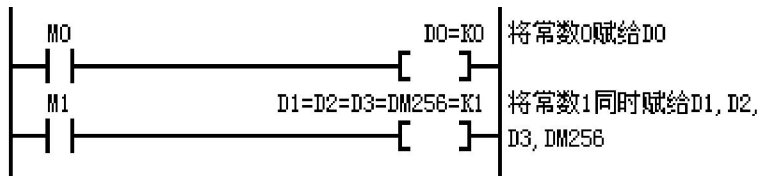
变量 1 = 变量 2 = = 表达式

其含义是将表达式的值同时赋给变量 1、变量 2、.....。

表达式的值和变量的数据类型必须一致，即若表达式的值为浮点型，则必须赋给浮点型的变量（浮点寄存器 F），而不能赋给整型或长整型的变量；若表达

式的值为整型（包含长整型），则必须赋给整型（包含长整型）的变量，而不能赋给浮点型的变量。否则将产生梯形图编译错误。

当把表达式的值赋给整型变量时，若该值超出了整型变量所能表示的范围（-32768~32767），则溢出继电器 OV 为 ON，但若没超出，并不使 OV 为 OFF。



赋值运算符的使用

2、算术运算符

EasyLad 梯形图语言的算术运算符有：+（加）、—（减）、*（乘）、/（除）共 4 种。

参加运算的两个运算对象的数据类型必须一致，即浮点型的数据只能和浮点型的数据进行算术运算，整型（包含长整型）的数据只能和整型（包含长整型）的数据进行算术运算。若两者的数据类型不一致，则必须使用数据类型转换函数转换为一致的数据类型，否则将产生梯形图编译错误。

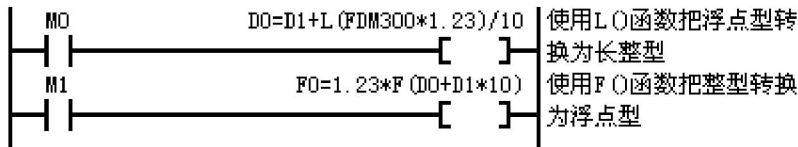
EasyLad 梯形图语言有两个数据类型转换函数：

（1）L（输入参数）

把浮点型的数据转换为长整型（包含整型）数据。输入参数必须为浮点型数据，函数的返回值为转换后的长整型（包含整型）数据。

（2）F（输入参数）

把长整型（包含整型）的数据转换为浮点型数据。输入参数必须为长整型（包含整型）数据，函数的返回值为转换后的浮点型数据。



数据类型转换函数的使用

3、位逻辑运算符

EasyLad 梯形图语言的位逻辑运算符有：&（按位与）、|（按位或）、^（按位异或）共 3 种。参加位逻辑运算的两个运算对象必须为整型（长整型）数据。

4、运算符的优先级

在求一个表达式的值时，要按运算符的优先级别进行。其中*（乘）、/（除）为高优先级，+（加）、-（减）、&（按位与）、|（按位或）、^（按位异或）为低优先级。需要时可在表达式中采用圆括号来改变运算符的优先级。对于优先级别相同的运算符，计算时按“从左至右”的结合方向进行。

例如在计算表达式 $D1+DM256/(D2-10)$ 的值时，首先计算 $(D2-10)$ ，然后再计算 $DM256/(D2-10)$ ，最后计算 $D1+DM256/(D2-10)$ 。

5、表达式中的溢出处理

在表达式的计算及赋值过程中，若发生了数据溢出，将会使溢出继电器 OV 为 ON，但若没有溢出，也不会使 OV 为 OFF。也就是说，OV 一旦为 ON，将会保持下去，除非执行了使 OV 为 OFF 的指令。利用这一特性，可以在若干个表达式执行之后集中进行一次溢出判断，而不必在每个表达式之后都要进行一次溢出判断，但事先必须使 OV 复位。

1.9.7 函数的调用形式

所谓函数调用就是在一个表达式中引用一个已经定义了的函数。其一般形式为：**函数名（参数 1，参数 2，……，参数 n）**

函数调用中的**参数**为向函数体中传递的数据，可以是一个常数或变量或表达式甚至是另一个函数的返回值。各个参数之间用逗号隔开。

函数调用中的**参数**与函数定义中的**参数传递元件**必须在个数、数据类型及顺序上严格保持一致。当执行函数调用时，将把参数 1 的值送给函数定义中的参数传递元件 1、参数 2 的值送给参数传递元件 2、……、参数 n 的值送给参数传递元件 n。例如：函数 FUNC1 的定义如下：

FUNC1: FUN I, X AS D0, Y AS D1, Z AS D2

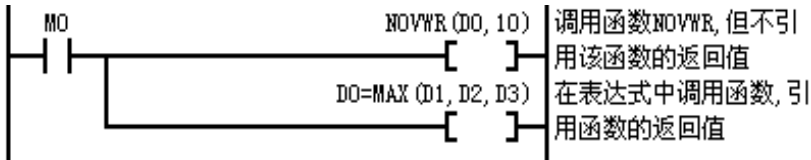
则该函数的调用形式如下：

FUNC1 (10, 20, 30)

执行调用时，将把数值 10 送给 D0，数值 20 送给 D1，数值 30 送给 D2。

一个线圈的操作元件也可以只是一个函数，此时只是调用函数，向函数体中传递数据，而不引用函数的返回值。

例如在表达式 $D0 = \text{MAX}(D1, D2, D3) + \text{MIN}(D4, D5, D6)$ 中包括了两个函数调用：MAX 函数和 MIN 函数。



函数的调用

函数既可以使用线圈调用，也可以使用触点调用，当使用触点调用函数时，若函数类型不为强制执行函数，则调用触点前的逻辑运算结果为 ON 时执行调用，可使用常开触点（函数返回值为 0 触点断开，不为 0 触点接通）和常闭触点（函数返回值为 0 触点接通，不为 0 触点断开）调用函数，触点只能处于逻辑与的位置；若函数类型为强制执行函数，则总是执行调用，可使用常开触点（函数返回值为 0 触点断开，不为 0 触点接通）、常闭触点（函数返回值为 0 触点接通，不为 0 触点断开）、上升沿和下降沿触点（注意操作对象前面要加*）调用函数，触点可处于逻辑取、逻辑与、逻辑或的位置。

1.9.8 自定义函数

自定义函数是由用户根据自己的需要编写的实现特定功能的函数，因此在使用时必须进行定义。

1、函数的定义

使用 FUN 指令来定义一个函数的开始，使用 ENDFUN 指令来表示一个函数的结束。FUN 指令处的标号即为函数名，其格式如下（“[]”内的为可选）：

函数名：FUN 函数类型，参数传递元件 I[] [参数类型]，…，参数传递元件 n[] [参数类型]

函数名是自定义函数的名字。用户选择时应遵从程序标号的要求，如不能有“(”、“)”、“\$”、“#”等特殊符号，也应避免和内部函数或元件（变量）或其他程序标号重名。

函数类型说明了函数返回值的类型，可以是 **I**（整型）或 **L**（长整型）或 **F**（浮点型）或 **E**（强制执行型）。当为强制执行型时，无论包含该函数的表达式接通或断开都执行，但会先把该表达式前的逻辑运算结果压入逻辑堆栈存储器，并把该逻辑堆栈存储器的内容复制给函数中的逻辑堆栈存储器（在函数体内可以使用 MRD 指令读出该逻辑运算结果），待表达式执行完后再弹出。**注意当该函数位于主控电路块或步进电路块中时，若电路块被断开则不执行调用。**

参数传递元件用于向函数中传递数据。被用于参数传递的元件具有动态局部变量的性质，即在函数内该元件内容的改变并不影响函数外该元件的内容。参数传递元件可以是位元件（传递 BOOL 型数据）、字元件（传递整型数据）、双字元件（传递长整型数据）、浮点型元件（传递浮点型数据），可以直接使用元件的名称如 D2.0、D3、LDM256、FDM256 等，也可以同时用 AS 宏指令或冒号“:”进行标识符的预定义。参数传递元件区为 D0-D15、DM256-DM383。

参数类型说明了该参数的输入、输出等类型。若省略，则该参数为输入型（IN）参数，在函数调用中传递的是数值。若为“**IO**”，则该参数为输入/输出型（IN_OUT）参数。若为“**O**”，则该参数为输出型（OUT）参数。若为“**A**”，则该参数为地址传递型（ADDR）参数，在函数调用中，若该参数为单个变量（包含变址寻址形式，例如 DM500、DM500[8]、DM500[D0]、DM500[DM300]），则传递的是该变量的实际地址（在上例中分别传递的数值为 500、508、500+D0、500+DM300），若该参数为表达式，则传递的是该表达式的计算数值。

例如：

FUN I, D0, D1|A, D2|O 或者

FUN I, X AS D0, Y AS D1|A, Z AS D2|O 或者

FUN I, X:D0, Y:D1|A, Z:D2|O

则 X 为“IN”型参数，Y 为“ADDR”型参数，Z 为“OUT”型参数，函数返回值为整型。

自定义函数在语句表中的一般形式如下：

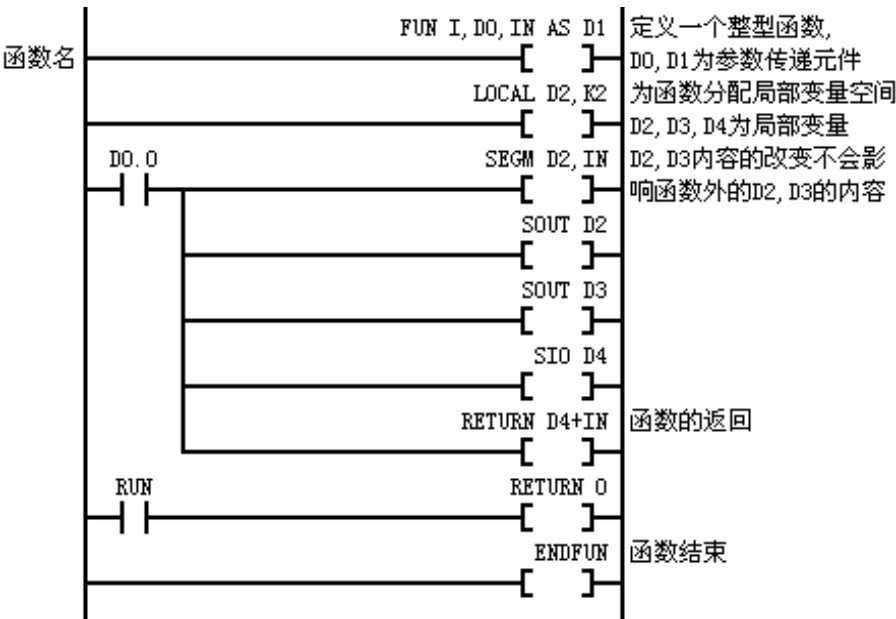
函数名: FUN 函数类型, 参数传递元件 1, 参数传递元件 2, …, 参数传递元件 n
函数的局部变量空间分配
函数体指令
ENDFUN

调用该函数的形式如下：

函数名 (参数 1, 参数 2, …, 参数 n)

当执行函数调用时，将把参数 1 的值送给参数传递元件 1、参数 2 的值送给参数传递元件 2、…、参数 n 的值送给参数传递元件 n。若参数类型为 IN_OUT 或 OUT 型，则函数返回时将把对应的参数传递元件的值再送给对应的参数（该参数必须是一个变量，并且不能与各个参数传递元件相同）。

在梯形图中，定义的函数名应该位于梯形图的标号区，其他格式同上。例：



自定义函数在梯形图中的一般形式

2、为函数分配动态局部变量空间

在函数内，若没有为函数分配动态局部变量空间，则函数内所有使用到的元件均为全局变量（参数传递元件已被定义为动态局部变量，故除外），当这些元件的内容被改变后，将会影响到函数外其他使用这些元件的程序。若要使函数内的元件（变量）内容的改变不会影响到函数外的程序，则必须为函数分配动态局部变量空间，即把函数内用到的元件定义为动态局部变量。当在函数内被分配为动态局部变量的元件的内容被改变后，并不影响函数外这些元件的内容。

但要注意每次进入函数时这些动态局部变量的内容是随机的（参数传递元件除外），若要希望进入函数时局部变量的内容为上次退出函数时的内容，则应使用静态局部变量。

使用 LOCAL 指令可为函数分配动态局部变量空间，有 2 种格式：

格式 1:

LOCAL IN, Kx

操作数	可使用的元件
IN	D0~D15、DM256~DM383
Kx	常数 (D: 0~15, DM: 0~127)

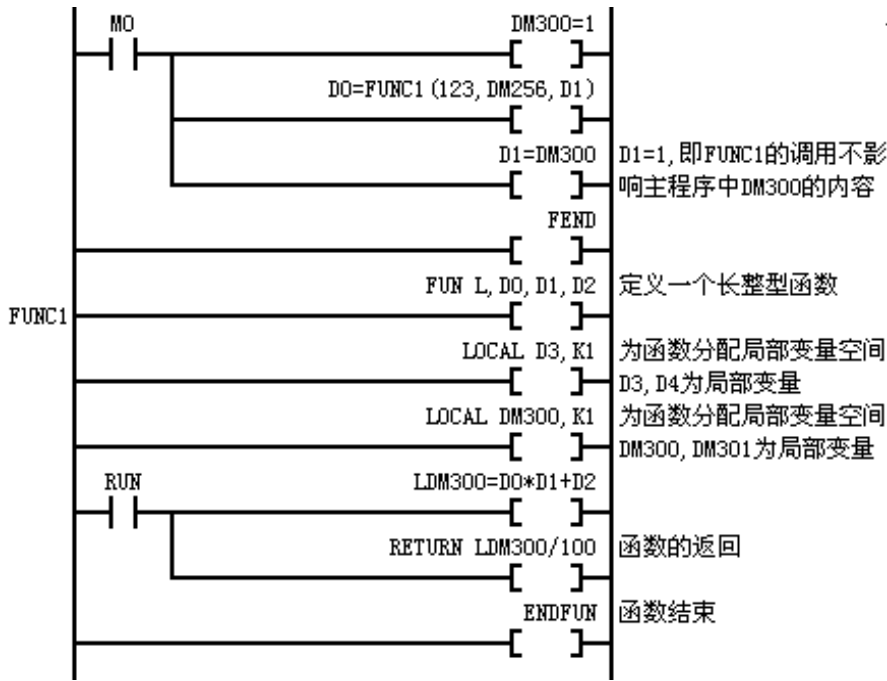
说明:

把从 IN 到 IN[x]共 x+1 个元件作为函数的动态局部变量空间。1 个 LOCAL 指令最多可为函数分配 16 个 D 动态局部变量或 128 个 DM 动态局部变量。例如:

LOCAL D2, K3 是把 D2、D3、D4、D5 作为函数的动态局部变量。

LOCAL DM256, K1 是把 DM256、DM257 作为函数的动态局部变量。

在一个函数内可以使用多个 LOCAL 指令为函数分配多种动态局部变量。但 LOCAL 指令必须用在函数定义指令 FUN 之后、函数内的其他指令之前, 否则可能导致不可预料的结果。



为函数分配动态局部变量空间

当使用数据存储器 DM 作为函数的动态局部变量空间时, 推荐仅使用 DM256~DM383 之间的单元。

格式 2:

LOCAL 变量名 1, 变量名 2, ..., 变量名 n AS 直接地址

操作数	可使用的元件
变量名 1~变量名 n	为变量的预定义标识符
直接地址	D0~D15、DM256~DM383、LDM256~LDM382、FDM256~FDM382

说明:

“变量名”也可数组格式。每个 LOCAL 指令最多可分配 16 个 D 整型变量或 128 个 DM 整型变量或 64 个长整型变量或 64 个浮点型变量（若含有数组，则应按数组的元素个数计算）。

“直接地址”为这些变量的起始地址，也即为“变量名 1”的元件地址，其他变量按顺序依次分配。当为 DM 或 LDM 或 FDM 时，推荐仅在 DM256~DM383 或 LDM256~LDM382 或 FDM256~FDM382 之间分配。

例如:

LOCAL VAR1, VAR2, VAR3 AS D2

则 VAR1 被分配为 D2，VAR2 被分配为 D3，VAR3 被分配为 D4。且均为整型变量。

LOCAL VAR1, VAR2, VAR3 AS DM256

则 VAR1 被分配为 DM256，VAR2 被分配为 DM257，VAR3 被分配为 DM258。且均为整型变量。

LOCAL VAR1, VAR2, VAR3 AS LDM256

则 VAR1 被分配为 LDM256，VAR2 被分配为 LDM258，VAR3 被分配为 LDM260。且均为长整型变量。

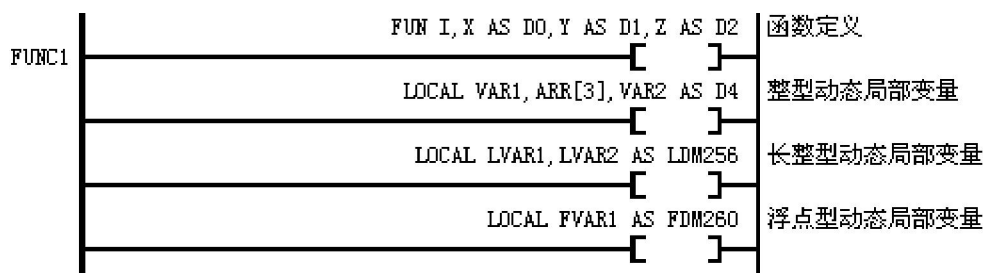
LOCAL VAR1, VAR2, VAR3 AS FDM256

则 VAR1 被分配为 FDM256，VAR2 被分配为 FDM258，VAR3 被分配为 FDM260。且均为浮点型变量。

LOCAL VAR1, ARR[3], VAR2 AS D2

则 VAR1 被分配为 D2，ARR[0]被分配为 D3，ARR[1]被分配为 D4，ARR[2]被分配为 D5，VAR2 被分配为 D6。且均为整型变量。

在梯形图中的例子如下:



函数的动态局部变量分配例子

3、函数的返回

函数的返回使用 RETURN 指令。RETURN 指令的格式为：

RETURN 表达式

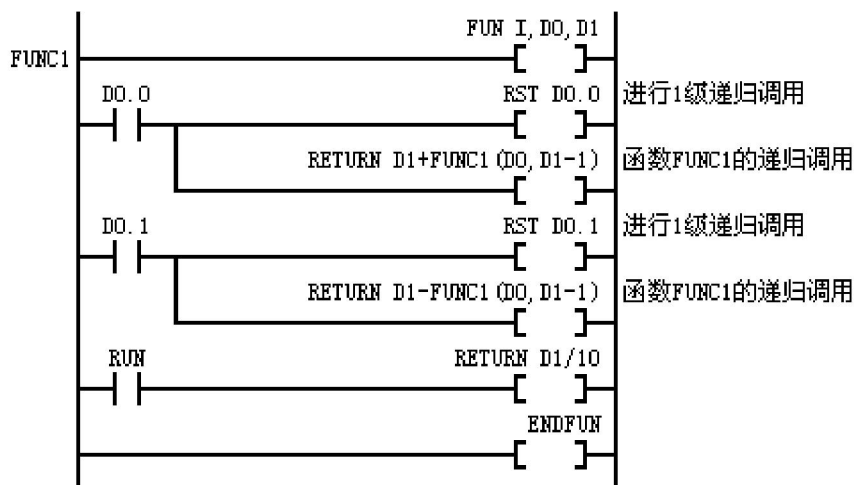
若 RETURN 指令被接通, 则计算出表达式的值并把该值作为函数的返回值, 同时执行函数返回并释放函数所用到的数据堆栈空间。若 RETURN 指令被断开, 则该 RETURN 指令不被执行。

表达式值的数据类型应与 FUN 指令中定义的函数类型一致，否则会引起编译错误。

只有执行 RETURN 指令才能使函数返回一个确定的值。函数也可由 ENDFUN 指令返回，此时，函数的返回值是不确定的。

4、函数的递归调用

如果在一个函数的内部又间接或直接地调用该函数本身，称为函数的递归调用。EasyLad 梯形图语言的函数均可进行函数的递归调用，但递归调用也为函数的嵌套，因此应注意最多允许 8 级函数或子程序嵌套的限制。



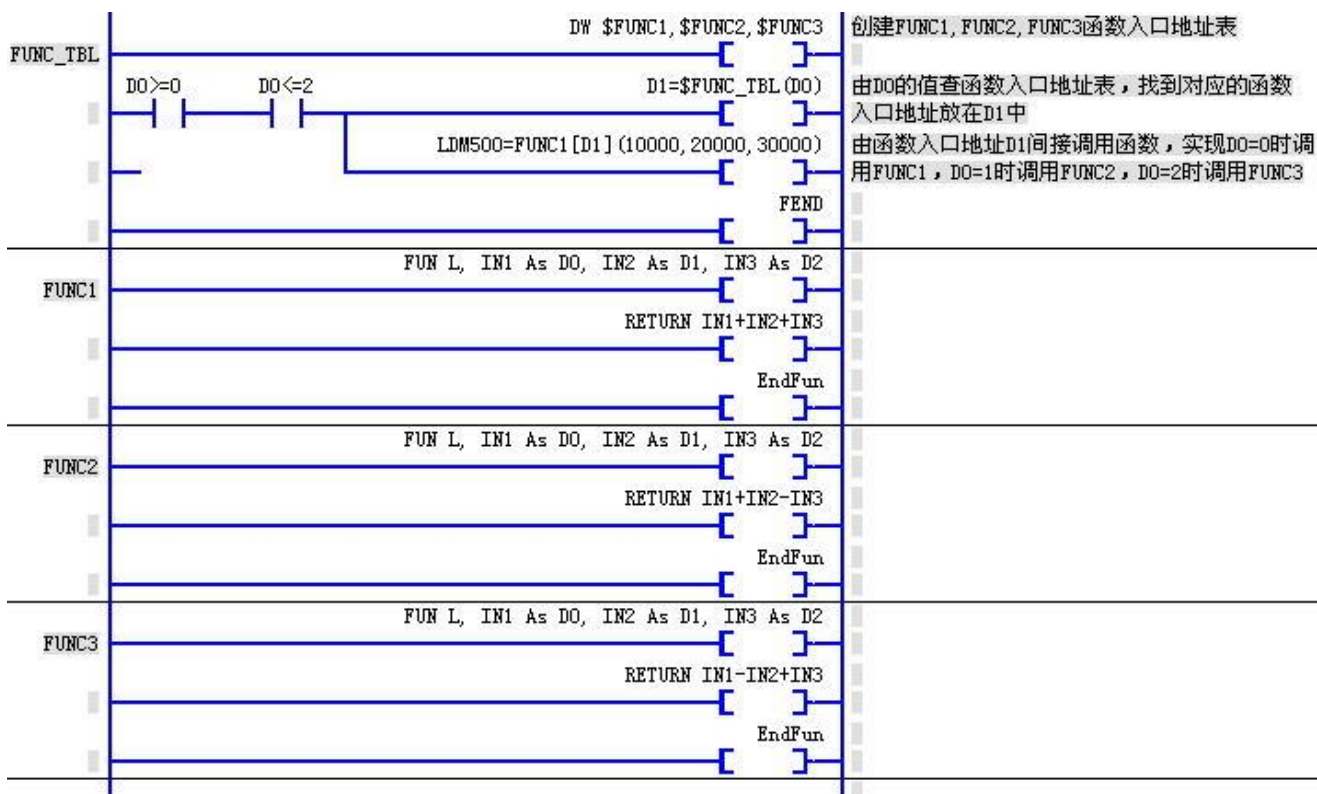
函数的递归调用

5、函数的间接调用

可以使用保存于数据寄存器 D0~D15 中函数入口地址来间接调用函数，间接调用函数的格式为：

参照函数名[间接地址寄存器] (参数 1, 参数 2, ……., 参数 n)

其中参照函数名为用于向函数传递参数的参照函数，通过间接地址调用的函数的参数定义必须和参照函数的相同。间接地址寄存器为保存有要调用函数的入口地址，可为 D0~D15 或定义为这些元件的符号名。函数的入口地址可使用“\$函数名”获得，例如 D0=\$FUNC2。函数的间接调用例子如下：



1.9.9 使用函数参数表编辑自定义函数的参数

用户也可以使用函数参数表来编辑或修改自定义函数的类型、参数、注释、分配动态局部变量空间，使函数的定义变得表格化，从而使函数的定义、编辑、维护更直观，更方便。

函数参数表的样式图如下：

函数定义

函数名: DisVal

函数类型: I 整型

	符号名	变量类型	数据类型
1	Val	IN	INT
2	DP	IN	INT
3	BufSel	IN	INT
4	Temp[3]	TEMP	INT

描述

送数字显示值。
 功能：把某个数值送到数显窗的显示缓冲区中显示出来。
 Val：数字显示的整型值（-1999~9999）。
 DP：小数点位置，为0：无小数点，为1：1位小数，为2：2位小数，为4：3位小数。
 BufSel：显示缓冲区选择，为0：选择数显1的显示缓冲区，为1：选择数显2的显示缓冲区。

插入 删除 确定 取消

函数参数表样式图

在梯形图编程软件 EasyLad 中，点击“工具”菜单中的“函数参数表编辑”菜单项（或点鼠标右键→弹出式菜单→函数参数表）即可打开函数参数表。

在函数参数表中：

“函数名”：用户要自定义的函数的名称。

“函数类型”：函数的返回值的类型，可以是“I 整型”、“L 长整型”、“F 浮点型”、“E 强制执行型”。

“参数表”：函数的输入输出参数和临时变量（动态局部变量）表。

“符号名”：用户要定义的函数的输入输出参数或临时变量的名称。

“变量类型”：该符号名的变量类型。可以是“IN”——输入型参数、“I

N_OUT”—— 输入输出型参数、“OUT”—— 输出型参数、“ADDR”—— 地址传递型参数、“TEMP”—— 临时变量（动态局部变量）。

“数据类型”：该符号名的数据类型。BOOL、INT、WORD 类型分配于 D 0~D15 中，其他类型分配于 DM256~DM383 中。

“描述”：该函数的注释说明。该内容可在“函数参照”中显示出来。

“插入”：在参数表中当前的位置插入一个新的符号。

“删除”：删除参数表中当前位置的符号或所选中的一块符号。

“确定”：确认修改后的函数参数表有效，同时关闭函数参数表，回到梯形图编辑界面。

“取消”：不改变函数参数表并关闭函数参数表，回到梯形图编辑界面。

自定义函数应用例子——数据块比较函数

函数名：BCMP

函数功能：比较两个位于数据存储器 DM 区的两个数据块是否相等。

函数类型：整型

函数原型：BCMP: FUN I, i AS D0, j AS D1, n AS D2

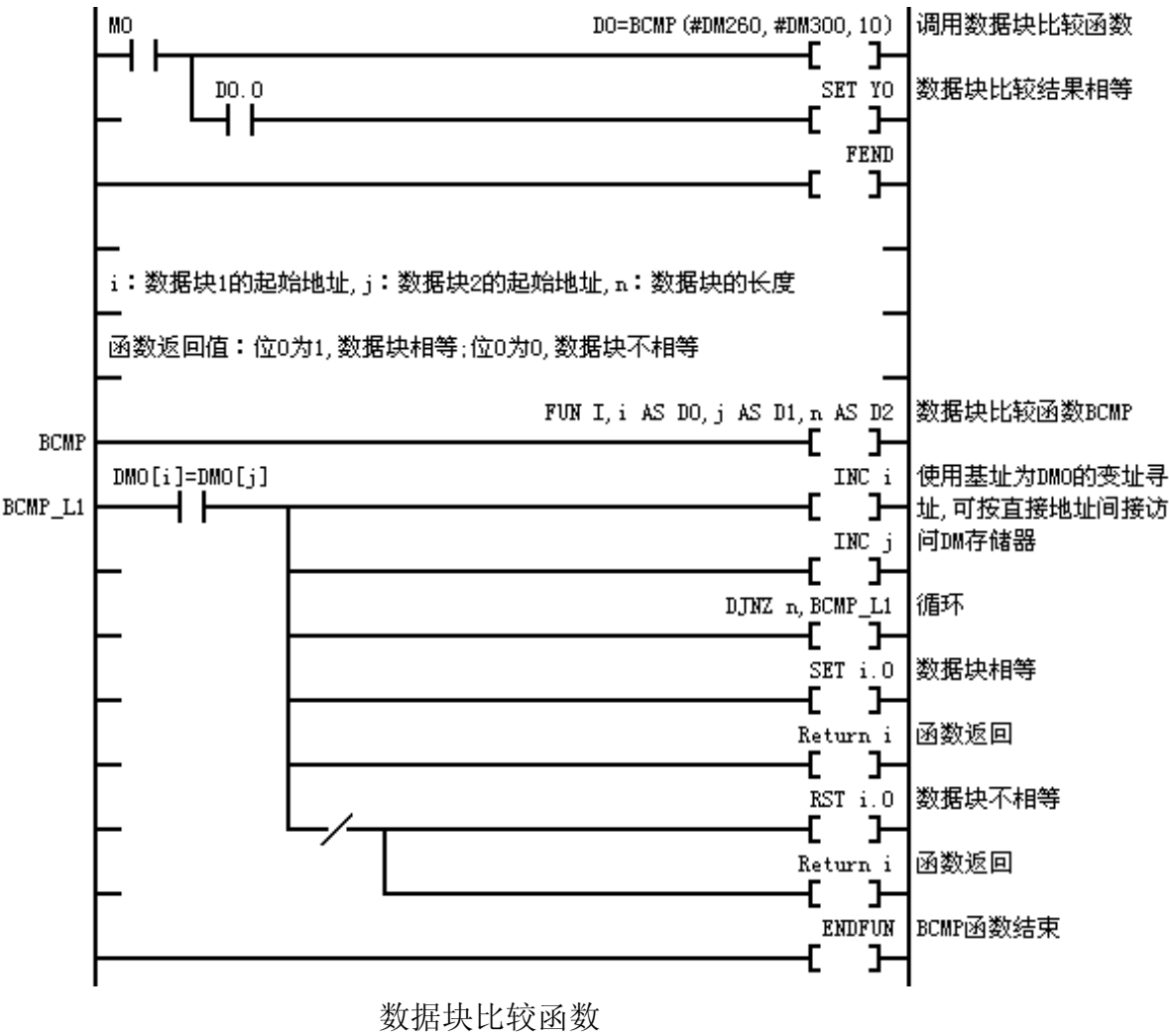
输入参数：i：数据块 1 的起始地址

j：数据块 2 的起始地址

n：数据块的长度

返回值：若位 0 为 1，则表示两个数据块相等；若位 0 为 0，则表示两个数据块不相等。

数据块比较函数的梯形图程序如下：



1.10 内部函数

内部函数由系统所提供，不需要用户定义，可直接调用。

1.10.1 变址函数

变址函数是一类特殊的函数，它以一个数据存储器（变量）或表格标号来作为函数名，用于以函数的参数为变址来间接访问数据存储器（变量）或表格。

1、数据存储器（变量）变址函数

其调用形式如下：

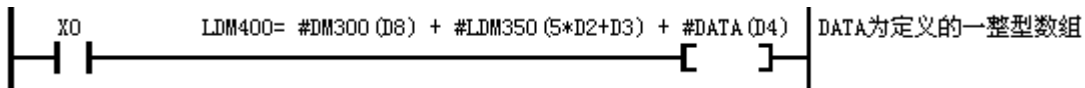
#数据存储器名(参数)

@数据存储器名(参数)

使用前缀“#”或“@”，表示为一个变量变址函数。数据存储器名可以是字 DM 或双字 LDM（!LDM）或浮点数 FDM，或者是被定义为这些元件的符号名，以及他们的变址寻址格式，如：#DM300(D8)、#LDM300(DM400+DM401)、#!LDM300(DM400)、#FDM280(DM400)、#VAR1(DM405)、@DM0(D6)、@LDM300(DM400)、@VAR1(DM400)、#VAR1[D0](DM400)等。其参数可以是一个常数或变量或表达式或另一个函数的返回值，但必须是整型数据且不能是负值，否则将返回一个不确定的值。该类函数的功能如下：

若数据存储器为字 DM，则#变址函数和@变址函数都是把函数参数加上数据存储器的编号（地址）来作为一个新的元件编号（地址），并把这个新的元件的内容作为函数的返回值，例如#DM300(10)，其函数的返回值为 DM310 的内容。

若数据存储器为双字 LDM 或 FDM，则#变址函数是把函数参数先乘 2 再加上数据存储器的编号（地址）来作为一个新的元件编号（地址），并把这个新的元件的内容作为函数的返回值，例如#LDM300(3)，先把参数 3 乘 2 再加上 300，实际是把 LDM306 的值作为函数的返回值。而@变址函数是把函数参数加上数据存储器的编号（地址）来作为一个新的元件编号（地址），并把这个新的元件的内容作为函数的返回值，例如@LDM300(3)，是把 LDM303 的值作为函数的返回值。



数据存储器（变量）的变址函数的使用

若函数参数为数组索引，则应使用#变址函数，若函数参数为元件地址，则应使用@变址函数。

注：变址函数可在表达式=号的左边用作表达式赋值，例如#LDM300(3)=123456

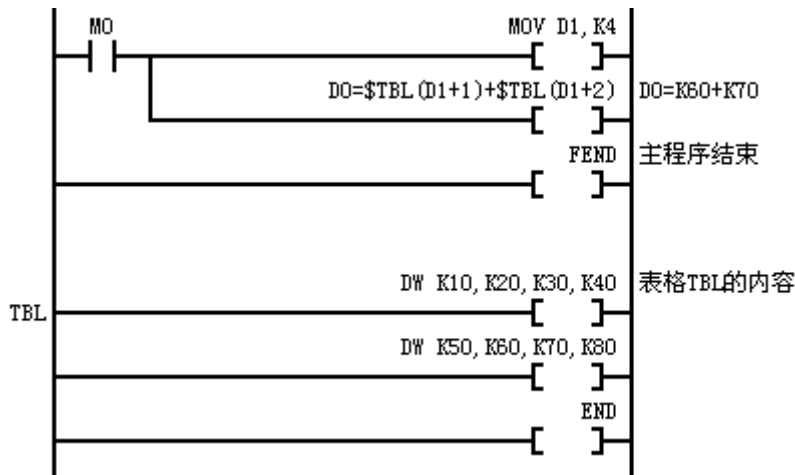
2、表格标号变址函数

其调用形式如下：

\$表格标号(参数)

使用前缀“\$”，表示为一个表格变址函数。表格标号必须是定义表格的一个标号。其参数可以是一个常数或变量或表达式甚至是另一个函数的返回值，但必须是整型数据且不能是负值，否则将返回一个不确定的值。执行该函数时，把表格标号所对应的程序步数（地址）加上函数参数的值来作为一个新的程序步数（地址），并把这个新的程序步数处的内容（以字为单位）作为函数的返回值。

例如：若表格标号 TBL 所对应的程序步数为 500，则函数\$TBL（10）的返回值为程序步数 510 处的内容（1 个字）。



表格变址函数的使用

1.10.2 数据类型转换函数（L / F）

1、浮点型转换为整型（或长整型）函数 L

函数定义：

FUN L, Val As FDM256

函数功能：

把某个浮点数转换为整型（长整型）并作为函数的返回值。

输入参数：

Val: 要转换的浮点数。

返回值：

转换后的整型（长整型）数据。

调用形式:

L(Val)

例:

DM300 = L(FDM400) + DM301

2、整型（或长整型）转换为浮点型函数 F

函数定义:

FUN F, Val As LDM256

函数功能:

把某个整型（长整型）数转换为浮点数并作为函数的返回值。

输入参数:

Val: 要转换的整型（长整型）数。

返回值:

转换后的浮点型数据。

调用形式:

F(Val)

例:

FDM300 = F(DM400*10) + 1.23

1.10.3 取绝对值函数（IABS / FABS）

1、整型（或长整型）取绝对值函数 IABS

函数定义:

FUN L, Val As LDM256

函数功能:

返回某个整型（长整型）数值的绝对值，若输入参数为正数，则进位继电器 CF 被清 0，若输入参数为负数，则 CF 被置 1。

输入参数:

Val: 要取绝对值的整型（长整型）数。

返回值:

Val 的绝对值。

调用形式:

IABS(Val)

例:

DM300 = IABS(DM400)

2、浮点数取绝对值函数 FABS

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型（实型）数值的绝对值，若输入参数为正数，则进位继电器 CF 被清 0，若输入参数为负数，则 CF 被置 1。

输入参数:

Val: 要取绝对值的浮点型（实型）数。

返回值:

Val 的绝对值。

调用形式:

FABS(Val)

例:

FDM300 = FABS(FDM400)

1.10.4 取负函数（INEG / FNEG）

1、整型（或长整型）取负函数 INEG

函数定义:

FUN L, Val As LDM256

函数功能:

把某个整型（长整型）数值取负后作为函数的返回值。

输入参数:

Val: 要取负的整型（长整型）数。

返回值:

Val 的取负值。

调用形式:

INEG(Val)

例:

DM300 = INEG(DM400)

2、浮点数取负函数 FNEG

函数定义:

FUN F, Val As FDM256

函数功能:

把某个浮点型（实型）数值取负后作为函数的返回值。

输入参数:

Val: 要取负的浮点型（实型）数。

返回值:

Val 的取负值。

调用形式:

FNEG(Val)

例:

FDM300 = FNEG(FDM400)

1.10.5 长整数高低字交换函数（DSWAP）

函数定义:

FUN L, Val As LDM256

函数功能:

把输入的长整数（双字）的高字和低字交换位置后作为函数的返回值。

输入参数:

Val: 要高低字交换位置的长整型（双字）数。

返回值:

高低字交换位置后的长整型（双字）数。

调用形式:

DSWAP(Val)

例:

LDM400 = DSWAP(LDM300) 如果 LDM300 中的值为 H12345678, 则执行该式子后
LDM400 中的值为 H56781234

1.10.6 浮点数开平方函数 (SQRT)

函数定义:

FUN L, Val As FDM256

函数功能:

返回某个浮点型 (实型) 数值的开平方值。

输入参数:

Val: 要开平方的浮点型 (实型) 数值。

返回值:

Val 的开平方值。

调用形式:

SQRT(Val)

说明:

其输入参数必须为浮点型 (实型) 数据, 若小于 0, 则溢出继电器 OV 被置为 ON。但不会使 OV 为 OFF。

例:

FDM300 = SQRT(FDM400)

1.10.7 读线圈和写线圈函数 (RdCoil / WrCoil/ RdLCoil / WrLCoil)

1、读短地址线圈函数 RdCoil

函数定义:

FUN I, CoilAddr As D0, Len As D1

函数功能:

按用户指定的起始位置和长度, 把某段继电器 (M、Y、X、DMx.y) 的状态作为函数的返回值, 编号小的线圈将放在返回值的低位, 若线圈数量不够 16 个, 则返回值的空余位被清 0。

输入参数:

CoilAddr: 该段继电器的起始地址 (编号)。若继电器位于 Y 区, 则 Y 继电器的编号应再加上 208; 若继电器位于 X 区, 则 X 继电器的编号应再加上 256; 若继电器位于 DMx.y 区 ($x < 4096$, 超出该范围则不能使用该函数), 则继电器的编号为 $x \times 16 + y$ 。

Len: 该段继电器的长度 (1~16)。

返回值:

把该段继电器按位组成的整型值, 编号小的继电器位于该值的低位。

调用形式:

RdCoil(CoilAddr, Len)

例:

DM300 = RdCoil(#M10,12) 则 DM300 的位 0 为 M10、位 1 为 M11、……、位 11 为 M21、位 12~15 被清 0。

2、写短地址线圈函数 WrCoil

函数定义:

FUN F, Val As D0, CoilAddr As D1, Len As D2

函数功能:

把某个数值按位写入到某段继电器 (M、Y、X、DMx.y) 中, 数值的低位写入到编号小的继电器中。

输入参数:

Val: 要写入的整型数值。

CoilAddr: 该段继电器的起始地址 (编号)。若继电器位于 Y 区, 则 Y 继电器的编号应再加上 208; 若继电器位于 X 区, 则 X 继电器的编号应再加上 256; 若继电器位于 DMx.y

区 ($x < 4096$, 超出该范围则不能使用该函数), 则继电器的编号为 $x \times 16 + y$ 。

Len: 该段继电器的长度 (1~16)。

返回值:

无。

调用形式:

WrCoil(Val, CoilAddr, Len)

例:

WrCoil(D0, #M10, 12) 则 D0 的位 0 写入 M10、位 1 写入 M11、……、位 11 写入 M21, 其他继电器不受影响。

3、读长地址线圈函数 RdLCoil

函数定义:

FUN I, CoilAddr As LDM256, Len As D1

函数功能:

按用户指定的起始位置和长度, 把某段继电器 (M、Y、X、DMx.y) 的状态作为函数的返回值, 编号小的线圈将放在返回值的低位, 若线圈数量不够 16 个, 则返回值的空余位被清 0。

输入参数:

CoilAddr: 该段继电器的起始地址 (编号)。若继电器位于 Y 区, 则 Y 继电器的编号应再加上 208; 若继电器位于 X 区, 则 X 继电器的编号应再加上 256; 若继电器位于 DMx.y 区, 则继电器的编号为 $x \times 16 + y$ 。

Len: 该段继电器的长度 (1~16)。

返回值:

把该段继电器按位组成的整型值, 编号小的继电器位于该值的低位。

调用形式:

RdLCoil(CoilAddr, Len)

例:

DM300 = RdLCoil(#M10, 12) 则 DM300 的位 0 为 M10、位 1 为 M11、……、位 11 为 M21、位 12~15 被清 0。

4、写长地址线圈函数 WrLCoil

函数定义:

FUN F, Val As D0, CoilAddr As LDM256, Len As D2

函数功能:

把某个数值按位写入到某段继电器 (M、Y、X、DMx.y) 中, 数值的低位写入到编号小的继电器中。

输入参数:

Val: 要写入的整型数值。

CoilAddr: 该段继电器的起始地址 (编号)。若继电器位于 Y 区, 则 Y 继电器的编号应再加上 208; 若继电器位于 X 区, 则 X 继电器的编号应再加上 256; 若继电器位于 DMx.y 区, 则继电器的编号为 $x \times 16 + y$ 。

Len: 该段继电器的长度 (1~16)。

返回值:

无。

调用形式:

WrLCoil(Val, CoilAddr, Len)

例:

WrLCoil(D0, #M10, 12) 则 D0 的位 0 写入 M10、位 1 写入 M11、……、位 11 写入 M21, 其他继电器不受影响。

1.10.8 编程通讯口设置函数 (CommSet)*函数定义:*

FUN I, Val As D0

函数功能:

根据输入的参数设置编程通讯口格式、初始化编程通讯口。

输入参数:

Val: 要设置的通讯格式。

调用形式:

CommSet(Val)

说明:

若输入参数的位 7 为 0, 则由协议自动设置校验位(编程协议或自由协议无校验位, Modbus 协议偶校验); 若位 7 为 1, 则由程序设置校验位, 此时若位 6 为 0 选偶校验, 位 6 为 1

选无校验位。

若输入参数的位 5 为 0，则为系统监控通讯模式；若位 5 为 1，则为完全自由通讯模式。

若输入参数的位 1 位 0 为 00，则波特率为 19200；若位 1 位 0 为 01，则波特率为 9600；
位 1 位 0 为 10，则波特率为 38400；位 1 位 0 为 11，则波特率为 115200。

输入参数的高字节(位 8~位 15)为完全自由通讯模式下的起始字符(0 表示无起始字符)，
当接收到起始字符时将使接收缓冲区指针复位，允许接收。

注：当从完全自由通讯模式切换为系统监控通讯模式后要重新设置编程口通讯地址。

例： CommSet(H0021)

1.10.9 编程口 ModBus-RTU 通讯函数

该组通讯函数必须在系统监控通讯模式下使用

1、网络读（读寄存器）函数 NETR

函数定义：

FUN I, CommAddr As D0, DM_Addr As D1, Len As D2

函数功能：

按指定的从机地址，读取从机中的某块寄存器的值（使用功能码“03”），并存入到本机中指定的 DM 数据块中。

输入参数：

CommAddr: 从机地址，不能为 2，为 0 表示广播地址。

DM_Addr: 本机中 DM 数据块（数组）的地址（DM256 以后的单元），数组中的[0]为从机寄存器地址，[1]~…为本机中存储读入的从机数据块的数据，读入寄存器个数为 **Len** - 1 个字。

Len: 本机中 DM 数据块（数组）的长度，最大值为 33。

返回值：

无

调用形式：

NETR(CommAddr, DM_Addr, Len)

说明：

通讯格式：1 个起始位，8 个数据位，偶校验，1 个停止位。

若特殊继电器 CBUSY 为 ON，则该函数不会执行。当特殊继电器 CBUSY 为 OFF 时，若调用该函数成功，则 CBUSY 变为 ON，表示本机作为主机正在与从机进行通讯，当通讯完成后 CBUSY 自动被复位；若调用不成功，则 CBUSY 依旧保持为 OFF，此时通常是由于其他主机正在对本机进行访问或错误的函数参数引起的。

例：

NETR(1,#DM300,8) 其中 DM300 为从机寄存器地址，DM301~DM307 存储读入的从机数据块的数据（读入 7 个字）。

2、网络写（写多寄存器）函数 NETW

函数定义：

FUN I, CommAddr As D0, DM_Addr As D1, Len As D2

函数功能:

按指定的从机地址，把本机中的 DM 数据块的数值写入到从机中的某块寄存器中（使用功能码“16”）。

输入参数:

CommAddr: 从机地址，不能为 2，为 0 表示广播地址。

DM_Addr: 本机中 DM 数据块（数组）的地址（DM256 以后的单元），数组中的[0]为从机寄存器地址，[1]~…为主机中要写到从机中的数据，写入寄存器个数为 **Len - 1** 个字。

Len: 本机中 DM 数据块（数组）的长度，最大值为 33。

返回值:

无。

调用形式:

NETR(CommAddr, DM_Addr, Len)

说明:

通讯格式：1 个起始位，8 个数据位，偶校验，1 个停止位。

若特殊继电器 CBUSY 为 ON，则该函数不会执行。当特殊继电器 CBUSY 为 OFF 时，若调用该函数成功，则 CBUSY 变为 ON，表示本机作为主机正在与从机进行通讯，当通讯完成后 CBUSY 自动被复位；若调用不成功，则 CBUSY 依旧保持为 OFF，此时通常是由于其他主机正在对本机进行访问或错误的函数参数引起的。

例:

NETW(1,#DM300,8) 其中 DM300 为从机寄存器地址，DM301~DM307 为主机中要写到从机中的数据（共写入 7 个字）。

3、读线圈状态函数 MRCS

函数定义:

FUN I, CommAddr As D0, DM_Addr As D1, CoilNum As D2

函数功能:

按指定的从机地址，读取从机中的某些线圈的状态（使用功能码“01”），并存入到本机中指定的 DM 存储器中。

输入参数:

CommAddr: 从机地址，不能为 2，为 0 表示广播地址。

DM_Addr: 本机中 DM 数据块（数组，占 2 个字）的地址（DM256 以后的单元），数

组中的[0]为从机线圈块首地址，[1]为本机中存储读入的线圈的状态。

CoilNum: 要读入的线圈的个数（最多为 16 个）。

返回值:

无

调用形式:

MRCS(CommAddr, DM_Addr, CoilNum)

说明:

通讯格式：1 个起始位，8 个数据位，偶校验，1 个停止位。

若特殊继电器 CBUSY 为 ON，则该函数不会执行。当特殊继电器 CBUSY 为 OFF 时，若调用该函数成功，则 CBUSY 变为 ON，表示本机作为主机正在与从机进行通讯，当通讯完成后 CBUSY 自动被复位；若调用不成功，则 CBUSY 依旧保持为 OFF，此时通常是由于其他主机正在对本机进行访问或错误的函数参数引起的。

例:

MRCS(1,#DM300,10) 其中 DM300 为从机线圈块首地址，DM301 存储读入的 10 个线圈的状态。

4、强制多线圈函数 MFMC

函数定义:

FUN I, CommAddr As D0, DM_Addr As D1, CoilNum As D2

函数功能:

按指定的从机地址，强制从机中的某些线圈为指定的状态（使用功能码“15”）。

输入参数:

CommAddr: 从机地址，不能为 2，为 0 表示广播地址。

DM_Addr: 本机中 DM 数据块（数组，占 2 个字）的地址（DM256 以后的单元），数组中的[0]为从机线圈块首地址，[1]为要把从机中的某些线圈强制为的状态，位 0 对应于从机线圈块的首个线圈。

CoilNum: 要强制的线圈的个数（最多为 16 个）。

返回值:

无

调用形式:

MFMC(CommAddr, DM_Addr, CoilNum)

说明:

通讯格式: 1 个起始位, 8 个数据位, 偶校验, 1 个停止位。

若特殊继电器 CBUSY 为 ON, 则该函数不会执行。当特殊继电器 CBUSY 为 OFF 时, 若调用该函数成功, 则 CBUSY 变为 ON, 表示本机作为主机正在与从机进行通讯, 当通讯完成后 CBUSY 自动被复位; 若调用不成功, 则 CBUSY 依旧保持为 OFF, 此时通常是由于其他主机正在对本机进行访问或错误的函数参数引起的。

例:

MFMC(1,#DM300,10) 其中 DM300 为从机线圈块首地址, DM301 中的位 0--位 9 为从机中的 10 个线圈要强制为的状态。

5、强制单线圈函数 MFSC*函数定义:*

FUN I, CommAddr As D0, CoilAddr As D1, Val As D2

函数功能:

按指定的从机地址, 强制从机中的某个线圈为指定的状态 (使用功能码 “05”)。

输入参数:

CommAddr: 从机地址, 不能为 2, 为 0 表示广播地址。

CoilAddr: 从机中要强制的线圈的地址。

Val: 要强制的状态, 0 为 OFF, 1 为 ON。

返回值:

无。

调用形式:

MFSC(CommAddr, CoilAddr, Val)

说明:

通讯格式: 1 个起始位, 8 个数据位, 偶校验, 1 个停止位。

若特殊继电器 CBUSY 为 ON, 则该函数不会执行。当特殊继电器 CBUSY 为 OFF 时, 若调用该函数成功, 则 CBUSY 变为 ON, 表示本机作为主机正在与从机进行通讯, 当通讯完成后 CBUSY 自动被复位; 若调用不成功, 则 CBUSY 依旧保持为 OFF, 此时通常是由于其他主机正在对本机进行访问或错误的函数参数引起的。

例:

MFSC(1,100,M10) 把从机中线圈 100 强制为本机中 M10 的状态。

6、调整单寄存器函数 MPSR

函数定义:

FUN I, CommAddr As D0, RegAddr As D1, Val As D2

函数功能:

按指定的从机地址，把指定的数据写入到从机中某个寄存器中（使用功能码“06”）。

输入参数:

CommAddr: 从机地址，不能为 2，为 0 表示广播地址。

RegAddr: 从机中要调整的寄存器的地址。

Val: 要写入的值。

返回值:

无。

调用形式:

MPSR(CommAddr, RegAddr, Val)

说明:

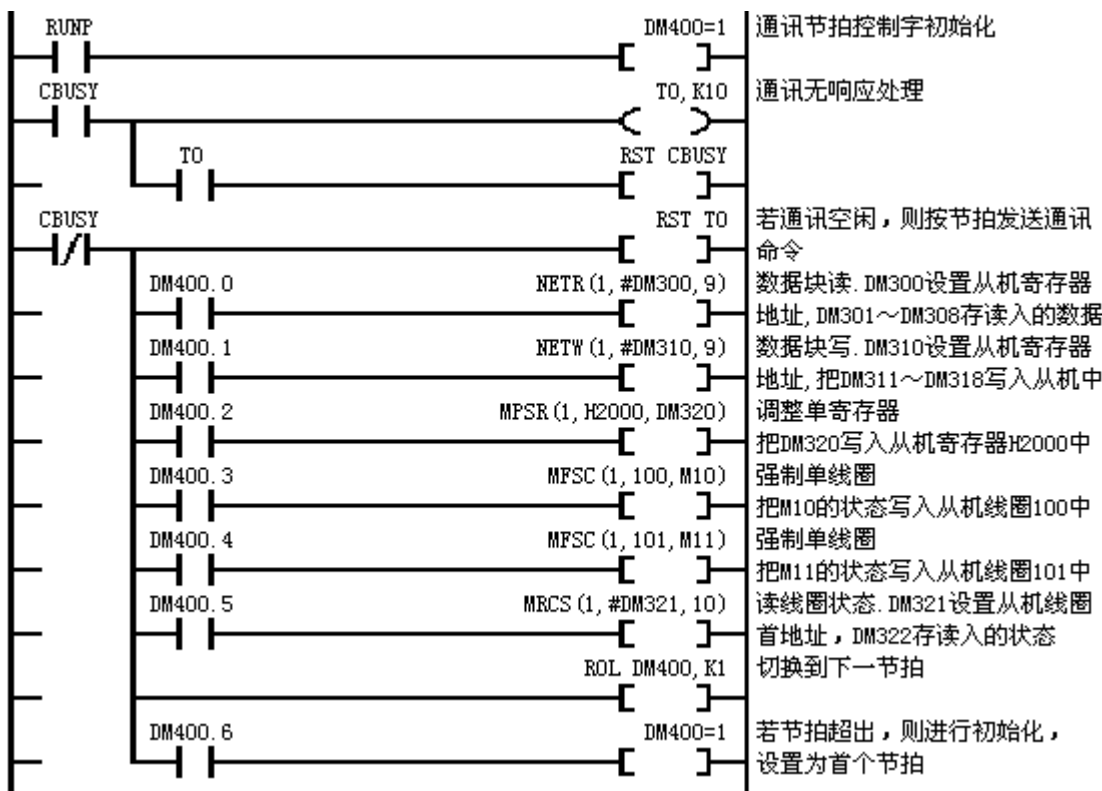
通讯格式：1 个起始位，8 个数据位，偶校验，1 个停止位。

若特殊继电器 CBUSY 为 ON，则该函数不会执行。当特殊继电器 CBUSY 为 OFF 时，若调用该函数成功，则 CBUSY 变为 ON，表示本机作为主机正在与从机进行通讯，当通讯完成后 CBUSY 自动被复位；若调用不成功，则 CBUSY 依旧保持为 OFF，此时通常是由于其他主机正在对本机进行访问或错误的函数参数引起的。

例:

MPSR(1,300,DM400) 把本机中 DM400 的值写入到从机中寄存器 300 中。

ModBus-RTU 主机通讯的梯形图例子：



1.10.10 编程口自由通讯发送函数

该组通讯函数在系统监控通讯模式下被跳过不执行，在完全自由通讯模式下才执行。

系统为自由口通讯设置了 128 个字节的发送缓冲区（与接收缓冲区共用）和 1 个发送缓冲区地址指针，发送缓冲区的地址编号为 0~127。当要发送某些数据时，要按以下步骤来处理：

- ① 首先使用 TxCon(0)函数使发送缓冲区地址指针复位为 0。
- ② 使用 TxChar、TxWord、TxStr 等函数向发送缓冲区写入要发送的数据，发送时按照“先入先出”顺序来发送，即先写入的数据先被发送出去。
- ③ 使用 TxCon(1)函数来启动发送，开始发送数据。

判断特殊继电器 CBUSY，则可判断这些数据是否发完。开始发送时，CBUSY 置为 ON，发送完后 CBUSY 复位为 OFF。

1、自由口发送控制函数 TxCon

函数定义：

FUN I, Val As D0

函数功能：

根据输入参数的值，执行不同的操作：

当输入参数为负数，则函数返回发送缓冲区指针的值。

当输入参数为 0，则使发送缓冲区指针复位为 0，同时溢出继电器 OV 复位，接收缓冲区指针置为-2（禁止接收状态，若无起始字符模式，则发送完数据后自动被允许）。

当输入参数为正数，若 CBUSY 为 OFF，且发送缓冲区有数据(其指针不为 0)，则开始按顺序发送缓冲区中的数据（从第 0 个开始），同时 CBUSY 置为 ON，当发送完数据后 CBUSY 置为 OFF。

输入参数：

Val: 发送控制选择。

返回值：

当输入参数为负数时，返回发送缓冲区指针的值

调用形式：

TxCon(Val)

说明：

通常调用参数为 0 的该函数，然后开始向发送缓冲区中写入数据。当数据写完后，再调

用参数为正数的该函数，则开始把这些数据通过通讯口发送出去。判断 CBUSY，则可判断这些数据是否发完。

例：

TxCon(0)

2、发送缓冲区写字符函数 TxChar

函数定义：

FUN I, CharVal As D0

函数功能：

从当前指针开始，把某个字符（该字符的值）写入到发送缓冲区中，执行后发送缓冲区指针增至下个位置。若该字符为英文字符（其高字节为 0），则向缓冲区写入 1 个字节；若该字符为中文字符（其高字节不为 0），则向缓冲区写入 1 个字（高字节在先，低字节在后）。

输入参数：

CharVal: 要写入的字符的值（编码）。

返回值：

无

调用形式：

TxChar(CharVal)

说明：

若该函数没有正确执行（通讯忙，缓冲区溢出等），则溢出继电器 OV 为 ON。但正确执行后并不使 OV 复位。

例：

TxChar(H02) 写入字节数据 H02。

TxChar('A') 写入字符 A 的编码（H41）。

3、发送缓冲区写字函数 TxWord

函数定义：

FUN I, Word As D0

函数功能：

从当前指针开始，把某个字（两个字节）写入到发送缓冲区中（高字节在先，低字节在后），执行后发送缓冲区指针增至下个位置（加 2）。

输入参数:

Word: 要写入的字（整型数）。

返回值:

无

调用形式:

TxWord (Word)

说明:

若该函数没有正确执行（通讯忙，缓冲区溢出等），则溢出继电器 OV 为 ON。但正确执行后并不使 OV 复位。

例:

TxWord(H0ABCD)

4、发送缓冲区写十六进制值函数 TxVal

函数定义:

FUN I, Val As D0

函数功能:

从当前指针开始，把某个十六进制值写入到发送缓冲区中，先把该数值转换为十六进制的 ASCII 码，然后把该 ASCII 码(4 个字符)写入缓冲区中，执行发送缓冲区指针增至下个位置（加 4）。

输入参数:

Val: 要写入的十六进制值（字）。

返回值:

无

调用形式:

TxVal (Val)

说明:

若该函数没有正确执行（通讯忙，缓冲区溢出等），则溢出继电器 OV 为 ON。但正确执行后并不使 OV 复位。

例:

TxVal(H1234) 则依次向缓冲区写入 H31、H32、H33、H34。

5、发送缓冲区写字符串函数 TxStr

函数定义:

FUN I, StrAddr As D0

函数功能:

从当前指针开始,把某个字符串常数写入到发送缓冲区中,若字符串中的字符为英文字符(高字节为 0),则占 1 个字节位置,若字符串中的字符为中文字符(高字节不为 0),则占 2 个字节位置(高字节在先,低字节在后),执行后发送缓冲区指针增至下个位置。

输入参数:

StrAddr: 要写入的字符串的地址。

返回值:

无

调用形式:

TxStr(StrAddr)

说明:

若该函数没有正确执行(通讯忙,缓冲区溢出等),则溢出继电器 OV 为 ON。但正确执行后并不使 OV 复位。

例:

TxStr("123456") 则依次向缓冲区写入 H31、H32、H33、H34、H35、H36。

TxStr("12 你好") 则依次向缓冲区写入 H31、H32、H0C4、H0E3、H0BA、H0C3。

TxStr({H02,H30,H31,H32,H33,H0D,H0A,K0}) 则依次向缓冲区写入 H02、H30、H31、H32、H33、H0D、H0A。注:当输入参数为数据表格式时,用户必须在该数据表的最后加上字符串结束标记 K0,并且该标记不会被发送,在数据表的其他地方不许有数值 0。

6、发送缓冲区求累加和函数 TxSum

函数定义:

FUN I, BuffPos As D0, Len As D1

函数功能:

按用户指定的起始位置和长度,对发送缓冲区中的数据求累加和,把该值作为函数的返回值(可选择为 1 个字节或 1 个字)。

输入参数:

BuffPos: 发送缓冲区中要求求和的数据块的起始位置(0~127)。若位 15 为 0,则累加

结果为 1 个字节（8 位），若位 15 为 1，则累加结果为 1 个字（16 位）。

Len: 该数据块的长度。

返回值:

该数据块的累加和，若 BuffPos 的位 15 为 0，则结果为 8 位；若 BuffPos 的位 15 为 1，则结果为 16 位。

调用形式:

TxSum(BuffPos, Len)

例:

D0 = TxSum(1,8) 对发送缓冲区位置 1 开始的 8 个字节求累加和，结果为 8 位。

D0 = TxSum(H8001,8) 对发送缓冲区位置 1 开始的 8 个字节求累加和，结果为 16 位。

7、发送缓冲区求异或和函数 TxXor

函数定义:

FUN I, BuffPos As D0, Len As D1

函数功能:

按用户指定的起始位置和长度，对发送缓冲区中的数据按字节求异或和，把该值作为函数的返回值(1 个字节)。

输入参数:

BuffPos: 发送缓冲区中要求和的数据块的起始位置（0~127）。

Len: 该数据块的长度。

返回值:

该数据块的异或和（1 个字节）。

调用形式:

TxXor(BuffPos, Len)

例:

D0 = TxXor(1,8)

8、发送缓冲区求 CRC 函数 TxCRC

函数定义:

FUN I, BuffPos As D0, Len As D1

函数功能:

按用户指定的起始位置和长度，对发送缓冲区中的数据求 CRC-16 校验，把该值作为函

数的返回值(1 个字)。

输入参数:

BuffPos: 发送缓冲区中要求 CRC-16 校验的数据块的起始位置 (0~127)。

Len: 该数据块的长度。

返回值:

该数据块的 CRC-16 校验值 (1 个字)。

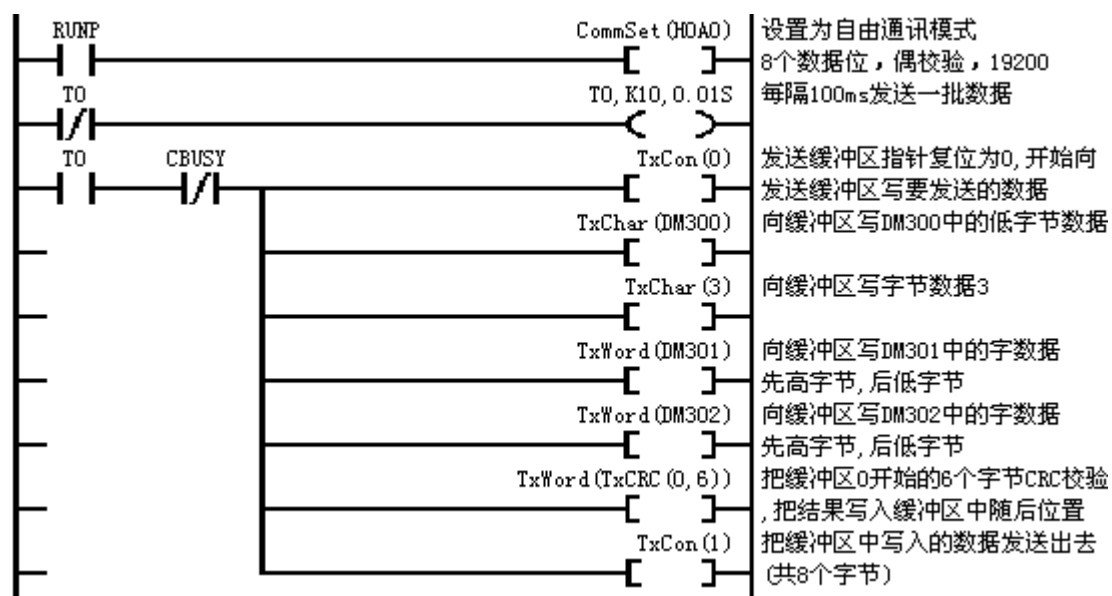
调用形式:

TxCRC(BuffPos, Len)

例:

$$D0 = TxCRC(0,8)$$

自由口通讯发送数据例子:



1.10.11 编程口自由通讯接收函数

该组通讯函数在系统监控通讯模式下被跳过不执行，在完全自由通讯模式下才执行。

系统为自由口通讯设置了 128 个字节的接收缓冲区（与发送缓冲区共用）和 1 个接收缓冲区指针，接收缓冲区的地址编号为 0~127。

判断接收缓冲区指针的值，则可判断是否接收到数据。若接收缓冲区指针小于 0，则表示还没有接收到数据。若接收缓冲区指针大于或等于 0，则表示已经接收到数据，并且该指针的值即为最后收到的字节数据在接收缓冲区中的位置（该值+1 为已接收到的字节数）

函数 RxCon(1)返回接收缓冲区指针的值，RxCon(0)使接收缓冲区指针复位为负数。

1、自由口接收控制函数 RxCon

函数定义：

FUN I, Val As D0

函数功能：

根据输入参数的值，执行不同的操作。

当输入参数为 0，则使接收缓冲区指针复位为负数，表示接收缓冲区空，同时接收字符间隔超时状态复位为非 0 的值；若通讯模式为完全自由通讯模式，则若无起始字符，则允许接收；若有起始字符，则禁止接收，直到收到起始字符后才允许。

当输入参数为正数，则函数返回接收缓冲区指针的值。若该值为负数，则表示接收缓冲区还没有收到数据；否则表示最新收到的数据在接收缓冲区中的位置（接收缓冲区从 0 开始按顺序存放收到的数据）。

当输入参数为负数，则函数返回接收字符间隔超时状态，为 0 表示接收字符间隔超时，即已经接收过数据但在规定的时间（波特率 9.6K 时 10ms，19.2K 和 38.4K 时 5ms，115.2K 时 1ms）内没有接收到新数据。

输入参数：

Val: 接收控制选择。

返回值：

输入参数为正数，返回接收缓冲区指针的值；为负数，返回接收字符间隔超时状态。

调用形式：

RxCon(Val)

例：

RxCon(0)

2、接收缓冲区字节读函数 RxChar

函数定义:

FUN I, BuffPos As D0

函数功能:

读接收缓冲区中指定位置(0~127)的单个字节数据作为函数的返回值。

输入参数:

BuffPos: 接收缓冲区中指定的位置。

返回值:

该位置处的数据（1 个字节）。

调用形式:

RxChar(BuffPos)

例:

DM300 = RxChar(2)

3、接收缓冲区字读函数 RxWord

函数定义:

FUN I, BuffPos As D0

函数功能:

读接收缓冲区中指定位置(0~127)的单个字数据作为函数的返回值。

输入参数:

BuffPos: 接收缓冲区中指定的位置。

返回值:

该位置处的数据（1 个字，先高后低）

调用形式:

RxWord (BuffPos)

例:

DM300 = RxWord(2)

4、接收缓冲区读十六进制值函数 RxVal

函数定义:

FUN I, BuffPos As D0, Len As D1

函数功能:

按用户指定的起始位置和长度, 把接收缓冲区中的某段 ASCII 码数据串(最多 8 个字符, 先高后低)转换为十六进制数值, 把该数值作为函数的返回值(最多为 32 位数)。

输入参数:

BuffPos: 接收缓冲区中的 ASCII 码数据串的起始位置(0~127)。

Len: 该数据串的长度(只能为 2、4、8, 分别对应于结果的字节、字、双字)

返回值:

转换后的十六进制数值。

调用形式:

RxVal (BuffPos, Len)

例:

DM300 = RxVal(0,4)

LDM302 = RxVal(4,8)

5、接收缓冲区读字符串函数 RxStr*函数定义:*

FUN I, DMAAddr As D0, BuffPos As D1, Len As D2

函数功能:

按用户指定的起始位置和长度, 把接收缓冲区中的某段数据串读入到 PLC 的数据存储器 DM 中, 接收缓冲区中的 1 个字节数据占用 PLC 的 1 个字的数据存储器。

输入参数:

DMAAddr: 数据存储器 DM 块的首地址 (256~)。

BuffPos: 接收缓冲区中的数据串的起始位置(0~127)。

Len: 接收缓冲区中的数据串的长度 (字节数)。

返回值:

无

调用形式:

RxStr(DMAAddr, BuffPos, Len)

例:

RxStr(#DM300,0,8)

6、接收缓冲区求累加和函数 RxSum

函数定义:

FUN I, BuffPos As D0, Len As D1

函数功能:

按用户指定的起始位置和长度,对接收缓冲区中的数据求累加和,把该值作为函数的返回值(可选择为 1 个字节或 1 个字)。

输入参数:

BuffPos: 接收缓冲区中要求和的数据块的起始位置 (0~127)。若位 15 为 0,则累加结果为 1 个字节 (8 位),若位 15 为 1,则累加结果为 1 个字 (16 位)。

Len: 该数据块的长度。

返回值:

该数据块的累加和,若 BuffPos 的位 15 为 0,则结果为 8 位;若 BuffPos 的位 15 为 1,则结果为 16 位。

调用形式:

RxSum(BuffPos, Len)

例:

D0 = RxSum(1,8) 对接收缓冲区位置 1 开始的 8 个字节求累加和,结果为 8 位。

D0 = RxSum(H8001,8) 对接收缓冲区位置 1 开始的 8 个字节求累加和,结果为 16 位。

7、接收缓冲区求异或和函数 RxXor

函数定义:

FUN I, BuffPos As D0, Len As D1

函数功能:

按用户指定的起始位置和长度,对接收缓冲区中的数据按字节求异或和,把该值作为函数的返回值(1 个字节)。

输入参数:

BuffPos: 接收缓冲区中要求和的数据块的起始位置 (0~127)。

Len: 该数据块的长度。

返回值:

该数据块的异或和 (1 个字节)。

调用形式:

RxXor(BuffPos, Len)

例:

D0 = RxXor(1,8)

8、接收缓冲区求 CRC 函数 RxCRC

函数定义:

FUN I, BuffPos As D0, Len As D1

函数功能:

按用户指定的起始位置和长度，对接收缓冲区中的数据求 CRC-16 校验，把该值作为函数的返回值(1 个字)。

输入参数:

BuffPos: 接收缓冲区中要求 CRC-16 校验的数据块的起始位置 (0~127)。

Len: 该数据块的长度。

返回值:

该数据块的 CRC-16 校验值 (1 个字)。

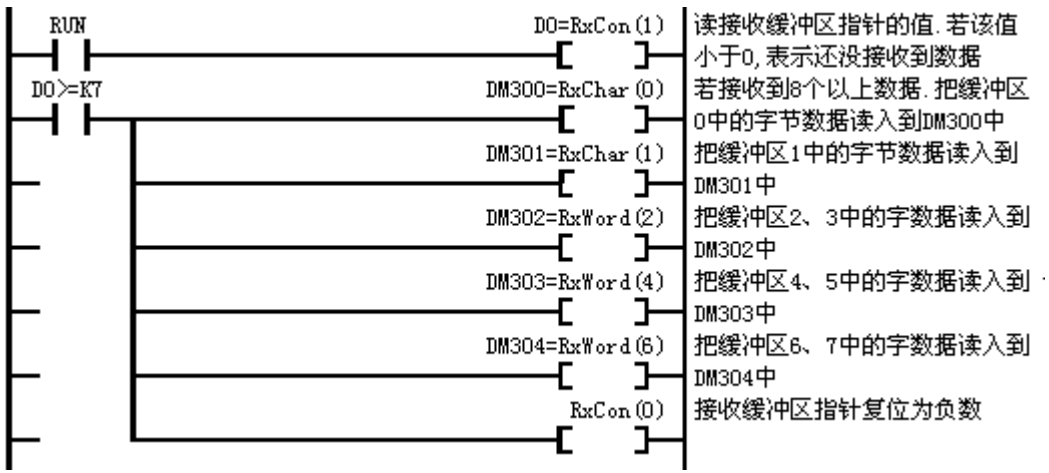
调用形式:

RxCRC(BuffPos, Len)

例:

D0 = RxCRC(0,8)

自由口通讯接收数据例子:



1.10.12 32 位数移位函数 (ROL/ROR/ASL/ASR/LSL/LSR)

1、32 位循环左移函数 ROL

函数定义:

FUN L, Val As LDM256, Num As D0

函数功能:

把输入的 32 位长整数 (双字) Val 自成一个封闭的左移环移位 Num 次 (每次最高位都移入位 0 中) 后作为函数的返回值。

输入参数:

Val: 输入的 32 位长整数 (双字) 数值。

Num: 移位的位数 (1~31)

返回值:

移位后的数值。

调用形式:

ROL(Val, Num)

例:

LDM400=ROL(LDM300,5)

2、32 位循环左移函数 ROR

函数定义:

FUN L, Val As LDM256, Num As D0

函数功能:

把输入的 32 位长整数 (双字) Val 自成一个封闭的右移环移位 Num 次 (每次位 0 都移入最高位中) 后作为函数的返回值。

输入参数:

Val: 输入的 32 位长整数 (双字) 数值。

Num: 移位的位数 (1~31)

返回值:

移位后的数值。

调用形式:

ROR(Val, Num)

例:

LDM400=ROR(LDM300,5)

3、32 位算术左移函数 ASL

函数定义:

FUN L, Val As LDM256, Num As D0

函数功能:

把输入的 32 位长整数（双字）Val 的内容向左移动 Num 位后作为函数的返回值，其左端移出的位被丢弃，并在其右端补以相应位数的“0”。

输入参数:

Val: 输入的 32 位长整数（双字）数值。

Num: 移位的位数（1~31）

返回值:

移位后的数值。

调用形式:

ASL(Val, Num)

例:

LDM400=ASL(LDM300,5)

4、32 位算术右移函数 ASR

函数定义:

FUN L, Val As LDM256, Num As D0

函数功能:

把输入的 32 位长整数（双字）Val 的内容向右移动 Num 位后作为函数的返回值，其右端移出的位被丢弃，若原来数据的最高位为“0”，则在其左端补以相应位数的“0”，若原来数据的最高位为“1”，则在其左端补以相应位数的“1”（即保持原来的符号不变）。

输入参数:

Val: 输入的 32 位长整数（双字）数值。

Num: 移位的位数（1~31）

返回值:

移位后的数值。

调用形式:

ASR(Val, Num)

例:

LDM400=ASR(LDM300,5)

5、32 位逻辑左移函数 LSL

函数定义:

FUN L, Val As LDM256, Num As D0

函数功能:

把输入的 32 位长整数（双字）Val 的内容向左移动 Num 位后作为函数的返回值，其左端移出的位被丢弃，并在其右端补以相应位数的“0”。

输入参数:

Val: 输入的 32 位长整数（双字）数值。

Num: 移位的位数（1~31）

返回值:

移位后的数值。

调用形式:

LSL(Val, Num)

例:

LDM400=LSL(LDM300,5)

6、32 位逻辑右移函数 ASR

函数定义:

FUN L, Val As LDM256, Num As D0

函数功能:

把输入的 32 位长整数（双字）Val 的内容向右移动 Num 位后作为函数的返回值，其右端移出的位被丢弃，并在其左端补以相应位数的“0”。

输入参数:

Val: 输入的 32 位长整数（双字）数值。

Num: 移位的位数（1~31）

返回值:

移位后的数值。

调用形式:

LSR(Val, Num)

例:

LDM400=LSR(LDM300,5)

1.10.13 取模（取余数）函数（IMOD）

函数定义:

FUN L, Dividend As LDM256, Divisor As LDM258

函数功能:

把两个长整数（或整数）的余数作为函数的返回值。

输入参数:

Dividend: 被除数。

Divisor: 除数。

返回值:

Dividend / Divisor 的余数。

调用形式:

IMOD(Dividend, Divisor)

例:

DM400 = IMOD(LDM300,10000)

1.10.14 取整数值的最低字节有符号数值函数（BYTE）

函数定义:

FUN L, Val As LDM256

函数功能:

取函数输入的整数值的最低字节有符号数值（-128~127）作为函数的返回值。

输入参数:

Val: 输入的整数值。

返回值:

输入的整数值的最低字节有符号数值（-128~127）。

调用形式:

BYTE(Val)

例:

DM400 = BYTE(DM300) 如果 DM300 中的值为 0XABCD, 则执行该式子后 DM400 中的值为 0XFFCD (-51)

1.10.15 DM 存储器区按字节读写函数 (RB/WB/RW/WW/RL/WL)

1、字节读函数 RB

函数定义:

FUN I, ByteAddr As LDM256

函数功能:

DM 存储器区按字节地址读字节作为函数的返回值。

输入参数:

ByteAddr: DM 存储器区的字节地址。DM 存储器字的低字节的字节地址为 2*字地址, 字的高字节的字节地址为 2*字地址+1。

返回值:

读出的字节值(0~255)。

调用形式:

RB(ByteAddr)

例:

DM400=RB(2*#DM300) 如果 DM300 中的值为 0XABCD, 则执行该式子后 DM400 中的值为 0XCD (205)

2、字节写函数 WB

函数定义:

FUN I, ByteAddr As LDM256, ByteVal As D0

函数功能:

DM 存储器区按字节地址写字节。

输入参数:

ByteAddr: DM 存储器区的字节地址。DM 存储器字的低字节的字节地址为 2*字地址, 字的高字节的字节地址为 2*字地址+1。

ByteVal: 要写入的数值(取低字节)。

调用形式:

WB(ByteAddr, ByteVal)

例:

WB(2*#DM300,123) 向 DM300 的低字节写入数值 123

3、字读函数 RW

函数定义:

FUN I, ByteAddr As LDM256

函数功能:

DM 存储器区按字节地址读字作为函数的返回值。

输入参数:

ByteAddr: DM 存储器区的字节地址。DM 存储器字的低字节的字节地址为 2*字地址, 字的高字节的字节地址为 2*字地址+1。

返回值:

读出的字值(按小端格式读)。

调用形式:

RW(ByteAddr)

例:

DM400=RW(#DMH500)

4、字写函数 WW

函数定义:

FUN I, ByteAddr As LDM256, Val As D0

函数功能:

DM 存储器区按字节地址写字。

输入参数:

ByteAddr: DM 存储器区的字节地址。DM 存储器字的低字节的字节地址为 2*字地址, 字的高字节的字节地址为 2*字地址+1。

Val: 要写入的数值(按小端格式写)。

调用形式:

WW(ByteAddr, Val)

例:

WW(#DMH500,1234)

5、双字读函数 RL

函数定义:

FUN I, ByteAddr As LDM256

函数功能:

DM 存储器区按字节地址读双字作为函数的返回值。

输入参数:

ByteAddr: DM 存储器区的字节地址。DM 存储器字的低字节的字节地址为 2*字地址, 字的高字节的字节地址为 2*字地址+1。

返回值:

读出的双字值(按小端格式读)。

调用形式:

RL(ByteAddr)

例:

DM400=RL(#DMH500)

6、双字写函数 WL

函数定义:

FUN I, ByteAddr As LDM256, Val As LDM258

函数功能:

DM 存储器区按字节地址写双字。

输入参数:

ByteAddr: DM 存储器区的字节地址。DM 存储器字的低字节的字节地址为 2*字地址, 字的高字节的字节地址为 2*字地址+1。

Val: 要写入的数值(按小端格式写)。

调用形式:

WL(ByteAddr, Val)

例:

WL(#DMH500,12345678)

1.10.16 指数和对数函数（FEXP/FLN/FLG）

1、浮点数 e 为底指数函数 FEXP

函数定义：

FUN F, Val As FDM256

函数功能：

返回某个浮点型数值的 e 为底指数。

输入参数：

Val: 求 e 为底指数的浮点型数值。

返回值：

e 为底指数值。

调用形式：

FEXP(Val)

例：

FDM400=FEXP(FDM300)

2、浮点数 e 为底对数函数 FLN

函数定义：

FUN F, Val As FDM256

函数功能：

返回某个浮点型数值的 e 为底对数。

输入参数：

Val: 求 e 为底对数的浮点型数值。

返回值：

e 为底对数值。

调用形式：

FLN(Val)

例：

FDM400=FLN(FDM300)

3、浮点数 10 为底对数函数 FLG

函数定义：

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的 10 为底对数。

输入参数:

Val: 求 10 为底对数的浮点型数值。

返回值:

10 为底对数值。

调用形式:

FLG(Val)

例:

FDM400=FLG(FDM300)

1.10.17 三角函数 (FCOS/FSIN/FTAN)

1、浮点数余弦函数 FCOS

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的余弦数。

输入参数:

Val: 求余弦的浮点型数值 (弧度)。

返回值:

余弦数。

调用形式:

FCOS(Val)

例:

FDM400=FCOS(FDM300)

2、浮点数正弦函数 FSIN

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的正弦数。

输入参数:

Val: 求正弦的浮点型数值（弧度）。

返回值:

正弦数。

调用形式:

FSIN(Val)

例:

FDM400=FSIN(FDM300)

3、浮点数正切函数 FTAN

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的正切数。

输入参数:

Val: 求正切的浮点型数值（弧度）。

返回值:

正切数。

调用形式:

FTAN(Val)

例:

FDM400=FTAN(FDM300)

1.10.18 反三角函数（FACOS/FASIN/FATAN）

1、浮点数反余弦函数 FACOS

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的反余弦数（弧度）。

输入参数:

Val: 求反余弦的浮点型数值。

返回值:

反余弦数（弧度）。

调用形式:

FACOS(Val)

例:

FDM400=FACOS(FDM300)

2、浮点数反正弦函数 FASIN

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的反正弦数（弧度）。

输入参数:

Val: 求反正弦的浮点型数值。

返回值:

反正弦数（弧度）。

调用形式:

FASIN(Val)

例:

FDM400=FASIN(FDM300)

3、浮点数反正切函数 FATAN

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的反正切数（弧度）。

输入参数:

Val: 求反正切的浮点型数值。

返回值:

反正切数（弧度）。

调用形式:

FATAN(Val)

例:

FDM400=FATAN(FDM300)

1.10.19 双曲函数 (FCOSH/FSINH/FTANH)

1、浮点数双曲余弦函数 FCOSH

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的双曲余弦数。

输入参数:

Val: 求双曲余弦的浮点型数值 (弧度)。

返回值:

双曲余弦数。

调用形式:

FCOSH(Val)

例:

FDM400=FCOSH(FDM300)

2、浮点数双曲正弦函数 FSINH

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的双曲正弦数。

输入参数:

Val: 求双曲正弦的浮点型数值 (弧度)。

返回值:

双曲正弦数。

调用形式:

FSINH(Val)

例:

FDM400=FSINH(FDM300)

3、浮点数双曲正切函数 FTANH

函数定义:

FUN F, Val As FDM256

函数功能:

返回某个浮点型数值的双曲正切数。

输入参数:

Val: 求双曲正切的浮点型数值（弧度）。

返回值:

双曲正切数。

调用形式:

FTANH(Val)

例:

FDM400=FTANH(FDM300)

1.10.20 取字符串变量字符长度函数（STRLEN）

函数定义:

FUN I, StrAddr As LDM256

函数功能:

取字符串变量的字符个数作为函数的返回值。

输入参数:

StrAddr: 字符串变量地址。

返回值:

字符串变量的字符个数。

调用形式:

STRLEN(StrAddr)

例:

DM500=STRLEN(#STR1) 注：字符串变量要加取地址前缀#

1.10.21 字符串变量加载字符串常数函数（STRLOD）

函数定义:

FUN L, StrAddr As LDM256, String As LDM258

函数功能:

字符串变量=字符串常数。

输入参数:

StrAddr: 字符串变量地址。

String: 字符串常数。

返回值:

字符串变量地址。

调用形式:

STRLOD(StrAddr, String)

例:

STRLOD(#STR1, "ABC123") 注: 字符串变量要加取地址前缀#

1.10.22 字符串变量与字符串变量传送函数 (STRMOV)

函数定义:

FUN L, Str1Addr As LDM256, Str2Addr As LDM258

函数功能:

Str1=Str2。

输入参数:

Str1Addr: 字符串变量 1 地址。

Str2Addr: 字符串变量 2 地址。

返回值:

字符串变量 1 地址。

调用形式:

STRMOV(Str1Addr, Str2Addr)

例:

STRMOV(#STR1, #STR2) 注: 字符串变量要加取地址前缀#

1.10.23 字符串变量比较函数 (STRCMPC/STRCMPV)

1、字符串变量与字符串常数比较函数 STRCMPC

函数定义:

FUN L, StrAddr As LDM256, String As LDM258

函数功能:

比较字符串变量与字符串常数是否相同。

输入参数:

StrAddr: 字符串变量地址。

String: 字符串常数。

返回值:

0: 相同, 1: 字符串变量>字符串常数, -1: 字符串变量<字符串常数。

调用形式:

STRCMPC(StrAddr, String)

例:

M0=STRCMPC(#STR1, "ABC123") 注: 字符串变量要加取地址前缀#

2、字符串变量与字符串变量比较函数 STRCMPV

函数定义:

FUN L, Str1Addr As LDM256, Str2Addr As LDM258

函数功能:

比较字符串变量与字符串变量是否相同。

输入参数:

Str1Addr: 字符串变量 1 地址。

Str2Addr: 字符串变量 2 地址。

返回值:

0: 相同, 1: 字符串变量 1>字符串变量 2, -1: 字符串变量 1<字符串变量 2。

调用形式:

STRCMPV(Str1Addr, Str2Addr)

例:

M0=STRCMPV(#STR1, #STR2) 注: 字符串变量要加取地址前缀#

1.10.24 字符串变量相加函数 (STRADD/STRRAD)

1、字符串变量与字符串变量相加函数 STRADD

函数定义:

FUN L, Str1Addr As LDM256, Str2Addr As LDM258

函数功能:

Str1=Str1+Str2, 例如 Str1="ABC"、Str2="123", 则执行后 Str1="ABC123"。

输入参数:

Str1Addr: 字符串变量 1 地址。

Str2Addr: 字符串变量 2 地址。

返回值:

字符串变量 1 地址。

调用形式:

STRADD(Str1Addr, Str2Addr)

例:

STRADD(#STR1,#STR2) 注: 字符串变量要加取地址前缀#

2、字符串变量与字符串变量反向相加函数 STRRAD

函数定义:

FUN L, Str1Addr As LDM256, Str2Addr As LDM258

函数功能:

Str1=Str2+Str1, 例如 Str1="ABC"、Str2="123", 则执行后 Str1="123ABC"。

输入参数:

Str1Addr: 字符串变量 1 地址。

Str2Addr: 字符串变量 2 地址。

返回值:

字符串变量 1 地址。

调用形式:

STRRAD(Str1Addr, Str2Addr)

例:

STRRAD(#STR1, #STR2) 注: 字符串变量要加取地址前缀#

1.10.25 字符串变量与字节字符串转换函数 (STRASCB/ASCBSTR)

1、字符串变量转换为字符串字节格式函数 STRASCB

函数定义:

FUN L, ASCBAddr As LDM256, StrAddr As LDM258

函数功能:

把 GB2312 编码字符串变量转换为字符串字节格式（1 个字存两个 ASCII 码字符）。

输入参数:

ASCBAAddr: 字符串字节格式地址。

StrAddr: 字符串变量地址。

返回值:

字符串字节格式字节数(不包括字符串结束标记 0)。

说明:

转换后的字符串字节格式保证最后 1 个字为 0，即若字符串字节长度为奇数，则除字符串结束标记字节 0 外再多 1 个字 0，而若字符串字节长度为偶数，则字符串最后 1 个字为 0（包含字符串结束标记字节 0）。

调用形式:

STRASCB(ASCBAAddr, StrAddr)

例:

DM500=STRASCB(#ASCBSTR1,#STR1) 注：变量要加取地址前缀#

2、字符串字节格式转换为字符串变量函数 ASCBSTR

函数定义:

FUN L, StrAddr As LDM256, ASCBAAddr As LDM258

函数功能:

把 GB2312 编码字符串字节格式（1 个字存两个 ASCII 码字符）转换为字符串变量。

输入参数:

StrAddr: 字符串变量地址。

ASCBAAddr: 字符串字节格式地址。

返回值:

字符串变量字符个数(不包括字符串结束标记 0)。

调用形式:

ASCBSTR(StrAddr, ASCBAAddr)

例:

ASCBSTR(#STR1,#ASCBSTR1) 注：变量要加取地址前缀#

1.10.26 字符串变量与 UTF8 字符串转换函数 (STRUTF8/UTF8STR)

1、字符串变量转换为 UTF8 字符串函数 STRUTF8

函数定义:

FUN L, UTF8Addr As LDM256, StrAddr As LDM258

函数功能:

把 UniCode 编码字符串变量转换为 UTF8 字符串。

输入参数:

UTF8Addr: UTF8 字符串地址。

StrAddr: 字符串变量地址。

返回值:

UTF8 字符串字节数(不包括字符串结束标记 0)。

说明:

转换后的 UTF8 字符串保证最后 1 个字为 0，即若 UTF8 字符字节长度为奇数，则除字符串结束标记字节 0 外再多 1 个字 0，而若 UTF8 字符字节长度为偶数，则字符串最后 1 个字为 0（包含字符串结束标记字节 0）。

调用形式:

STRUTF8(UTF8Addr, StrAddr)

例:

DM500=STRUTF8(#UTF8STR1,#STR1) 注：变量要加取地址前缀#

2、UTF8 字符串转换为字符串变量函数 UTF8STR

函数定义:

FUN L, StrAddr As LDM256, UTF8Addr As LDM258

函数功能:

把 UTF8 字符串转换为 UniCode 编码字符串变量。

输入参数:

StrAddr: 字符串变量地址。

UTF8Addr: UTF8 字符串地址。

返回值:

字符串变量字符个数(不包括字符串结束标记 0)。

调用形式:

UTF8STR(StrAddr, UTF8Addr)

例:

UTF8STR(#STR1,#UTF8STR1) 注: 变量要加取地址前缀#

1.10.27 整型数值转换为字符串变量函数 (VALSTR)

函数定义:

FUN L, StrAddr As D0, Style As D1, Val As LDM256

函数功能:

把整型 (长整型) 数值转换为字符串变量函数。

输入参数:

StrAddr: 字符串变量地址。

Style: 数值样式设置。低 4 位为数值要占用的英文字符数(包含小数点和"-号), 1 为占用 1 个字符, 2 为占用 2 个字符, ...; 位 8 为 0 居右显示, 为 1 居左显示; 位 7--位 4 为小数位数, 0 为无小数, 1 为 1 位小数, 2 为 2 位小数, ...。

Val: 要转换的整型 (长整型) 数值。

返回值:

字符串变量地址。

调用形式:

VALSTR (StrAddr, Style, Val)

例:

VALSTR(#STR1,0X28,LDM300) 注: 字符串变量要加取地址前缀#

1.10.28 字符串字节格式转换数值函数 (STRTOF/STRTOL)

1、字符串字节格式转换为浮点型数值函数 STRTOF

函数定义:

FUN F, ASCBAddr As LDM256

函数功能:

把字符串字节格式 (1 个字存两个 ASCII 码字符) 转换为浮点型数值。

输入参数:

ASCBAddr: 字符串字节格式地址。

返回值:

转换后的浮点型数值。

调用形式:

STRTOF(ASCBAddr)

例:

FDM500=STRTOF(#ASCBSTR1) 注：字符串字节格式变量要加取地址前缀#

2、字符串字节格式转换为整型数值函数 STRTOL

函数定义:

FUN L, ASCBAddr As LDM256

函数功能:

把字符串字节格式（1 个字存两个 ASCII 码字符）转换为整型数值。

输入参数:

ASCBAddr: 字符串字节格式地址。

返回值:

转换后的整型数值。

调用形式:

STRTOL(ASCBAddr)

例:

LDM500=STRTOL(#ASCBSTR1) 注：字符串字节格式变量要加取地址前缀#

1.10.29 32 位无符号除法函数（UDIV）

函数定义:

FUN L, Dividend As LDM256, Divisor As LDM258

函数功能:

把两个 32 位无符号数相除，把商作为函数的返回值。

输入参数:

Dividend: 32 位无符号被除数。

Divisor: 32 位无符号除数。

返回值:

Dividend / Divisor 的商。

调用形式:

UDIV(Dividend, Divisor)

例:

DM400 = UDIV(LDM300,10000)

1.10.30 32 位无符号取模（取余数）函数（UMOD）

函数定义:

FUN L, Dividend As LDM256, Divisor As LDM258

函数功能:

把两个 32 位无符号数相除，把余数作为函数的返回值。

输入参数:

Dividend: 32 位无符号被除数。

Divisor: 32 位无符号除数。

返回值:

Dividend / Divisor 的余数。

调用形式:

UMOD(Dividend, Divisor)

例:

DM400 = UMOD(LDM300,10000)

第二章 外部函数

外部函数是以函数库文件的形式存在，用户要使用这些函数，则必须要在梯形图中添加函数库连接，连接要使用的函数所在的函数库文件。

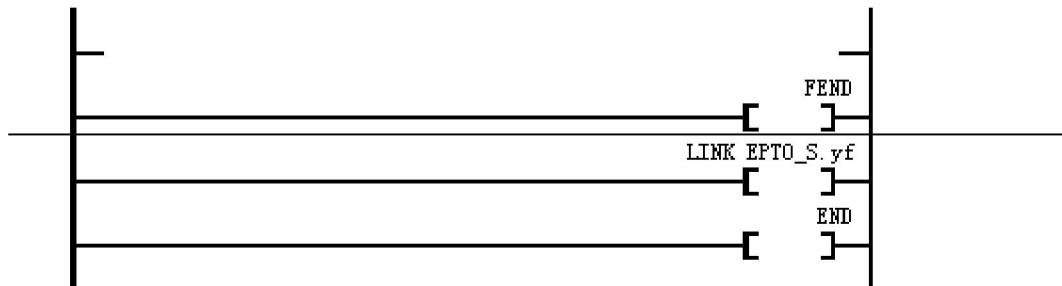
2.1 如何添加函数库连接和创建函数库

若要使用函数库，用户必须在自己的程序中添加函数库连接，连接要使用的函数库文件，使自己的梯形图程序所在的文件夹中有该文件。在编程软件 EasyLad 中的操作如下：

- 点击鼠标右键，弹出以下菜单内容：



- 点击“添加函数库连接”，弹出“打开文件”对话框，找到需要的函数库文件打开即可，同时会自动把该函数库文件复制到梯形图程序所在的文件夹中，若该函数库有所需的全局符号定义文件（扩展名为 `dfn` 的同名文件），则把该文件中的全局符号定义自动导入到用户的全局符号表中（该文件中若有函数库连接则添加到梯形图中）。连接后的程序例子如下：



函数库连接

要创建函数库，方法是：在 EasyLad 编程软件的“工具”菜单中用“转换为语句表”功能，把梯形图程序转换为语句表，在语句表中把主程序部分（FEND 指令上面的，包含 FEND 指令）全删掉，如果有函数库连接指令的话也删掉，即只保留函数库需要的那些函数和那些函数需要的公共静态局部变量定义（函数外静态局部变量定义），把最后面的 END 指令改为 ENDMOD 指令，然后保存为自己起的文件名即可（保存对话框中的保存类型选择“所有文件”），扩展名可以是.yf，也可以不是。如果函数库有需要全局符号定义或系统函数库连接的话，可把这些全局符号定义和系统函数库连接指令放到文件名为函数库文件名、扩展名为.dfn 的文件中，连接该函数库时会自动把这些需要的定义和连接导入到梯形图中。

2.2 数据存储器块操作函数库 (DMBlock.yf)

DM 数据存储器块操作函数库文件为 DMBlock.yf，其中有 BMOV（块传送）、FILL（块填充）、BCMP（块比较）、RAMRD（非易失存储器读）、RAMWR（非易失存储器写）等函数，这些函数与对应的数据存储器块操作指令使用类似，区别是块操作指令的块长度最大为 64 个字，而块操作函数的块长度最大没有限制，当块长度为 64 个字以下时，可使用块操作指令，而当块长度超过 64 个字时，则应使用块操作函数。（注意，非易失存储器读写指令最大为 16 个字）

2.2.1 块传送函数 (BMOV)

函数定义：

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能：

把数据存储器 DM 中的数据块 2 的内容传送到数据块 1 中。从数据块起始处开始传送。

输入参数：

Addr1：数据块 1 的起始地址。

Addr2：数据块 2 的起始地址。

Len：数据块的长度（DM 存储器的个数）。

调用形式：

BMOV(Addr1, Addr2, Len)

说明：

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址，例如 DM300 开始的数据块，在参数中不应是 DM300，而是#DM300。

例如：

BMOV(#DM300,#DM400,100) 表示把从 DM400 开始的 100 个字的内容传送到从 DM300 开始的数据块中。

2.2.2 块右向传送函数 (BRMOV)

函数定义：

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能：

把数据存储器 DM 中的数据块 2 的内容传送到数据块 1 中。从数据块结尾处开始传送。

输入参数:

Addr1: 数据块 1 的起始地址。

Addr2: 数据块 2 的起始地址。

Len: 数据块的长度 (DM 存储器的个数)。

调用形式:

BRMOV(Addr1, Addr2, Len)

说明:

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

BRMOV(#DM300,#DM400,100) 表示把从 DM400 开始的 100 个字的内容传送到从 DM 300 开始的数据块中。

2.2.3 块高低字节交换传送函数 (BXMOV)

函数定义:

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能:

把数据存储器 DM 中的数据块 2 每个字的高低字节交换传送到数据块 1 中。从数据块起始处开始传送。

输入参数:

Addr1: 数据块 1 的起始地址。

Addr2: 数据块 2 的起始地址。

Len: 数据块的长度 (DM 存储器的个数)。

调用形式:

BXMOV(Addr1, Addr2, Len)

说明:

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

BXMOV(#DM300,#DM400,100) 表示把从 DM400 开始的 100 个字的高低字节交换后传送到从 DM300 开始的数据块中。

2.2.4 块加载数据表函数 (BLOD)

函数定义:

FUN I, DM_Addr As D0, TBL_Addr As D1, Len As D2

函数功能:

把数据常数表中的数据加载到 DM 数据块中。

输入参数:

DM_Addr: DM 存储器块的起始地址。

TBL_Addr: 由 DW 指令定义的数据常数表的起始地址。

Len: DM 数据块的长度 (DM 存储器的个数)。

调用形式:

BLOD(DM_Addr, TBL_Addr, Len)

说明:

参数 DM_Addr 要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是 #DM300, 参数 TBL_Addr 要用表格的地址, 即表格名前要加 \$。

例如:

BLOD(#DM300,\$DataTBL,100) 表示从表格 DataTBL 中加载 100 个字到 DM300 开始的数据块中。

2.2.5 块填充函数 (FILL)

函数定义:

FUN I, Addr As D0, Val As D1, Len As D2

函数功能:

把数据存储器 DM 中的某段块用同一个指定数据填充。

输入参数:

Addr: 要填充的 DM 存储器块的起始地址。

Val: 要填充的值。

Len: 数据块的长度 (DM 存储器的个数)。

调用形式:

FILL(Addr, Val, Len)

说明:

参数 Addr 要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

FILL(#DM300,1234,100) 表示把从 DM300 开始的 100 个字都设置为数值 1234。

2.2.6 块比较函数 (BCMP)

函数定义:

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能:

比较数据存储器 DM 中的两个数据块是否相同, 若全部相同, 则特殊继电器 CF(M202) 为 ON, 否则 CF(M202)为 OFF。

输入参数:

Addr1: 数据块 1 的起始地址。

Addr2: 数据块 2 的起始地址。

Len: 数据块的长度 (DM 存储器的个数)。

返回值:

为 1, 表示数据块相同; 为 0, 表示数据块不相同。

调用形式:

BCMP(Addr1, Addr2, Len)

说明:

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

M0 = BCMP(#DM300,#DM400,50) 表示比较 DM300 开始的数据块和 DM400 开始的数据块是否相同, 数据块的长度为 50 个字, 并把比较结果送给 M0, 若相同则 M0 为 ON, 若不相同则 M0 为 OFF。

2.2.7 非易失存储器读函数 (RAMRD)

函数定义:

FUN I, DM_Addr As D0, NOV_Addr As D1, Len As D2

函数功能:

按 NOV_Addr 指定的开始地址从非易失性数据存储器中读入 Len 个字的数据放入由 DM_Addr 所指定的 DM 存储器块中。

输入参数:

DM_Addr: DM 存储器块的起始地址。

NOV_Addr: 非易失性存储器块的起始地址。

Len: 数据块的长度 (DM 存储器的个数)。

调用形式:

RAMRD(DM_Addr, NOV_Addr, Len)

说明:

参数 DM_Addr 要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

RAMRD(#DM300,K10,100) 表示把从非易失性数据存储器地址 10 开始的 100 个字的内容读入到从 DM300 开始的数据块中。

2.2.8 非易失存储器写函数 (RAMWR)

函数定义:

FUN I, DM_Addr As D0, NOV_Addr As D1, Len As D2

函数功能:

把由 DM_Addr 所指定的 DM 存储器块中的内容写入到由 NOV_Addr 所指定的非易失性数据存储器块中。

输入参数:

DM_Addr: DM 存储器块的起始地址。

NOV_Addr: 非易失性存储器块的起始地址。

Len: 数据块的长度 (DM 存储器的个数)。

调用形式:

RAMWR(DM_Addr, NOV_Addr, Len)

说明:

参数 DM_Addr 要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

RAMWR(#DM300,K10,100) 表示把从 DM300 开始的 100 个字的内容写入到非易失性数据存储器地址 10 开始的存储块中。

2.2.9 添加数据到队列函数 (ATT)

函数定义:

FUN I, Val As D3, DM_Addr As D0, Len As D2

函数功能:

把某个数值添加到由 DM_Addr 所指定的数据队列表中, 添加后若队列满则溢出继电器 OV 为 ON, 否则溢出继电器 OV 为 OFF。

输入参数:

Val: 要添加的数值, 整型 (字)。

DM_Addr: 所指定的数据队列表的起始地址, [0]: 移入指针 (指向下一个数据要移入的位置, 0~Len-3); [1]: FIFO 移出指针 (指向下一个要移出数据的位置, 0~Len-3); [2]~...: 队列中的数据。

Len: 数据队列表的长度, 队列中的最大数据个数为 Len-3。

调用形式:

ATT(Val, DM_Addr, Len)

说明:

参数 DM_Addr 要用 DM 存储器的地址, 例如 DM400 开始的队列表数据块, 在参数中不应是 DM400, 而是#DM400。

例如:

ATT(DM280,#DM400,62) 表示把 DM280 的值添加到 DM400 开始的数据队列表中, 队列表的长度为 62 个字, 队列中的最大数据个数为 59 个字。

2.2.10 先进先出移出数据函数（FIFO）

函数定义：

FUN I, DM_Addr As D0, Len As D2

函数功能：

从由 DM_Addr 所指定的数据队列表中按先进先出移出一个数据作为函数的返回值，移出前若队列空则溢出继电器 OV 为 ON，否则溢出继电器 OV 为 OFF。

输入参数：

DM_Addr: 所指定的数据队列表的起始地址，[0]: 移入指针（指向下一个数据要移入的位置，0~Len-3）；[1]: FIFO 移出指针（指向下一个要移出数据的位置，0~Len-3）；[2]~...: 队列中的数据。

Len: 数据队列表的长度，队列中的最大数据个数为 Len-3。

返回值：

移出的数据。

调用形式：

FIFO(DM_Addr, Len)

说明：

参数 DM_Addr 要用 DM 存储器的地址，例如 DM400 开始的队列表数据块，在参数中不应是 DM400，而是#DM400。

例如：

DM300=FIFO(#DM400,62) 表示从 DM400 开始的数据队列表中按先进先出移出一个字数据送给 DM300，队列表的长度为 62 个字，队列中的最大数据个数为 59 个字。

2.2.11 后进先出移出数据函数（LIFO）

函数定义：

FUN I, DM_Addr As D0, Len As D2

函数功能：

从由 DM_Addr 所指定的数据队列表中按后进先出移出一个数据作为函数的返回值，移出前若队列空则溢出继电器 OV 为 ON，否则溢出继电器 OV 为 OFF。

输入参数：

DM_Addr: 所指定的数据队列表的起始地址，[0]: 移入指针（指向下一个数据要移入

的位置, 0~Len-3); [1]: FIFO 移出指针 (指向下一个要移出数据的位置, 0~Len-3); [2]~...: 队列中的数据。

Len: 数据队列表的长度, 队列中的最大数据个数为 Len-3。

返回值:

移出的数据。

调用形式:

LIFO(DM_Addr, Len)

说明:

参数 DM_Addr 要用 DM 存储器的地址, 例如 DM400 开始的队列表数据块, 在参数中不应是 DM400, 而是#DM400。

例如:

DM300=LIFO(#DM400,62) 表示从 DM400 开始的数据队列表中按后进先出移出一个字数据送给 DM300, 队列表的长度为 62 个字, 队列中的最大数据个数为 59 个字。

2.2.12 获得队列表中数据的个数函数 (GTDN)

函数定义:

FUN I, DM_Addr As D0, Len As D2

函数功能:

把由 DM_Addr 所指定的数据队列表中所存的数据 (字) 的个数作为函数的返回值。

输入参数:

DM_Addr: 所指定的数据队列表的起始地址, [0]: 移入指针 (指向下一个数据要移入的位置, 0~Len-3); [1]: FIFO 移出指针 (指向下一个要移出数据的位置, 0~Len-3); [2]~...: 队列中的数据。

Len: 数据队列表的长度, 队列中的最大数据个数为 Len-3。

返回值:

队列表中所存的数据的个数。

调用形式:

GTDN(DM_Addr, Len)

说明:

参数 DM_Addr 要用 DM 存储器的地址, 例如 DM400 开始的队列表数据块, 在参数中

不应是 DM400，而是#DM400。

例如：

DM300=GTDN(#DM400,62) 表示把 DM400 开始的数据队列表中所存的数据的个数送给 DM300，队列表的长度为 62 个字，队列中的最大数据个数为 59 个字。

2.2.13 查找为 ON 线圈位置函数 (FindON)

函数定义：

FUN I, CoilAddr As D0, CoilNum As D1, StartPos As D2

函数功能：

查找一组线圈中为 ON 的线圈在该组所在的位置。

输入参数：

CoilAddr: 该组线圈的起始地址。

CoilNum: 该组线圈的个数。

StartPos: 开始查找位置。从第该值+1 个线圈开始查找，即若该值为 0，则从第 1 个线圈开始查找。若该值>=线圈的个数 CoilNum，则从第 1 个线圈开始查找。每次都要查找该组所有线圈，直到碰到为 ON 的线圈为止。

返回值：

查找到的为 ON 的线圈在该组所在的位置（从 1 开始），若为 0，则表示该组线圈中没有为 ON 的线圈。

调用形式：

FindON(CoilAddr, CoilNum, StartPos)

说明：

若从开始查找位置处到结尾没有查找到为 ON 的线圈，则继续从第 1 个线圈开始查找，直到碰到为 ON 的线圈为止，若查遍所有线圈都没有为 ON 的线圈，则函数返回 0。

例如：

DM300=FindON(#M10, 32, DM300) 表示查找从 M10 开始的 32 个线圈中为 ON 的线圈所在的位置，从第 DM300 的值+1 个线圈开始查找，并把查找的结果送给 DM300。

2.2.14 DM 存储器块高低字节交换函数 (SWAP)

函数定义:

FUN I, DM_Addr As D0, Len As D2

函数功能:

把由 DM_Addr 所指定的 DM 存储器块中的每个字的高低字节进行交换。

输入参数:

DM_Addr: 所指定的 DM 存储器块的起始地址。

Len: DM 存储器块的长度。

调用形式:

SWAP(DM_Addr, Len)

说明:

参数 DM_Addr 要用 DM 存储器的地址，例如 DM400 开始的数据块，在参数中不应是 DM400，而是#DM400。

例如:

SWAP(#DM400,32) 表示把从 DM400 开始共 32 个字的数据块中的每个字的高低字节进行交换。如果 DM400 的值为 H1234，则执行函数后 DM400 的值为 H3412。

2.2.15 DM 存储器块字节分离函数 (SPLIT)

函数定义:

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能:

把数据存储器 DM 中的数据块 2 的每个字分离为 2 个字节后传送到数据块 1 中。数据块 1 中的 1 个字存储分离后的 1 个字节（低地址存分离后的高字节，高地址存分离后的低字节）。

输入参数:

Addr1: 数据块 1 的起始地址。

Addr2: 数据块 2 的起始地址。

Len: 数据块 2 的长度（DM 存储器的个数）。

调用形式:

SPLIT(Addr1, Addr2, Len)

说明:

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

SPLIT(#DM300,#DM400,20) 表示把从 DM400 开始的 20 个字的高字节和低字节分离为 40 个字传送到从 DM300 开始的 40 个字的数据块中。如果 DM400 的值为 H1234, 则执行函数后 DM300 的值为 H0012, DM301 的值为 H0034。

2.2.16 DM 存储器块字节组合函数 (UNITE)

函数定义:

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能:

把数据存储器 DM 中的数据块 2 的每 2 个字的低字节组合为 1 个字后传送到数据块 1 中。数据块 2 中的低地址的低字节在组合字的高字节, 高地址的低字节在组合字的低字节。若数据块 2 的长度 Len 为奇数, 则数据块 2 的最后 1 个字的低字节在数据块 1 的最后 1 个字的低字节中。

输入参数:

Addr1: 数据块 1 的起始地址。

Addr2: 数据块 2 的起始地址。

Len: 数据块 2 的长度 (DM 存储器的个数)。

调用形式:

UNITE (Addr1, Addr2, Len)

说明:

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

UNITE (#DM300,#DM400,20) 表示把从 DM400 开始的 20 个字的低字节组合为 10 个字传送到从 DM300 开始的 10 个字的数据块中。如果 DM400 的值为 H0012, DM401 的值为 H0034, 则执行函数后 DM300 的值为 H1234。

2.2.17 DM 存储器块字转字节函数 (WTOB)

函数定义:

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能:

把数据存储器 DM 中的数据块 2 的每个字分离为 2 个字节后传送到数据块 1 中。数据块 1 中的 1 个字存储分离后的 1 个字节 (低地址存分离后的低字节, 高地址存分离后的高字节)。

输入参数:

Addr1: 数据块 1 的起始地址。

Addr2: 数据块 2 的起始地址。

Len: 数据块 2 的长度 (DM 存储器的个数)。

调用形式:

WTOB(Addr1, Addr2, Len)

说明:

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是 #DM300。

例如:

WTOB(#DM300,#DM400,20) 表示把从 DM400 开始的 20 个字的高字节和低字节分离为 40 个字传送到从 DM300 开始的 40 个字的数据块中。如果 DM400 的值为 H1234, 则执行函数后 DM300 的值为 H0034, DM301 的值为 H0012。

2.2.18 DM 存储器块字节转字函数 (BTOW)

函数定义:

FUN I, Addr1 As D0, Addr2 As D1, Len As D2

函数功能:

把数据存储器 DM 中的数据块 2 的每 2 个字的低字节组合为 1 个字后传送到数据块 1 中。数据块 2 中的低地址的低字节在组合字的低字节, 高地址的低字节在组合字的高字节。若数据块 2 的长度 Len 为奇数, 则数据块 2 的最后 1 个字的低字节在数据块 1 的最后 1 个字的低字节中。

输入参数:

Addr1: 数据块 1 的起始地址。

Addr2: 数据块 2 的起始地址。

Len: 数据块 2 的长度 (DM 存储器的个数)。

调用形式:

BTOW (Addr1, Addr2, Len)

说明:

参数 Addr1 和 Addr2 都是要用 DM 存储器的地址, 例如 DM300 开始的数据块, 在参数中不应是 DM300, 而是#DM300。

例如:

BTOW (#DM300,#DM400,20) 表示把从 DM400 开始的 20 个字的低字节组合为 10 个字传送到从 DM300 开始的 10 个字的数据块中。如果 DM400 的值为 H0012, DM401 的值为 H0034, 则执行函数后 DM300 的值为 H3412。

2.2.19 时间测量函数 (MTS/MTE)

1、测量时间开始 (时间测量起点) 函数 MTS

函数定义:

FUN I, DM_Addr As D0

函数功能:

该函数接通则开始测量时间。测量的时间为 MTS 接通到 MTE 接通之间的时间 (单位: ms)。

输入参数:

DM_Addr: 存储测量时间结果数据块首地址, 占 4 个字, [0]为当前测量时间, [1]为最大测量时间, [2]为最小测量时间, [3]为内部使用。

注:

把最大测量时间和最小测量时间设置为-1 则重新统计最大值和最小值。

调用形式:

MTS(DM_Addr)

例:

MTS(#DM500)

2、测量时间结束（时间测量终点）函数 MTE

函数定义：

FUN I, DM_Addr As D0

函数功能：

该函数接通则结束测量时间。测量的时间为 MTS 接通到 MTE 接通之间的时间（单位：ms）。

输入参数：

DM_Addr: 存储测量时间结果数据块首地址，占 4 个字，[0]为当前测量时间，[1]为最大测量时间，[2]为最小测量时间，[3]为内部使用。

注：

把最大测量时间和最小测量时间设置为-1 则重新统计最大值和最小值。

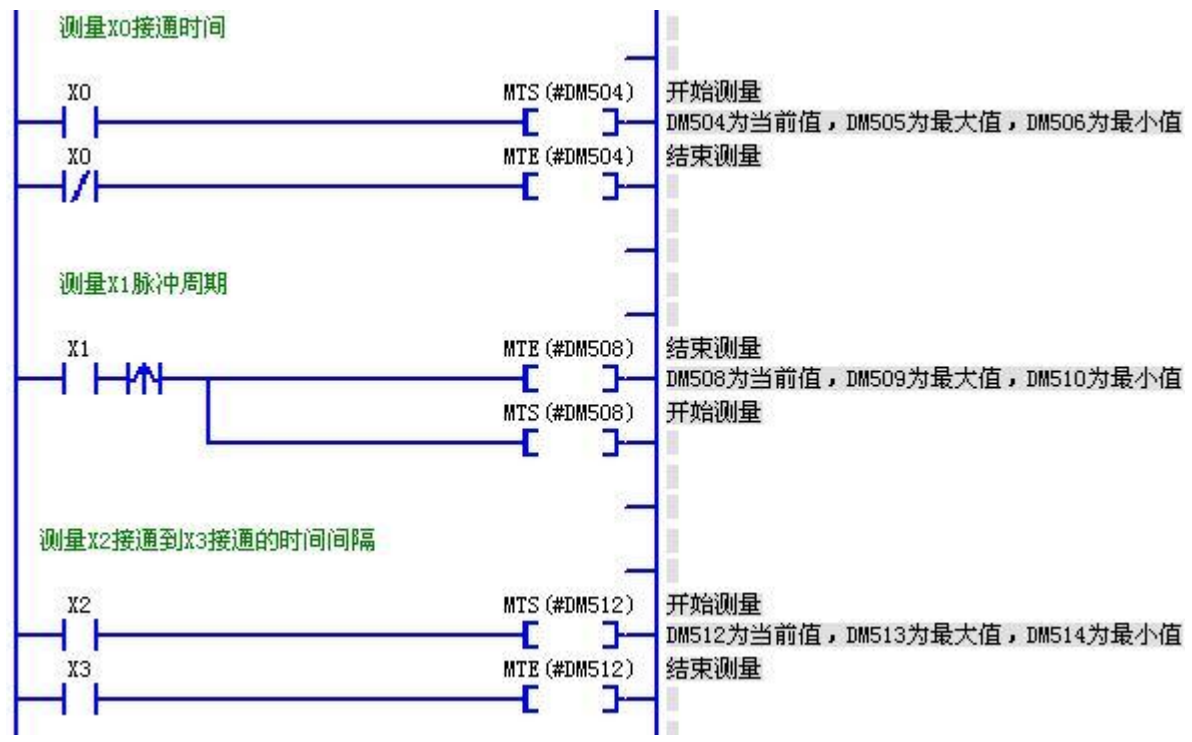
调用形式：

MTE(DM_Addr)

例：

MTE(#DM500)

时间测量梯形图例子：



2.2.20 按字节地址块复制函数 (MEMCPY)

函数定义:

FUN I, ByteAddr1 As LDM256, ByteAddr2 As LDM258, Len As D0

函数功能:

按字节地址把数据存储器 DM 中数据块 2 的内容传送到数据块 1 中。从数据块起始处开始传送。

输入参数:

ByteAddr1: 数据块 1 的起始字节地址。

ByteAddr2: 数据块 2 的起始字节地址。

Len: 数据块的字节长度。

调用形式:

MEMCPY(ByteAddr1, ByteAddr2, Len)

说明:

参数 ByteAddr1 和 ByteAddr2 都是要用 DM 存储器的字节地址, 例如 DM300 低字节开始的数据块, 在参数中不应是 #DM300, 而是 #DML300。

例如:

MEMCPY(#DML500, #DMH800, 50) 表示把从 DM800 高字节开始的 50 个字节的内容传送到从 DM500 低字节开始的数据块中。

2.2.21 数据块求 CRC16 校验函数 (CRC16)

函数定义:

FUN I, ByteAddr As LDM256, Len As D0

函数功能:

按字节地址对 DM 存储器数据块求 CRC16 校验, 把校验值作为函数的返回值。

输入参数:

ByteAddr: 数据块的起始字节地址。

Len: 数据块的字节长度。

返回值:

CRC16 校验值。

调用形式:

CRC16(ByteAddr, Len)

说明:

参数 ByteAddr 要用 DM 存储器的字节地址, 例如 DM300 低字节开始的数据块, 在参数中不应是#DM300, 而是#DML300。

例如:

DM300=CRC16(#DMH500,50) 表示对从 DM500 高字节开始的 50 个字节的内容求 CRC 16 校验并把校验值传送给 DM300。

2.2.22 字符串比较触点函数 (strEQ/strNE)

1、字符串相等比较触点函数 strEQ

函数定义:

FUN E, Str1Addr As LDM256, Str2Addr As LDM258

函数功能:

在触点中比较字符串变量与字符串常数或字符串变量是否相等。

输入参数:

Str1Addr: 字符串变量 1 地址。

Str2Addr: 字符串变量 2 或字符串常数地址。

返回值:

相等返回 ON (通), 不相等返回 OFF (断)。

注:

该函数要使用触点调用。用常开触点调用时若相等则触点接通、不相等则触点断开。

调用形式:

strEQ(Str1Addr, Str2Addr)

例:

strEQ(#STR1,"ABC123");字符串变量与字符串常数比较

strEQ(#STR1,#STR2);字符串变量与字符串变量比较

2、字符串不相等比较触点函数 strNE

函数定义:

FUN E, Str1Addr As LDM256, Str2Addr As LDM258

函数功能:

在触点中比较字符串变量与字符串常数或字符串变量是否不相等。

输入参数:

Str1Addr: 字符串变量 1 地址。

Str2Addr: 字符串变量 2 或字符串常数地址。

返回值:

不相等返回 ON (通), 相等返回 OFF (断)。

注:

该函数要使用触点调用。用常开触点调用时若不相等则触点接通、相等则触点断开。

调用形式:

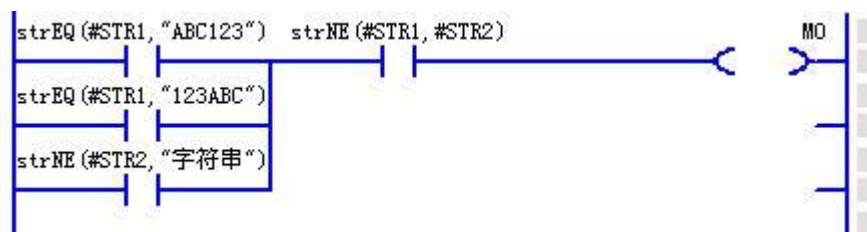
strNE (Str1Addr, Str2Addr)

例:

strNE(#STR1,"ABC123");字符串变量与字符串常数比较

strNE(#STR1,#STR2);字符串变量与字符串变量比较

字符串比较触点函数梯形图例子:

**2.2.23 实时时钟比较函数 (RTCCMP)****函数定义:**

FUN I, RTC_Addr As D0, SET_Addr As D1

函数功能:

当前时间 (年月日时分) 与设置时间比较, 若在设置时间以后 (含设置时间) 则返回 1, 否则返回 0。

输入参数:

RTC_Addr: 当前时间数据块的起始地址, 占 5 个字, 依次为年、月、日、时、分。

SET_Addr: 设置时间数据块的起始地址, 占 5 个字, 依次为年、月、日、时、分。

返回值:

当前时间在设置时间以后（含设置时间）返回 1，否则返回 0。

调用形式:

RTCCMP(RTC_Addr, SET_Addr)

说明:

参数 RTC_Addr 和 SET_Addr 要用 DM 存储器的地址，例如 DM300 开始的数据块，在参数中不应是 DM300，而是#DM300。

例如:

M0=RTCCMP(#DM400,#DM410)

第三章 配方和记录

3.1 配方设置

使用配方功能，可以在易失性存储器和非易失性存储器（或外部闪存）之间实现加载、存储、上组、下组等功能操作，可用来完成单组参数设置、多组同结构配方、数据记录和记录显示等功能。

3.1.1 配方设置说明

点击“配置”菜单，找到“配方设置”菜单项点击即可进入配方设置，其界面如下：



总共有 16 个配方，通过其左边的选择框来选择都使用那些配方，点击对应的“设置”按钮，则进入该配方的具体参数设置界面，如下：

配方设置

说明：当配方保存区为内部非易失性或易失性DM存储器时，必须要在梯形图中连接数据块操作函数库DMBlock.yf。当配方保存区为外部闪存时，必须要连接大容量闪存操作函数库FROM.yf

配方4

加载线圈：		把当前组的数据从保存区加载到工作DM区
保存线圈：		把当前组的数据从工作DM区保存到保存区
下组线圈：		保存当前组数据，然后把下一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
上组线圈：		保存当前组数据，然后把上一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
最大组号：		组索引指针的最大值
每组字数：		每组占用存储器字数(当保存在内部非易失性时必须为2、4、8或16的倍数)
组指针：		当前组索引指针，从0开始。可以是常数、变量 <input type="checkbox"/> 超出最大组号时变为0
当前组DM地址：		当前组数据工作的DM存储器首地址，可以是常数、变量或表达式
配方保存地址：		配方数据保存的存储器首地址(当为内部非易失性时必须为16的倍数)，可以是常数、变量或表达式
配方保存区选择：	<input checked="" type="radio"/> 内部非易失性 <input type="radio"/> 外部闪存 <input type="radio"/> 易失性DM存储器	

确定

在每个配方中，有一组或很多组数据，每组数据的结构、占用存储器字数都相同，同时用户要在易失性 DM 存储器中为该配方设置一块当前组工作 DM 区，该 DM 区的数据结构、占用存储器字数也都与配方中的组数据结构和占用存储器字数相同。用户可选择使用那组数据（由组指针决定），当使用某组数据时，要把该组数据加载到工作 DM 区，需要保存变化时，要把工作 DM 区保存到该组配方中。

■ 加载线圈

把当前组（由组指针决定）的数据从保存区加载到工作 DM 区。每当把该线圈设置为 ON 时就执行该功能，执行完后该线圈自动复位（该线圈若是 RUNP，则不会在此被复位）。

该线圈可为 M 继电器或 DMx.y 位。可以是元件名、符号名、符号名[常数]变址等格式，例如：M10、DM300.9、FLAG1（要在全局符号表中定义过）、FLAG1[3]等。

若该线圈为空，则表示没有该功能。若不需要该项功能，则可把该线圈设置为空。

■ 保存线圈

把当前组（由组指针决定）的数据从工作 DM 区保存到保存区。每当把该线圈设置为 ON 时就执行该功能，执行完后该线圈自动复位。

该线圈可为 M 继电器或 DMx.y 位。可以是元件名、符号名、符号名[常数]变址等格式，

例如：M10、DM300.9、FLAG1（要在全局符号表中定义过）、FLAG1[3]等。

若该线圈为空，则表示没有该功能。若不需要该项功能，则可把该线圈设置为空。

■ 下组线圈

保存当前组数据，然后把组指针加 1，把下一组数据加载到工作 DM 区，但若“不保存”选项有效，则不保存当前组数据，若“不加载”选项有效，则不加载下组数据。每当把该线圈设置为 ON 时就执行该功能，执行完该功能后该线圈自动复位。

该线圈可为 M 继电器或 DMx.y 位。可以是元件名、符号名、符号名[常数]变址等格式，例如：M10、DM300.9、FLAG1（要在全局符号表中定义过）、FLAG1[3]等。

若该线圈为空，则表示没有该功能。若不需要该项功能，则可把该线圈设置为空。

若配方为单组数据，则该线圈应为空。

■ 上组线圈

保存当前组数据，然后把组指针减 1，把上一组数据加载到工作 DM 区，但若“不保存”选项有效，则不保存当前组数据，若“不加载”选项有效，则不加载上组数据。每当把该线圈设置为 ON 时就执行该功能，执行完该功能后该线圈自动复位。

该线圈可为 M 继电器或 DMx.y 位。可以是元件名、符号名、符号名[常数]变址等格式，例如：M10、DM300.9、FLAG1（要在全局符号表中定义过）、FLAG1[3]等。

若该线圈为空，则表示没有该功能。若不需要该项功能，则可把该线圈设置为空。

若配方为单组数据，则该线圈应为空。

■ 最大组号

表示配方中有多少组数据，为这些组的最大组号（组号从 0 开始），也为组索引指针的最大值。

该参数可为常数或字变量。可以是常数或字元件名、符号名、符号名[常数]变址等格式，例如：10、K10、DM300、Var1（要在全局符号表中定义过）、Var1[3]等。

该参数不能为空。

若配方为单组数据，则该参数应为 0。

■ 每组字数

表示配方中的每组数据占用的存储器字数。

该参数可为常数或字变量。可以是常数或字元件名、符号名、符号名[常数]变址等格式，例如：10、K10、DM300、Var1（要在全局符号表中定义过）、Var1[3]等。

该参数不能为空。

■ 组指针

当前组索引指针（从 0 开始），表示哪一组为当前组。

该参数为字变量。可以是字元件名、符号名、符号名[常数]变址等格式，例如：DM300、Var1（要在全局符号表中定义过）、Var1[3]等。但也可为常数 0，此时表示为单组配方。

该参数不能为空。

若“超出最大组号时变为 0”选项有效，则执行“下组线圈”功能时如果组指针超出最大组号则变为 0、执行“上组线圈”功能时如果组指针小于 0 则变为最大组号。

■ 当前组 DM 地址

当前组数据的工作 DM 区存储器首地址。

该参数可以是常数、变量或表达式，例如：常数 400（或#DM400）表示工作 DM 区在从 DM400 开始的存储器块中，DM300 表示工作 DM 区存储器首地址放在 DM300 中。

该参数不能为空。

■ 配方保存地址

配方数据保存的存储器首地址。

该参数可以是常数、变量或表达式，例如：常数 16640 表示配方保存在 PLC 内部非易失性存储器从 16640 开始的存储器块中（配方保存区选择为内部非易失性），DM300 则表示配方保存的存储器首地址放在 DM300 中。

该参数不能为空。

■ 配方保存区选择

配方保存区可选择为 PLC 内部非易失性存储器、PLC 外部扩展闪存或 PLC 内部易失性 DM 存储器。当选择 PLC 内部非易失性存储器或内部易失性 DM 存储器时，要在梯形图中连接数据块操作函数库 DMBLOCK.yf。当选择外部闪存时，要在梯形图中连接大容量闪存操作函数库 FROM.yf。

■ 配方的画面设计

设计配方的画面，可用数据显示或数据设定等元件来显示或设置工作 DM 区和组指针变量，用位控元件来控制（用设定线圈为 ON 功能）加载线圈、保存线圈、下组线圈、上组线圈。

3.1.2 单组参数设置例子

用单组配方可用于完成诸如系统参数设置等只有一种参数的数据类型,在该类参数设置中,要把配方设置中的参数做如下设置:

【加载线圈】设置为 RUNP,表示上电就加载参数到工作 DM 区。

【保存线圈】设置为某个继电器,在画面中可用位控元件控制该继电器。

【下组线圈】设置为空。

【上组线圈】设置为空。

【最大组号】设置为 0。

【每组字数】要根据参数的多少设置。

【组指针】设置为 0。

【当前组 DM 地址】根据实际情况设置,注意 DM 存储器或符号名前面要加“#”号。

【配方保存地址】根据实际情况设置。注意内部非易失性时不要与掉电保持配置所用到的内部非易失性存储器冲突。

【配方保存区选择】应选内部非易失性(若参数需要掉电保持时)。

具体设置例子如下:

配方设置
✕

说明:当配方保存区为内部非易失性或易失性DM存储器时,必须要在梯形图中连接数据块操作函数库DMBlock.yf。当配方保存区为外部闪存时,必须要连接大容量闪存操作函数库FROM.yf

配方1

加载线圈: <input type="text" value="RUNP"/>	把当前组的数据从保存区加载到工作DM区
保存线圈: <input type="text" value="M14"/>	把当前组的数据从工作DM区保存到保存区
下组线圈: <input type="text"/>	保存当前组数据,然后把下一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
上组线圈: <input type="text"/>	保存当前组数据,然后把上一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
最大组号: <input type="text" value="0"/>	组索引指针的最大值
每组字数: <input type="text" value="32"/>	每组占用存储器字数(当保存在内部非易失性时必须为2、4、8或16的倍数)
组指针: <input type="text" value="0"/>	当前组索引指针,从0开始。可以是常数,变量 <input type="checkbox"/> 超出最大组号时变为0
当前组DM地址: <input type="text" value="#DM350"/>	当前组数据工作的DM存储器首地址,可以是常数,变量或表达式
配方保存地址: <input type="text" value="K128"/>	配方数据保存的存储器首地址(当为内部非易失性时必须为16的倍数),可以是常数,变量或表达式
配方保存区选择: <input checked="" type="radio"/> 内部非易失性 <input type="radio"/> 外部闪存 <input type="radio"/> 易失性DM存储器	

确定

3.1.3 多组配方设置例子

用多组配方可用于完成诸如工艺配方设置等多种参数的数据类型，在该类数据类型中，要把配方设置中的参数做如下设置：

【加载线圈】设置为某个继电器，在画面中可用位控元件控制该继电器。

【保存线圈】设置为某个继电器，在画面中可用位控元件控制该继电器。

【下组线圈】设置为某个继电器，在画面中可用位控元件控制该继电器。

【上组线圈】设置为某个继电器，在画面中可用位控元件控制该继电器。

【最大组号】根据实际情况设置。

【每组字数】要根据每组数据的多少设置。

【组指针】设置为某个 DM 存储器。

【当前组 DM 地址】根据实际情况设置。注意 DM 存储器或符号名前面要加“#”号。

【配方保存地址】根据实际情况设置。注意内部非易失性时不要与掉电保持配置所用到的内部非易失性存储器冲突，外部闪存时要为闪存的扇区号（每个扇区为 4096 个字节）。

【配方保存区选择】根据实际情况设置。

具体设置例子如下：

配方设置
✕

说明：当配方保存区为内部非易失性或易失性DM存储器时，必须要在梯形图中连接数据块操作函数库DMBlock.yf。当配方保存区为外部闪存时，必须要连接大容易闪存操作函数库FROM.yf

配方2

加载线圈: M15	把当前组的数据从保存区加载到工作DM区
保存线圈: M16	把当前组的数据从工作DM区保存到保存区
下组线圈: M17	保存当前组数据，然后把下一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
上组线圈: M18	保存当前组数据，然后把上一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
最大组号: 99	组索引指针的最大值
每组字数: 32	每组占用存储器字数 (当保存在内部非易失性时必须为2、4、8或16的倍数)
组指针: DM1000	当前组索引指针，从0开始。可以是常数, 变量 <input type="checkbox"/> 超出最大组号时变为0
当前组DM地址: #DM1001	当前组数据工作的DM存储器首地址, 可以是常数, 变量或表达式
配方保存地址: K17360	配方数据保存的存储器首地址 (当为内部非易失性时必须为16的倍数), 可以是常数, 变量或表达式
配方保存区选择: <input checked="" type="radio"/> 内部非易失性 <input type="radio"/> 外部闪存 <input type="radio"/> 易失性DM存储器	

确定

3.1.4 数据记录保存例子

用多组配方可用于完成诸如数据记录等有多种数据要保存的数据类型，在该类数据类型中，要把配方设置中的参数做如下设置：

【加载线圈】和【保存线圈】都设置为空。

【下组线圈】设置为某个继电器，“不加载”选项有效。当需要保存当前条数据记录时则可在梯形图中把该继电器置位。

【上组线圈】设置为空。

【最大组号】根据实际情况设置。该值+1 为可保存数据记录总条数。

【每组字数】为每条记录的字数。注意内部非易失性时要为 2、4、8 或 16 的倍数，外部闪存时该数值要能被 2048 整除。

【组指针】设置为某个 DM 存储器，该 DM 存储器要掉电保持，“超出最大组号时变为 0”选项有效。

【当前组 DM 地址】该数据块为要保存的当前条数据记录。注意变量前面要加“#”号。

【配方保存地址】为保存数据记录的首地址。注意外部闪存时要为 4096 的倍数。

【配方保存区选择】根据实际情况设置。

具体设置例子如下：

配方设置

说明：当配方保存区为内部非易失性或易失性DM存储器时，必须要在梯形图中连接数据块操作函数库DMBlock.yf。当配方保存区为外部闪存时，必须要连接大容易闪存操作函数库FROM.yf

配方4

加载线圈：		把当前组的数据从保存区加载到工作DM区
保存线圈：		把当前组的数据从工作DM区保存到保存区
下组线圈：	M13	保存当前组数据，然后把下一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input checked="" type="checkbox"/> 不加载
上组线圈：		保存当前组数据，然后把上一组数据加载到工作DM区 <input type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
最大组号：	79	组索引指针的最大值
每组字数：	8	每组占用存储器字数(当保存在内部非易失性时必须为2、4、8或16的倍数)
组指针：	AlmDataI	当前组索引指针，从0开始。可以是常数, 变量 <input checked="" type="checkbox"/> 超出最大组号时变为0
当前组DM地址：	#RTC	当前组数据工作的DM存储器首地址, 可以是常数, 变量或表达式
配方保存地址：	AlmData	配方数据保存的存储器首地址(当为内部非易失性时必须为16的倍数), 可以是常数, 变量或表达式
配方保存区选择：	<input checked="" type="radio"/> 内部非易失性 <input type="radio"/> 外部闪存 <input type="radio"/> 易失性DM存储器	

确定

上图中符号 AlmDataI 定义为 DM860（要掉电保持），符号 RTC 定义为 DM500（实时

时钟数据), 符号 AlmData 定义为 K16656 (非易失性存储区地址), 都在全局符号表中定义。

在梯形图中用于保存数据记录的程序例子如下:



3.1.5 数据记录显示例子

用多组配方可用于完成诸如数据记录显示等多种数据要显示的数据类型,在该类数据类型中,要把配方设置中的参数做如下设置:

【加载线圈】设置为某个继电器,在画面中可置位该继电器实现加载显示数据。

【保存线圈】设置为空。

【下组线圈】设置为某个继电器,“不保存”选项有效,在画面中可置位该继电器实现向下翻页。

【上组线圈】设置为某个继电器,“不保存”选项有效,在画面中可置位该继电器实现向上翻页。

【最大组号】该值+1 为要显示的总页数。应为记录总条数÷每页显示记录的条数-1。

【每组字数】应设置为每页显示记录的条数×每条记录的字数。

【组指针】设置为某个 DM 存储器,为当前要显示的页数(从 0 开始)。

【当前组 DM 地址】当前页显示的数据,注意 DM 存储器或符号名前面要加“#”号。

【配方保存地址】保存数据记录的首地址,要与该数据记录保存中的配方保存地址相同。

【配方保存区选择】根据实际情况设置。

具体设置例子如下:

配方设置

说明:当配方保存区为内部非易失性或易失性DM存储器时,必须要在梯形图中连接数据块操作函数库DMBlock.yf。当配方保存区为外部闪存时,必须要连接大容易闪存操作函数库FROM.yf

配方3

加载线圈: M12	把当前组的数据从保存区加载到工作DM区
保存线圈:	把当前组的数据从工作DM区保存到保存区
下组线圈: M11	保存当前组数据,然后把下一组数据加载到工作DM区 <input checked="" type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
上组线圈: M10	保存当前组数据,然后把上一组数据加载到工作DM区 <input checked="" type="checkbox"/> 不保存 <input type="checkbox"/> 不加载
最大组号: 9	组索引指针的最大值
每组字数: 64	每组占用存储器字数(当保存在内部非易失性时必须为2、4、8或16的倍数)
组指针: AlmShowP	当前组索引指针,从0开始。可以是常数,变量 <input type="checkbox"/> 超出最大组号时变为0
当前组DM地址: #AlmShow	当前组数据工作的DM存储器首地址,可以是常数,变量或表达式
配方保存地址: AlmData	配方数据保存的存储器首地址(当为内部非易失性时必须为16的倍数),可以是常数,变量或表达式
配方保存区选择: <input checked="" type="radio"/> 内部非易失性 <input type="radio"/> 外部闪存 <input type="radio"/> 易失性DM存储器	

确定

上图中若每条记录占 8 个字,每页要显示 8 条记录,则每组字数设为 64。

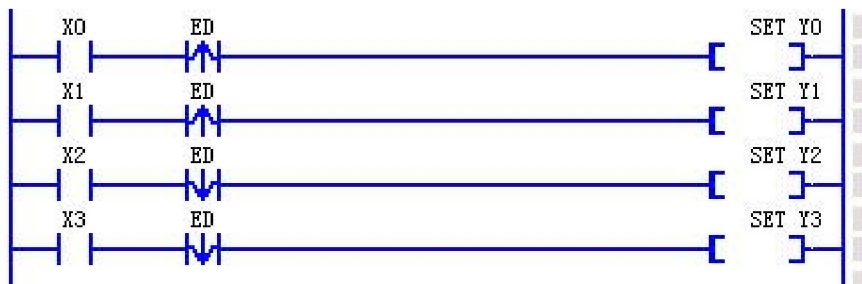
附录 A 边缘检测触点使用个数溢出处理

无操作数边缘检测触点（包括 MDO 指令）最多可使用 512 个，当使用个数超出后，可使用 DMx.y 来保存上个扫描周期该边缘检测触点前的逻辑运算结果，具体方法就是使用操作数为 DM 字元件的边缘检测触点，推荐使用数组符号名来定义该 DM 字元件数据块，具体例子如下：

使用全局符号表定义边缘检测触点用 DM 数据块符号名 ED：

	符号名	类型或地址	描述
1	ED[8]	DM500	定义边缘检测触点用DM数据块，8个字可使用128个边缘检测触点

在程序中使用操作数为符号名 ED 的边缘检测触点（8 个字可使用 128 个）：



附录 B 保持运行状态写程序注意事项

当使用保持运行状态写程序功能时，要注意以下事项：

- 1、不要增加或删除边缘检测触点（包括 MDO 指令），即要保持边缘检测触点使用个数不变。
- 2、不要改变边缘检测触点（包括 MDO 指令）之间的相互顺序。
- 3、若使用操作数为 DMx.y 位元件的边缘检测触点（即指定由位 DMx.y 来保存上个扫描周期该边缘检测触点前的逻辑运算结果），则不受上面两条影响，可自由使用。