



程序执行的控制

- 程序的调用
- 例行程序内的逻辑控制
- 停止程序执行



1、程序的调用

- ◆ **Procall** 调用例行程序
- ◆ **CallByVar** 通过带变量的例行程序名称调用例行程序
- ◆ **RETURN** 返回原例行程序



(1) ProCall

Procedure {Argument};

Procedure :例行程序名称。(Identifier)

{Argument} :例行程序参数。(all)

应用:

机器人调用相应例行程序，同时给带有参数的例行程序中相应参数赋值。

实例:

```
Weldpipe1;
```

```
Weldpipe2 10,lowspeed;
```

```
Weldpipe3 10\speed:=20;
```



(1) ProCall

限制:

机器人调用带参数的例行程序时，必须包括所有强制性参数。

例行程序所有参数位置次序必须与例行程序设置一致。

例行程序所有参数数据类型必须与例行程序设置一致。

例行程序所有参数数据性质必须为Input, Variable或Persistent。



1. 选中“<SMT>”为要调用的例行程序的位置。
2. 在添加指令的列表中，选择“ProcCall”指令。

NewProgramName - T_NOB1/Module1/Routine2

任务与程序	模块	例行程序
20	PROC Routine2 ()	Common
21	IF di1 = 1 THEN	
22	<SMT>	=
23	ENDIF	Compact IF
24	ENDPROC	FOR
25		IF
26	ENDMODULE	MoveAbsJ
		MoveC
		MoveJ
		MoveL
		ProcCall
		Reset
		RETURN
		Set
		← 上一个
		下一个 →

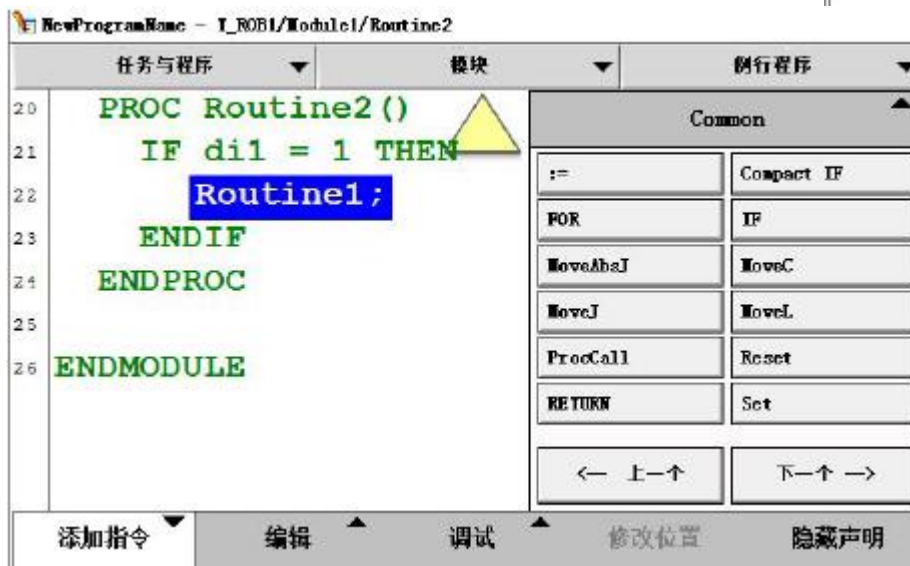
添加指令 编辑 调试 修改位置 隐藏声明



3. 选中要调用的例行程序Routine1，然后单击“确定”。



4. 调用例行程序指令执行的结果。





(2) CallByVar

CallByVar Name,Number;

Name :例行程序名称第一部分。（string）

Number: 例行程序名称第二部分。（num）

应用:

通过指令中相应数据，机器人调用相应例行程序，但无法调用带有参数的例行程序。

实例:

```
reg1:=Ginput(gi_Type);  
callByVar "proc",reg1;
```



(2) CallByVar

限制:

不能调用带参数的例行程序。

所有被调用的例行程序名称第一部分必须相同。

例如: proc1, proc2, proc3.

使用CallByVar指令调用例行程序比直接采用ProCall 调用例行程序需要更长时间。

Error Handling

ERR_REFUNKPRC

系统无法找到例行程序名称第一部分。

ERR_CALLPROC

系统无法找到例行程序第二部分。



(2) CallByVar

实例比较:

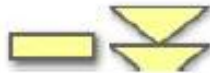

```
TEST reg1
    CASE 1:
        lf_door door_Loc;
    CASE 2:
        rf_door door_loc;
    CASE 3:
        lf_door door_loc;
    CASE 4:
        rr_door door_loc;
    EDFAULT:
        EXIT;
ENDTEST
CallByVar "proc",reg1;
%"proc"+NumToStr(reg1,0)% door_loc;
```



(3) RETURN

RETURN 返回例行程序指令，当此指令被执行时，则马上结束本例行程序的执行，返回程序指针到调用此例行程序的位置。

```
PROC Routine1()  
  MoveL p10, v1000, fine, tool1\WObj:=wobj1;  
  Routine2;  
  Set do1;  
ENDPROC  
PROC Routine2()  
  IF di1 = 1 THEN  
    RETURN;  
  ELSE  
    Stop;  
  ENDIF  
ENDPROC
```



当di1=1时，执行RETURN指令，程序指针返回到调用Routine2的位置并继续向下执行Set do1这个指令。



2、例行程序内的逻辑控制

- ◆ Compact IF 如果条件满足，就执行一条指令
- ◆ IF 当满足不同的条件时，执行对应的程序
- ◆ FOR 根据指定的次数，重复执行对应的程序
- ◆ WHILE 如果条件满足，重复执行对应的程序
- ◆ TEST 对一个变量进行判断，从而执行不同的程序
- ◆ GOTO 跳转到例行程序内标签的位置
- ◆ Label 跳转标签



(1) Compact IF

Compact IF 紧凑型条件判断指令用于当一个条件满足了以后，就执行一句指令。

IF flag1 = TRUE Set do1;

如果flag1 的状态为TRUE，则do1 被置位为1。

IF Condition ...

Condition

数据类型：bool

必须满足与执行指令相关的条件。



IF reg1 > 5 GOTO next;

如果reg1 大于5，在next标签处继续程序执行。

IF counter > 10 Set do1;

如果counter > 10，则设置do1信号



(2) IF

IF 条件判断指令，就是根据不同的条件去执行不同的指令。

```
IF reg1 > 5 THEN  
    Set do1;  
    Set do2;  
ENDIF
```

仅当reg1大于5时，设置信号do1和do2。



依次测试条件，直至满足其中一个条件。通过与该条件相关的指令，继续程序执行。如果未满足任何条件，则通过符合**ELSE**的指令，继续程序执行。如果满足多个条件，则仅执行与第一个此类条件相关的指令。



```
IF num1=1 THEN
    flag:=TRUE;
ELSEIF num1=2 THEN
    flag1:=FALSE;
ELSE
    Set do1;
ENDIF
```

如果num1 为1，则flag1 会赋值为TRUE。如果num1 为2，则flag1 会赋值为FALSE。除了以上两种条件之外，则执行do1 置位为1。
*条件判定的条件数量可以根据实际情况进行增加与减少。



```
IF reg1 > 5 THEN  
    Set do1;  
    Set do2;  
ELSE  
    Reset do1;  
    Reset do2;  
ENDIF
```

根据reg1是否大于5，设置或重置信号do1和do2。



变元

```
IF Condition THEN ...  
    {ELSEIF Condition THEN ...}  
[ELSE ...]  
ENDIF  
Condition  
数据类型: bool
```

必须满足关于待执行THEN和ELSE/ELSEIF
之间指令的条件。



如果a大于100则A等于100，
如果A小于0则A等于0
如果A在0-100之间则A=50

```
IF A > 100 THEN  
    A:= 100;  
ELSEIF A < 0 THEN  
    A := 0;  
ELSE  
    A:=50;  
ENDIF
```



(3) FOR

FOR 重复执行判断指令，是用于一个或多个指令需要重复执行次数的情况

```
FOR i FROM 1 TO 10 DO
```

```
    Routine1;
```

```
ENDFOR
```

例行程序Routine1，重复执行10次。



变元

```
FOR Loop counter FROM Start value TO End value [STEP Step value] DO  
.....  
ENDFOR
```

Loop counter Identifier

将包含当前循环计数器数值的数据名称。自动声明该数据。
如果循环计数器名称与实际范围中存在的任意数据相同，则将现有数据隐藏在FOR循环中，且在任何情况下均不受影响。

Start value

数据类型：Num

循环计数器的期望起始值（通常为整数值）。



End value

数据类型：Num

循环计数器的期望结束值（通常为整数值）。

Step value

数据类型：Num

循环计数器在各循环的增量（或减量）值（通常为整数值）。

如果未指定该值，则自动将步进值设置为1（或者如果起始值大于结束值，则设置为-1）。



程序的执行

- 1 评估起始值、结束值和步进值的表达式。
- 2 向循环计数器分配起始值。
- 3 检查循环计数器的数值，以查看其数值是否介于起始值和结束值之间，或者是否等于起始值或结束值。如果循环计数器的数值在此范围之外，则FOR循环停止，且程序继续执行紧接ENDFOR的指令。
- 4 执行FOR循环中的指令。
- 5 按照步进值，使循环计数器增量（或减量）。
- 6 重复FOR循环，从点3开始。



限制条件

仅可在FOR循环内评估循环计数器（数据类型为num），随之隐藏其他具有相同名称的数据和路径。其仅可通过FOR循环中的指令来进行读取。

无法使用起始值、结束值或停止值的小数值，以及FOR循环的确切终止条件



(4) WHILE

WHILE 条件判断指令，用于在给定条件满足的情况下，一直重复执行对应的指令。

```
WHILE num1>num2 DO  
    num1:=num1-1;  
ENDWHILE
```

当num1>num2 的条件满足的情况下，就一直执行num1:=num1-1 的操作。



```
WHILE reg1 < reg2 DO
```

```
...
```

```
reg1 := reg1 + 1;
```

```
ENDWHILE
```

只要 $reg1 < reg2$ ，则重复WHILE块中的指令。



变元

WHILE **Condition** DO ... ENDWHILE

Condition

数据类型: bool

必须评估为TRUE的条件为用以满足待执行WHILE块中指令的值。



程序的执行

- 评估条件表达式。如果表达式评估为TRUE值，则执行WHILE块中的指令。
- 随后，再次评估条件表达式，且如果该评估结果为TRUE，则再次执行WHILE块中●的指令。
- 该过程继续，直至表达式评估结果成为FALSE。
- 随后，终止迭代，并在WHILE块后，根据本指令，继续程序执行。
- 如果表达式评估结果在开始时为FALSE，则不执行WHILE块中的指令，且程序控制立即转移至WHILE块后的指令。



(5) TEST

TEST Test data

{CASE Test value {,Test value}:...}

[DEFAULT:...]

ENDTEST

Test data: 判断数据变量 (all)

Test value: 判断数据值 (Same as)

应用:

当前指令通过判断相应数据变量与其所对应的值，控制需要执行的相应指令。



```
TEST reg1
CASE 1,2,3 :
    routine1;
CASE 4 :
    routine2;
DEFAULT :
    TPWrite "Illegal choice";
    Stop;
ENDTEST
```

根据reg1的值，执行不同的指令。如果该值为1、2或3时，则执行routine1。如果该值为4，则执行routine2。否则，打印出错误消息，并停止执行。



程序的执行

将测试数据与第一个CASE条件中的测试值进行比较。如果对比真实，则执行相关指令。此后，通过ENDTEST后的指令，继续程序执行。

如果未满足第一个CASE条件，则对其他CASE条件进行测试等。如果未满足任何条件，则执行与DEFAULT相关的指令（如果存在）。



(6) GOTO

GOTO Label:

Label: 程序执行位置标签 (identifier)

应用:

当前指令必须与指令Label同时例用，执行当前指令后，机器人将从相应标签位置Label处继续运行程序指令。



```
GOTO next;
```

```
...
```

```
next:
```

通过以下指令，
继续程序执行。

```
reg1 := 1;
```

```
next:
```

```
...
```

```
reg1 := reg1 + 1;
```

```
IF reg1<=5 GOTO next;
```

将执行转移至next四次
(reg1= 2、3、4、5)。



```
IF reg1>100 THEN
    GOTO highvalue
ELSE
    GOTO lowvalue
ENDIF
lowvalue:
...
GOTO ready;
highvalue:
...
ready:
```

如果reg1 大于100，
则将执行转移至标
签 highvalue，否则，
将执行转移至标签
lowvalue。



实例:

IF reg1 > 100 GOTO highvalue;

Lowvalue;

...

GOTO ready;

highvalue:

...

Reg1:=1;

next:

Reg1:=reg1 + 1

IF reg1 <= 5 GOTO next;



限制：

只能使用当前指令跳跃至同一例行程序内相应位置标签Label。

如果相应位置标签Label处于指令TEST或IF内，相应指令GOTO必须同处于相同的判断指令内或其分支内。

如果相应位置标签Label处于指令WHILE或FOR内，相应指令GOTO必须同处于相同的循环指令内。



(7) Label

Label: 程序执行位置标签 (identifier)

应用:

当前指令必须与指令GOTO同时例用，执行当前指令GOTO后，机器人将从相应标签位置Label处继续运行程序指令，当前指令使用后，程序内不会显示Label字样，直接显示相应标签。

限制: 在同一个例行程序内，程序位置标签Label的名称必须唯一。标记会隐藏在其所在程序内具有相同名称的全局数据和程序。



3、停止程序执行

指令	说明
Stop	停止程序执行
EXIT	停止程序执行并禁止在停止处再开始
Break	临时停止程序的执行，用于手动调试
SystemStopAction	停止程序执行与机器人运动
ExitCycle	中止当前程序的运行并将程序指针pp复位到主程序的第一条指令，如果选择了程序连续运行模式，程序将从主程序的第一句重新执行



(1) Stop

Stop [\NoRegain];

[\NoRegain]:路径恢复参数

应用:

机器人在当前指令行停止运行，程序运行指针停留在下一行指令。可以用Start键继续运行机器人，属于临时性停止，如果机器人停止器件被手动移动后，然后直接启动机器人，机器人将警告确认路径，如果此时采用参变量[\NoRegain]，机器人将直接运行。

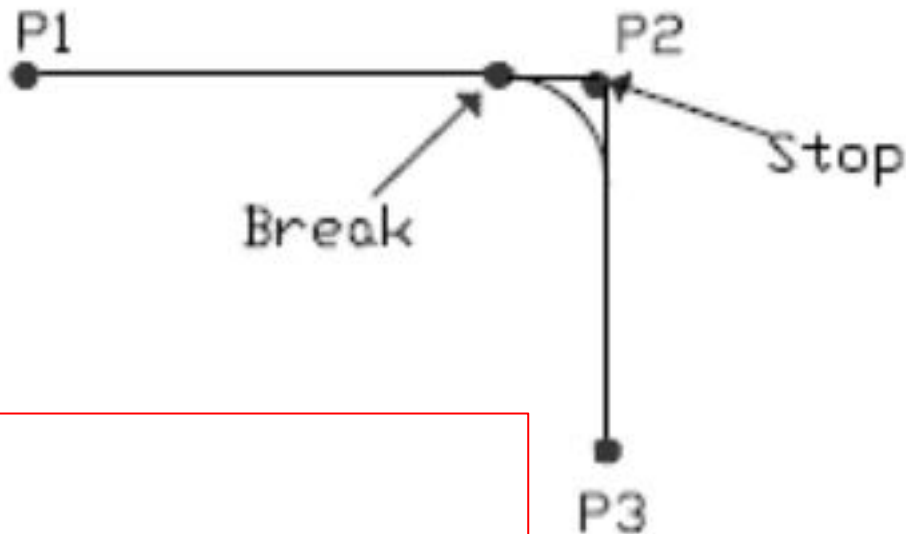


(2) Break

Break;

应用：

机器人在当前指令行立刻停止运行，程序运行指针停留在下一行指令，可以用Start键继续运行机器人。



实例:

...
Stop
...

区别:

```
MoveL p2, v100, z30, tool0;  
stop; (Break);  
MoveL p3, v100, fine, tool0;
```



(3) Exit

Exit;

应用:

机器人在当前指令行停止运行，并且程序重置，程序运行指针停留在主程序第一行。



(4) ExitCycle

ExitCycle;

应用:

机器人在当前指令行停止运行，并且设定当前循环结束，机器人自动从主程序第一行继续运行下一个循环。



实例:

```
PROC main()
    IF cyclecount=0 THEN
        CONNECT error_intno WITH error_trap;
        ISignalDI di_error,1,error_intno;
    ENDIF
    cyclecount:=cyclecount+1;
    ! Start to do something intelligent
    . . . .
ENDPROC
TRAP error_trap
    TPWrite "I will start on the next item"
    ExitCycle;
ENDTRAP
```