

符合 IEC 标准的蓝普锋 RPC2000 系列 PLC 编程语言

蓝普锋 RPC2000 系列 PLC 的编程软件 RunPro 符合国际电工委员会（IEC）的 IEC61131-3 标准，支持 LD、FBD、IL、ST 和 SFC 等符合 IEC 标准的 PLC 编程语言。

1 梯形图

PLC 的梯形图（Ladder Diagram）编程语言简称为 LD 语言。LD 语言是一种图形化的编程语言。采用 LD 语言编写 PLC 程序，可以方便地构造逻辑运算。LD 主要由触点、线圈、功能块和连接线等编程元件组成。LD 通过水平线和垂直线连接成平面网状图。一般称最左边的垂直线为“能量线”，其状态永远为真（TRUE）。从最左边的“能量线”开始，各个编程元件以一定的规则互相连接，形成一些“节”、“段”或“网络”，完成特定的逻辑运算。梯形图编程语言编辑器如图 1 所示。

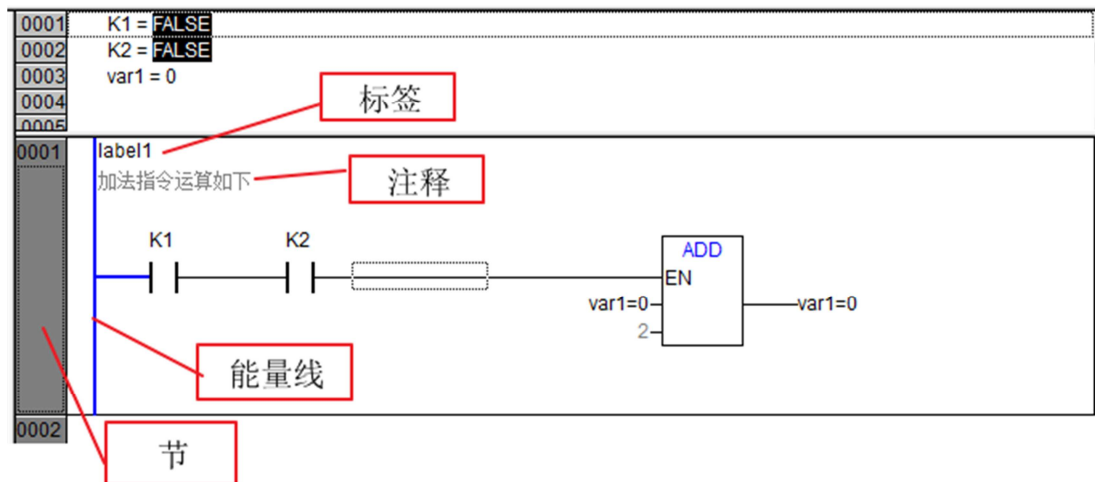


图 1 梯形图编程语言编辑器

在 RunPro 软件工作区域的梯形图部分，点击鼠标右键，在右键快捷菜单中，可以看到当前状态的常用命令，如图 2 所示。

- “前节（N）”命令：在光标所在当前节的前面增加空节。
- “后节（E）”命令：在光标所在当前节的后面增加空节。
- “串联（O）”命令：在光标位置增加串联触点。
- “并联触点（P）”命令：在光标位置增加并联触点。
- “功能块（F）”命令：在光标位置增加功能块。
- “线圈（L）”命令：在光标位置增加输出线圈。
- “使能运算符（E）”命令：在光标位置增加带有使能端的运算符。
- “插入选项（K）”命令：在光标位置增加输入、输出、运算符或赋值。
- “跳转（J）”命令：在光标位置设置一个跳转。如果条件为真，则跳转。
- “返回（R）”命令：在光标位置设置一个返回。如果当前 POU 被其它 POU 调用，当返回条件为真时，返回到调用它的 POU。
- “注释（C）”命令：可以在每一节中加入注释，增加程序的可读性。

前节 (N)	
后节 (E)	Ctrl+T
触点 (Q)	Ctrl+K
并联触点 (P)	Ctrl+R
功能块 (F)	Ctrl+B
线圈 (L)	Ctrl+L
使能运算符 (E)	
插入选项 (K)	▶
跳转 (J)	
返回 (R)	
注释 (C)	

图 2 右键快捷菜单

2 功能块图

PLC 的功能块图（Function Block Diagram）编程语言简称为 FBD 语言。FBD 语言是一种图形化的编程语言，与 LD 语言类似。采用 FBD 语言编写的 PLC 程序，由一系列“节”组成，每“节”由一系列“方块”组成。每“节”完成一段相对独立的运算。采用 FBD 语言编写 FBD 程序，可以进行逻辑表达式、算术表达式、功能块、连线、输入、输出、跳转和返回等计算和操作。功能块图编程语言编辑器如图 3 所示。

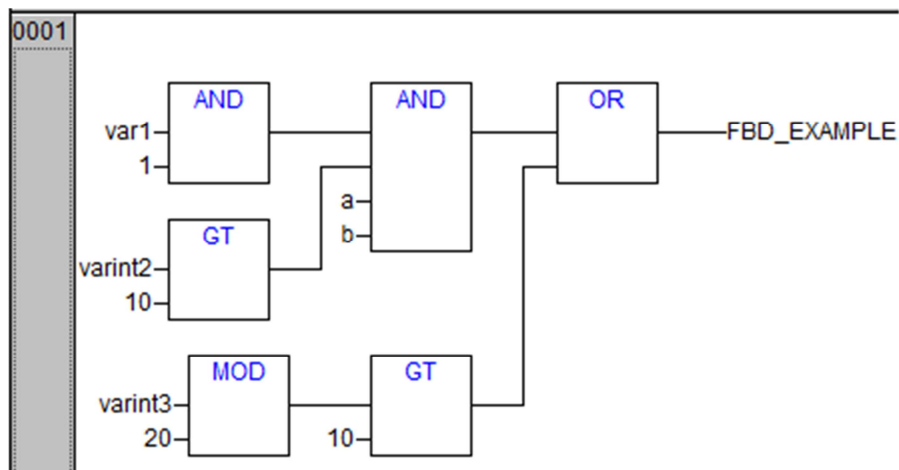
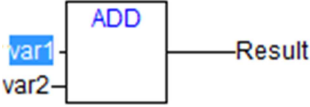
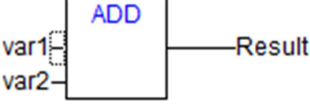
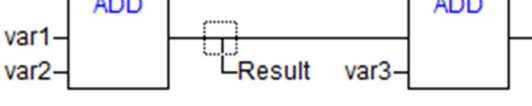
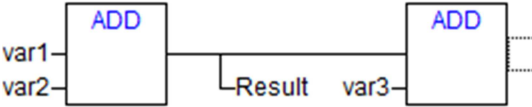
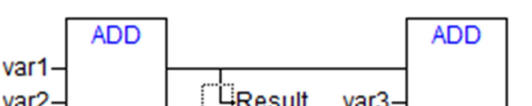


图 3 功能块图编程语言编辑器

采用 FBD 语言编写的 PLC 程序时，可以通过虚线矩形框来识别和确认当前的光标位置。当前可能的光标位置如表 1 所示。在使用“插入”菜单下的各项命令时，需要首先识别和确认当前的光标位置。

下面对“插入”菜单下的各项命令进行说明。

表 1 当前可能的光标位置与菜单命令

光标位置	菜单命令	功能块图示例
光标位置 1	文本	
光标位置 2	输入	
光标位置 3	运算符、函数或功能块	
光标位置 4	输出，后接赋值或跳转	
光标位置 5	赋值、跳转或返回的 线段交叉点	
光标位置 6	节的最后输出端	
光标位置 7	赋值前面的交叉线	

(1) 添加“输入”

快捷菜单：。在当前光标位置插入一个函数或功能块的输入端。

对于某些运算符，输入的数量是变化的，有时需要扩展运算符的输入。例如 ADD 可以是两个数相加，也可以是更多的数相加。选中输入（光标位置 2），插入的新输入成为功能块的第一个输入。如果要插入一个位于末端位置的输入，必须选中功能块本身（光标位置 3）。插入的输入缺省值为文本“???”。点击选中文本，改变成所需要的常量或变量。对此可以使用输入辅助快捷键 F2。

在需要的时候，增加功能块的输入，可以大大简化程序，如图 4 所示。

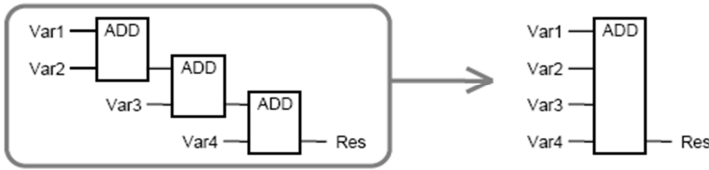


图 4 扩展输入端

如果输入端“???”位于运算符右边的分支与函数的第一个输入端相连，但此时需要改为与第二个输入端相连。可以选中第一个输入，执行命令“编辑”/“剪切”。然后选中第二个输入，执行命令“编辑”/“粘贴”。这样，此分支连接到第二个输入端。

（2）添加“输出”

快捷菜单：。在当前光标位置插入一个功能块的输出端。

对于某些功能块，输出的数量可能是变化的。此命令可以扩展功能块的输出。

（3）插入“运算符”

快捷菜单：。在当前位置插入一个运算符。

如果输入被选中（光标位置 2），运算符被插入到输入的前面。此运算符的第一个输入连接到选中输入的左边分支，新运算符的输出连接到选中的输入。

如果输出被选中（光标位置 4），运算符被插入到输出的后面。此运算符的第一个输入连接到选中的输出，新运算符的输出连接到原来连接的分支上。

如果运算符、函数或功能块被选中（光标位置 3），旧的元素被新的运算符代替。分支的连接与没有被替换之前的情形相同。如果旧元素有比新运算符分支更多，那么多余的分支将被删除。

如果一个跳转或返回被选中，那么运算符会插入到跳转或返回之前。运算符的第一个输入与选中的元素的左边的分支相连，运算符的输出连接到选中元素的右边的分支。

如果“节”的最后一个光标位置被选中（光标位置 6），那么运算符会被插入到最后一个元素之后，运算符的第一个输入连接到选中位置的左边分支。

被插入的运算符缺省总是 AND。选中关键字，可以把 AND 转换成其它的运算符。也可以借助“提示输入”，或选中关键字并使用输入辅助快捷键 F2，从运算符类型列表中选择所要的运算符。新运算符的输入端将被自动连接到前面分支上。所有没有连接的输入端都标以“???”，可以将其删除，或改变成所要的常量或变量。

（4）添加“赋值”

快捷菜单：。插入一个赋值符号，将运算符的结果赋值输出。

根据当前选中位置的不同，插入的位置也有所不同。具体地说，赋值符号可以插入到输入（光标位置 2）的前面、输出（光标位置 4）的后面、交叉线（光标位置 5）的前面、节的最后（光标位置 6）的后面。

为了给一个已存在的赋值插入一个附加赋值，使用“输出”命令。

（5）设置“跳转”

快捷菜单：。设置一个跳转。如果条件为真，则跳转到指定位置。

根据选中的位置，可以设置到输入的前面（光标位置 2）、输出的后面（光标位置 4）、交叉线的前面（光标位置 5）、节的最后（光标位置 6）的后面。

对于设置的跳转，可以被赋给它的标签代替。

（6）插入“返回”

快捷菜单：。插入一个返回。

在当前 POU 被其它 POU 调用，且返回条件为真时，返回到调用它的 POU。

根据选中的位置，可以插入到选中的输入的前面（光标位置 2）、选中的输出的后面（光标位置 4）、选中的交叉线前面（光标位置 5）、节的最后（光标位置 6）。

（7）“反向”操作

快捷菜单：。反向输入、输出、跳转或返回指令。

反向的图标是一个连接的小圆。如果光标选中输入（光标位置 2），那么输入被反向。

如果选中输出（光标位置 4），那么输出被反向。如果选中跳转或返回，那么跳转或返回的输入端被反向。再次执行反向命令，则取消反向。

（8）“置位/复位”操作

快捷菜单：。定义输出为置位输出或复位输出。

置位输出显示为 S，复位输出显示为 R，如图 5 所示。如果输出 TRUE 值，则置位端设为 TRUE 并一直保持此值。如果输出 FALSE，则复位端设为 FALSE 并一直保持此值。此命令多次交替执行，输出在置位、复位和正常值之间交替。

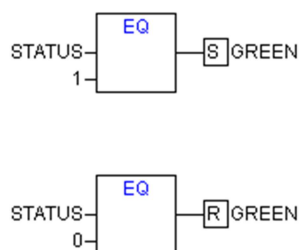


图 5 置位与复位

举例：采用 FBD 语言编写的 PLC 简单应用示例如图 6 所示，该程序可以产生“1s 断 2s 通”的脉冲信号。

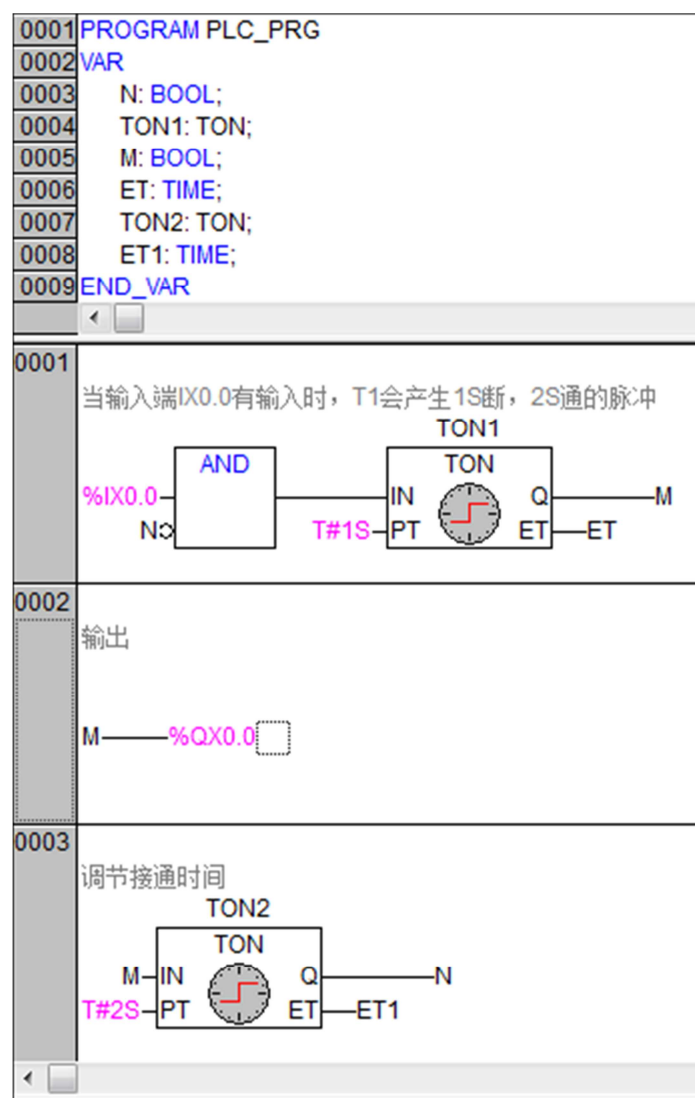


图 6 采用 FBD 语言编写的 PLC 简单应用示例

3 指令列表

PLC 的指令列表（Instruction List）编程语言简称为 IL 语言。IL 语言是一种汇编语言风格的编程语言，程序不易阅读，但执行速度最快。IL 语言包含一系列的指令，每条指令占据一行，包含一个运算符和一个或多个用逗号隔开的操作数。操作数之间用逗号分隔。在每行的开始部分，可以有标签，标签后要有冒号。在每行的结束部分，可以有注释，注释用标识符“(* *)”表示，注释的内容放在标识符的中间。每行指令之间可以插入空行。IL 语言编辑器是一种文本编辑器，具有 Windows 文本编辑器的常见功能。点击菜单栏或鼠标右键，可以进行 IL 语言的编辑。

在 RunPro 软件 IL 语言编辑器中，打开“插入”菜单，可以看到当前状态的常用命令，如图 7 所示，选择所要插入的内容。

“操作符（O）”命令：在编程区点击此命令，从弹出窗口中选择所需的操作符。

“操作数（P）”命令：在编程区点击此命令，从弹出窗口中选择所需的操作数。

“函数（F）”命令：在编程区点击此命令，从弹出窗口中选择所需的函数。

“功能块（B）”命令：在编程区点击此命令，从弹出窗口中选择所需的功能块。



图 7 IL 语言编辑器的“插入”菜单

IL 语言支持 C 和 N 两种修饰符。C 表示条件执行，只有当前一个表达式的值为真(TRUE)时，指令才被执行。N 与 JMP、CAL 和 RET 连用时表示条件非执行，只有当前一个表达式的值为假 (FALSE) 时，指令才被执行。在其它情况下，N 表示操作数取负。表 2 列出了 IL 语言的所有运算符及其修饰符和相应意义。

表 2 IL 语言的运算符

运算符	修饰符	意义
LD	N	将操作数赋予当前值
ST	N	将当前值赋予操作数
S		如果当前结果是 TRUE，把布尔型操作数置为 TRUE
R		如果当前结果是 TRUE，把布尔型操作数置为 FALSE
AND	N、(	位逻辑运算符 AND
OR	N、(	位逻辑运算符 OR
XOR	N、(	位逻辑运算符 XOR
ADD	(	加
SUB	(	减
MUL	(	乘
DIV	(	除
GT	(	>大于判断
GE	(	>=大于等于判断
EQ	(	=等于判断
NE	(	<>不等于判断
LE	(	<=小于等于判断
LT	(	<小于判断
JMP	C、N	转移到标签
CAL	C、N	调用其它 POU
RET	C、N	退出 POU 并且返回到调用处
)		计算延迟操作

采用 IL 语言编写的 PLC 简单应用示例如下所示。

举例：下面是一个用 IL 语言实现的简单运算程序。

声明：

VAR

A: INT;

B: INT;

C: INT;

END_VAR

程序：

```
LD      100  (*将数字 100 赋予当前值*)
SUB     A    (*当前值与变量 A 减运算后结果存入当前值*)
GE      B    (*当前值与变量 B 进行大于等于比较*)
JMPC Next1  (*上一表达式结果为真，则跳转至标志 Next1 处*)
LD      A    (*将变量 A 的值赋予当前值*)
ADD B    (*当前值与变量 B 加运算后结果存入当前值*)
ST      C    (*将当前值赋予变量 C*)
JMPNext2  (*无条件跳转至标志 Next2 处*)
Next1:   (*标志*)
LD      A    (*将变量 A 的值赋予当前值*)
ADD B    (*当前值与变量 B 加运算后结果存入当前值*)
ST      C    (*将当前值赋予变量 C*)
Next2:   (*标志*)
```

举例：以下是利用修饰符进行 IL 编程的例子。

```
LD      TRUE  (*将 TRUE 赋予当前值*)
ANDN    BOOL1 (*BOOL1 变量取反后与当前值进行与运算*)
JMPC    example (*如果结果是真，那么跳转到标签 “example” 处*)
LDN     BOOL2 (*将 BOOL2 取反后赋予当前值*)
ST      ERG   (*将当前值存入 ERG*)
```

example:

```
LD      BOOL2(*将 BOOL2 赋予当前值 *)
ST      ERG(*将当前值存入 ERG*)
```

如果在运算符之后插入括号，那么括号里的值可以看成是一个操作数。例如：

```
LD      5
MUL     6
ADD     7
ST      ERG
```

运行后 ERG 的值是 37。但是，如果加入一对圆括号：

```
LD      5
MUL     (6
ADD     7
)
```

ST ERG

运行后 ERG 的值是 65，MUL 运算符只有到 “)” 才执行，等同于执行 MUL 13。

举例：该 IL 程序可以产生“1s 断 2s 通”的脉冲信号。

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     TI:TON;
0004     ET:TIME;
0005     M:BOOL;
0006     T2:TON;
0007     ET1:TIME;
0008     N:BOOL;
0009     T1: TON;
0010 END VAR

0001 (*当IX0.0接通时,T1会产生1S断2S通的脉冲*)
0002 LD    %IX0.0
0003 ANDN  N
0004 ST    T1.IN
0005 CAL   T1(PT:=T#1S) (*通过T1延时，产生1S断*)
0006 LD    T1.ET
0007 ST    ET
0008 LD    T1.Q
0009 ST    M
0010 LD    M
0011 ST    T2.IN
0012 CAL   T2(PT:=T#2S) (*通过T2延时，保持T1的输出，产生2S通*)
0013 LD    T2.ET
0014 ST    ET1
0015 LD    T2.Q
0016 ST    N
0017 LD    M
0018 ST    %QX0.0
```

图 8 采用 IL 语言编写的 PLC 简单应用示例

4 结构文本

PLC 的结构文本(Structured Text)编程语言简称为 ST 语言。ST 语言是一种类似于 PASCAL 或 BASIC 等高级语言风格的编程语言,容易实现复杂的计算功能。ST 语言编辑器是一种文本编辑器,具有 Windows 文本编辑器的常见功能。ST 语言程序由一系列的语句组成,每条语句以分号结束。ST 语言程序可以有注释,注释用标识符“(* *)”表示,注释的内容放在标识符的中间。

ST 语言中的表达式由运算符和操作数组成。操作数可以是常量、变量、函数调用或另一个表达式。表达式的计算通过执行具有不同优先级的运算符完成。具有最高优先级的运算符先被执行,然后依次执行下一个优先级的运算符,直到所有的运算符被执行完毕。对于具有相同优先级的运算符,按照从左到右的顺序依次执行。ST 语言的运算符如表 3 所示,ST 语言的指令如表 4 所示。

ST 语言具有简洁和易读的特点。“结构文本”这个名称就已经表明,ST 语言用于结构化编程。也就是说,ST 语言为结构化编程提供了预先确定的结构。

表 3 ST 语言的运算符

运算	符号	优先级
放入圆括号	(表达式)	最高优先级
函数调用	函数名(参数列表)	
求幂	EXPT	
求负	-	
取非	NOT	
乘积	*	
除	/	
取模	MOD	
加	+	
减	-	
小于	<	
大于	>	
小于等于	<=	
大于等于	>=	
相等	=	
不等	<>	
逻辑与	AND	
逻辑异或	XOR	
逻辑或	OR	最低优先级

表 4 ST 语言的指令

序号	指令	举例
1	赋值	A := B; CV := CV + 1; C := SIN(X);
2	调用功能块	TP(IN := %IX0.1, PT := T#10); A := TP.Q;
3	返回	RETURN;
4	IF 条件	D := B * B; IF D < 100 THEN C := A; ELSIF D = 100 THEN C := B; ELSE C := D; END_IF;
5	CASE 条件	CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE;
6	FOR 循环	J := 101; FOR I := 1 TO 100 BY 2 DO IF ARR[I] = 50 THEN J := I; EXIT; END_IF; END_FOR;
7	WHILE 循环	J := 1; WHILE J <= 100 AND ARR[J] <> 50 DO J := J + 2; END_WHILE
8	REPEAT 循环	J := -11; REPEAT J := J + 2; UNTIL J = 51 OR ARR[J] = 50 END_REPEAT;
9	退出程序	EXIT;
10	空指令	;

（1）赋值

执行赋值操作时，等号左边是操作数、变量或地址，等号右边是被赋予的表达式的值。

举例：

变量声明部分：

Var1: REAL;

Var2: REAL;

程序部分：

Var1 := Var2 * 100;

（2）调用功能块

通过写入功能块实例的名字来调用功能块。在调用功能块时，需要在功能块实例名后面的圆括号中赋给参数值。

举例：

变量声明部分：

TP1: TP;

VarBOOL1: BOOL;

VarBOOL2: BOOL;

程序部分：

TP1(IN := VarBOOL1, PT := T#10s); (*输入参数 IN 和 PT 分别设定时钟脉冲的触发信号和高电平的长度*)

VarBOOL2 := TP1.Q; (*输出脉冲值 Q 赋给变量 VarBOOL2*)

（3）返回指令

返回指令可以根据条件退出 POU。

（4）IF 条件指令

使用 IF 条件指令可以检查条件，根据条件执行相应的指令。

语法：

IF <逻辑表达式> THEN

 <IF 指令>

{ELSIF <逻辑表达式 1> THEN

 <ELSE IF 指令 1>

ELSIF <逻辑表达式 n> THEN

 <ELSE IF 指令 n>

ELSE

 <ELSE 指令>}

END_IF;

其中 “{ }” 的部分是可选项。

如果<逻辑表达式>返回 TRUE，那么只有<IF 指令>被执行，其它的指令不被执行。同样，

从<逻辑表达式 1>开始，相继执行逻辑表达式，直到其中一个表达式返回 **TRUE** 为止，返回 **TRUE** 的逻辑表达式对应的指令被执行。

如果没有逻辑表达式生成 **TRUE**，那么只有<ELSE 指令>被执行。

举例：

变量声明部分：

A: WORD;

B: BOOL;

程序部分：

IF A < 20 THEN

 B := TRUE;

ELSE

 B := FALSE;

END_IF;

结果说明：如果 A 小于 20，则 B 为 **TRUE**，反之 B 为 **FALSE**。

举例：下面是一个用 **ST** 语言实现简单运算的小程序：

变量声明部分：

PROGRAM PLC

VAR

 A:BOOL;

 B:BOOL;

 C:INT;

END_VAR

程序部分：

IF A = TRUE THEN

 C := 1;

ELSIF B = TRUE THEN

 C := 2;

ELSE

 C:=3;

END_IF

（5）CASE 指令

使用 **CASE** 指令可以在结构中用一个相同的条件变量表示几个条件指令。

语法：

CASE <Var1> OF

 <Value1>: <指令 1>

 <Value2>: <指令 2>

 <Value3, Value4, Value5>: <指令 3>

 <Value6 .. Value10>: <指令 4>

 <Value n>: <指令 n>

ELSE

<ELSE 指令>

END_CASE;

根据下面的模型来执行 CASE 指令：

如果变量<Var1>的值等于< Value i>的值，那么<指令 i>被执行。

如果变量<Var1>没有任何指定的值，那么<ELSE 指令>被执行。

如果变量的几个值都需要执行相同的指令，那么可以把几个值相继写在一起，并且用逗号分开。这样，就会有相同的执行指令。

如果对于变量的一个范围需要执行相同的指令，可以写入初值和终值，中间用两个点分开。这样，条件就会有相同的执行。

举例：

CASE INT1 OF

1,2,3, 5: BOOL1 := TRUE; BOOL3 := FALSE;

4: BOOL2 := FALSE; BOOL3 := TRUE;

10..20: BOOL1 := TRUE; BOOL3:= TRUE;

ELSE

BOOL1 := NOT BOOL1;

BOOL2 := BOOL1 OR BOOL2;

END_CASE;

（6）FOR 循环指令

使用 FOR 循环指令可以编写循环过程。

语法：

INT_Var: INT;

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step Size>} DO

<Instructions>

END_FOR;

其中“{ }”的部分是可选项。

只要计数<INT_Var>不大于<END_VAULE>，指令就会被执行。指令执行之前，首先检查这个条件，如果<INIT_VALUE>大于<END_VALUE>，指令就永远不会被执行。当指令被执行时，<INT_Var>总是增加步长<Step Size>。步长可以是任意的整数值。如果不写步长，缺省值是 1。当<INT_Var>大于<END_VALUE>时，循环结束。

注意，<END_VALUE>一定不能等于计数变量<INT_Var>的极限值。如果计数变量<INT_Var>的类型是 SINT，其范围是-128-127，而<END_VALUE>的极限值是 127，则会进入死循环。

举例：

FOR Counter := 1 TO 4 BY 1 DO

Var1 := Var1 * 2;

END_FOR;

Erg := Var1;

如果 Var1 的缺省值是 1，那么循环结束后，Var1 的值为 16。

（7）WHILE 循环指令

在使用方面，WHILE 循环指令与 FOR 循环指令类似。二者不同的是，WHILE 循环指令的结束条件可以是任意的逻辑表达式。可以指定一个条件，当条件满足时，循环被执行。

语法：

```
WHILE <逻辑表达式> DO
    <指令>
END_WHILE;
```

只要<逻辑表达式>的值返回 TRUE，那么<指令>就会被重复执行。在第一次计算时，如果<逻辑表达式>的值已经是 FALSE，那么<指令>永远不会被执行。如果<逻辑表达式>的取值永远不会是 FALSE，那么<指令>被无休止地执行，产生一个相对时间的延迟，即死循环。

举例：

```
WHILE Counter <> 0 DO
    Var1 := Var1*10;
    Counter := Counter - 1;
END_WHILE
```

（8）REPEAT 循环指令

在使用方面，REPEAT 循环指令与 WHILE 循环指令不同。在指令执行以后，REPEAT 循环指令才检查结束条件。无论结束条件怎样，循环至少执行一次。

语法：

```
REPEAT
    <指令>
UNTIL <逻辑表达式>
END_REPEAT;
```

直到<逻辑表达式>的值返回 TRUE，<指令>才停止执行。在第一次计算时，如果<逻辑表达式>产生 TRUE，那么<指令>只被执行一次。如果<逻辑表达式>不会产生 TRUE，那么<指令>被无休止地执行，产生一个相对时间的延迟，即死循环。

举例：

```
REPEAT
    Var1 := Var1 * 10;
    Counter := Counter - 1;
UNTIL Counter = 0
END_REPEAT;
```

在一定意义上，WHILE 和 REPEAT 循环比 FOR 循环功能更强大。因为不需要在执行循环之前计算循环次数。因此，在有些情况下，用这两种循环就可以了。然而，如果清楚知道循环次数，那么 FOR 循环更好。

（9）EXIT 指令

如果 EXIT 指令出现在 FOR 循环指令、WHILE 循环指令或 REPEAT 循环指令中，那么无论循环结束条件如何，当 EXIT 出现时，循环终止。

举例：比较下面两段分别用 ST 语言和 IL 语言实现 2 的十次乘幂的循环的程序代码。

用 ST 语言：

变量声明部分：

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    Counter: INT := 10;
```

```
    Var1: INT := 1;
```

```
    Var2: INT;
```

```
END_VAR
```

程序部分：

```
WHILE Counter <> 0 DO
```

```
    Var1 := Var1*2;
```

```
    Counter := Counter - 1;
```

```
END_WHILE
```

```
Var2 := Var1;
```

用 IL 语言：

变量声明部分：

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    Counter: INT := 10;
```

```
    Var1: INT := 1;
```

```
    Var2: INT;
```

```
END_VAR
```

程序部分：

```
Loop:
```

```
    LD      Counter
```

```
    NE      0
```

```
    NOT
```

```
    JMPC    END_LOOP
```

```
    LD      Var1
```

```
    MUL     2
```

```
    ST      Var1
```

```
    LD      Counter
```

```
    SUB     1
```

```
    ST      Counter
```

```
    JMP     Loop
```

```
End_Loop:
```

```
    LD      Var1
```

```
    ST      Var2
```

可以看出，用 ST 语言实现的循环不仅简洁，而且易于阅读。

5 顺序功能图

PLC 的顺序功能图（Sequential Function Chart）编程语言简称为 SFC 语言。SFC 语言是一种图形化的编程语言，用来描述程序中不同动作的时间顺序。采用 SFC 语言编写的 PLC 程序，由一系列的“步”和“转移”组成，“步”定义动作，“转移”控制顺序。顺序功能图编程语言编辑器如图 9 所示。

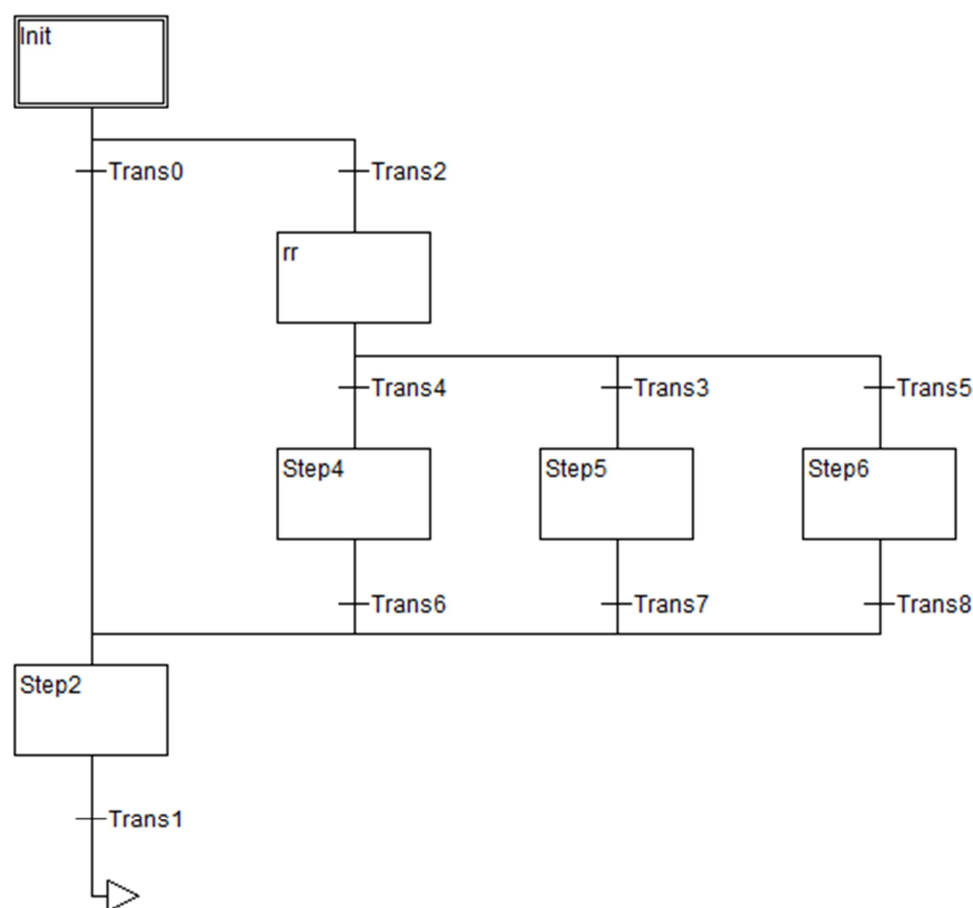


图 9 顺序功能图编程语言

（1）步

SFC 语言包含一系列的步，这些步通过有向的连接彼此联系。步分为简化步和 IEC 步。

简化步包含一个动作和一个显示步动作的标志。如果步包含动作，则在步的右上角出现一个小三角形。

IEC 步包含一个或多个动作和逻辑变量。新插入的步是否为 IEC 步，取决于是否选择了“高级”/“使用 IEC 步 (I)”命令。必须添加特殊的 SFC 库 `iecsfc.lib` 才能够使用 IEC 步。

（2）动作

动作是使用其它语言实现的一系列指令，既可以用 IL 或 ST 语言实现的指令语句，也可以用 LD、FBD 或 SFC 实现的图形网络。

对于简化步，动作总是和步直接相关。用鼠标双击动作所属的步，可以进行编辑。

对于 IEC 步，在对象组织器中选中 SFC 程序，点击右键，选择“添加动作”命令创建新

动作。可以赋给 IEC 步多个动作，同时这些动作也可以被多个步重复使用。

IEC 步的动作显示在步的右边框中。左边字段包含可能有时间常量的限定符，右边字段包含动作名即逻辑变量名。包含五个动作的 IEC 步示例如图 10 所示。

IEC 的动作分散在步中。在它们所属的 POU 中，这些动作可以被重复使用。使用“高级”/“关联动作”命令添加动作，使用“高级”/“清除动作或转移”命令删除已添加的动作。

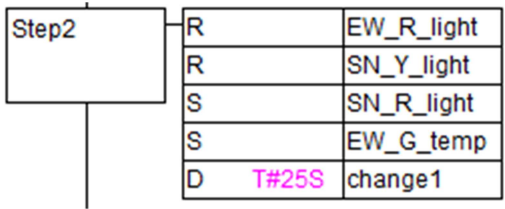


图 10 顺序功能图编程语言示例

（3）入口动作和出口动作

在顺序功能图编程语言编辑器中，点击“添加入口动作”和“添加出口动作”命令，可以在步中加入入口动作和出口动作。

对于入口动作，只有当步在活动状态时，立即执行一次。

对于出口动作，只在步不活动之前，执行一次。

有入口动作的步，在左下角有“E”标志进入。

有出口动作的步，右下角有“X”标志退出。

可以采用任意语言实现入口动作和出口动作。用鼠标双击步的相应角，即可编辑入口或者出口动作。入口动作和出口动作步示例如图 11 所示。

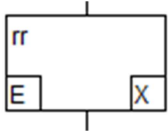


图 11 入口动作和出口动作步示例

（4）转换和转换条件

步之间的切换就是转换。只有当步的转换条件为真时，步的转换才能够进行。即前步的动作执行完后，如果有出口动作则执行一次出口动作，后步如果有入口动作则执行一次后步的入口动作，然后按照控制周期执行该活动步的所有动作。

转换条件可以是逻辑变量、逻辑地址或逻辑常量，也可以是其它语言编程实现的逻辑。

（5）活动步

调用 POU 之后，初始化步（双边的步）的动作首先执行。动作被执行的步称为活动步。每次循环执行活动步的动作。

在线模式下，活动步以蓝色显示。为了更容易地跟随过程，在线模式下，所有的活动动作像活动步一样以蓝色显示。每次循环以后做一次检查，查看哪个动作是活动的。在一个控制周期中，执行活动步的所有动作。之后，如果下步的转换条件是 TRUE，下步将变成活动步，在下个周期执行。

（6）限定符

把 IEC 步和动作关联，需要使用限定符，限定符对应的含义如表 5 所示。

表 5 SFC 语言的限定符

限定符	英文名称	中文名称
N	Non-stored	不存储
R	overriding Reset	复位
S	Set (Stored)	置位
L	time Limited	限定时间
D	time Delayed	延迟时间
P	Pulse	脉冲
SD	Stored and time Delayed	存储与延迟时间
DS	Delayed and Stored	延迟时间与存储
SL	Stored and time limited	存储与限定时间

（7）隐含变量

在 SFC 中，可以隐含声明的变量。每步的标志都存储了步的状态。

对于简化步，步标志为<步名>。对于 IEC 步，步标志为<步名>.x。

当步是活动状态的时候，步标志的值为 TRUE。当步是非活动状态的时候，步标志的值为 FALSE。

对于 IEC 步，可以使用步标志<步名>.x 询问步是否是活动的，也可以使用隐含变量<步名>.t 用来询问步的活动时间。

（8）标志

在 SFC 中，步的活动状态时间取决于它的属性状态时间，有时会设置一些特殊标志。同样，可以设置变量来控制顺序功能图中的程序流。要使用这些标志，必须在全局或局部变量中作为输入或者输出变量声明这些标志。例如，如果在一个 SFC 程序组织单元中一个步激活的时间超过了它定义的属性，那么就会设置一个标志符，通过变量“SFCErrror”可以访问到这个标志符，此时 SFCErrror 的值为 TRUE。

6 连续功能图

PLC 的连续功能图（Continuous Function Chart）编程语言简称为 CFC 语言。CFC 语言是一种图形化的编程语言。CFC 语言基于 FBD 语言，但是没有“节”的限制，元素的摆放位置更加灵活。连续功能图编程语言编辑器如图 12 所示，通过菜单栏或鼠标右键可以进行编辑，元素可以摆放在编程区的任意位置。用鼠标拖拽在元素之间的连线，当元素的位置移动时，编辑器会自动调整连线的长度。如果元素之间距离较小，普通连线会变为红色连线。如果元素之间距离较大，红色连线会变为普通连线。

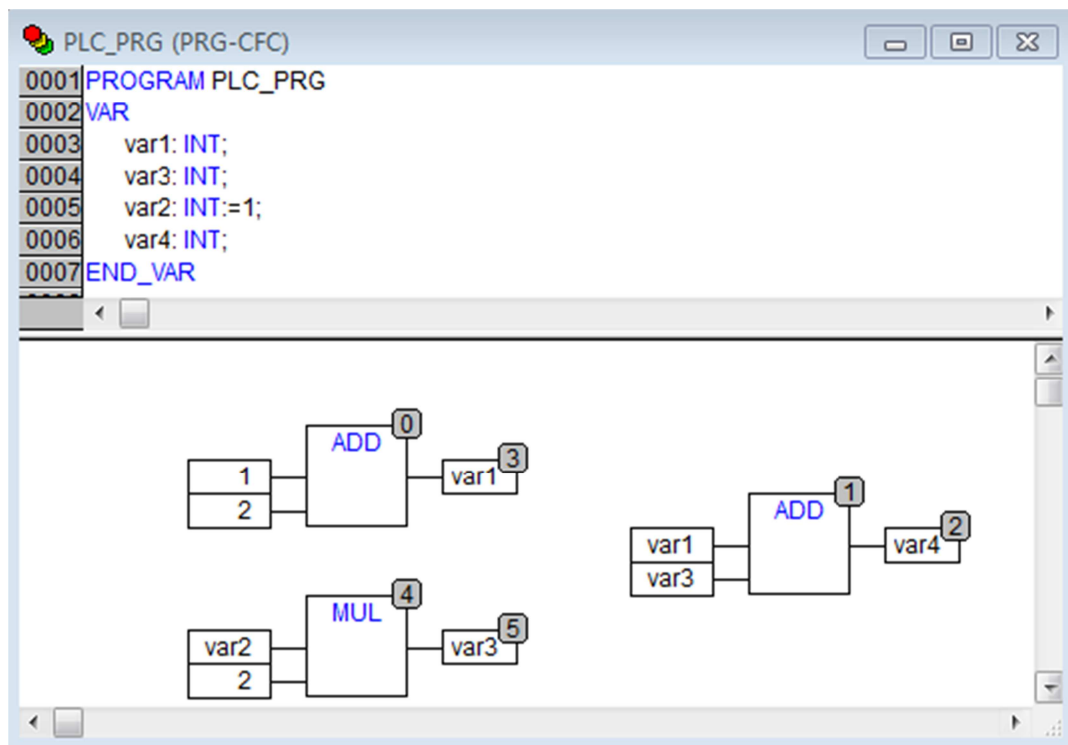


图 12 连续功能图编程语言编辑器