Robostar Robot Controller

N2 Series
Programming Manual

Version: N2-PM-C04
Issued Data: May 07, 2020

Robostar

产品保证相关

Robostart的产品在严格品质管理的监督下制造，其整体产品的保修期间自制造日起1年。在此期只对Robostar侧的过失导致的机械故障或正常使用中的设计及制造商的问题导致的故障无常服务。

以下几种情况无法保证无常服务。

(1) 已满保修期间
(2) 因贵公司或第三者不适当的修理、改造、移动或其他使用不注意导致的故障
(3) 配件及润滑油使用了本公司指定的产品以外的产品而导致的故障
(4) 因火灾、灾害、地震、风水灾害以及其他天然灾害导致的故障
(5) 因喷料及被水淹等本公司产品配置以外环境中使用导致的故障
(6) 消耗品消耗而导致的故障
(7) 因未按照使用说明书中记载的保修检验作业内容执行而导致的故障
(8) Robot修理以外的费用损失

Robostar有限公司地址及联系方式

- 总公司及工厂
  京畿道安山市常绿区水仁路 700
  (沙士洞119-38)
  700, Suin-ro, Sangnok-gu, Ansan-City,
  Gyeonggi-do, Republic of South Korea
  (15523)

- AS及产品咨询
  -营业咨询
  TEL. 031-400-3600
  FAX. 031-419-4249
  -顾客咨询
  TEL. 1588-4428

- 水原工厂
  京畿道水原市劝善区产业路155号 37
  (古索洞997)
  37, saneop-ro 155beon-gil,
  Gwonseon-gu, Suwon-City,
  Gyeonggi-do,
  Republic of South Korea (16648)

Robostar

www.robostar.co.kr

目录

图目录

第1章 程序设计

1. 基本程序结构

Robostar的Robot语言 (RRL, Robostar Robot Language)为 Robostar自创专用语言，是制作系统内的Robot作业程序时使用的语言。

一般制作程序时，根据下图中的结构制作。下图的结构为推荐的方式，但并不需要完全一模一样。

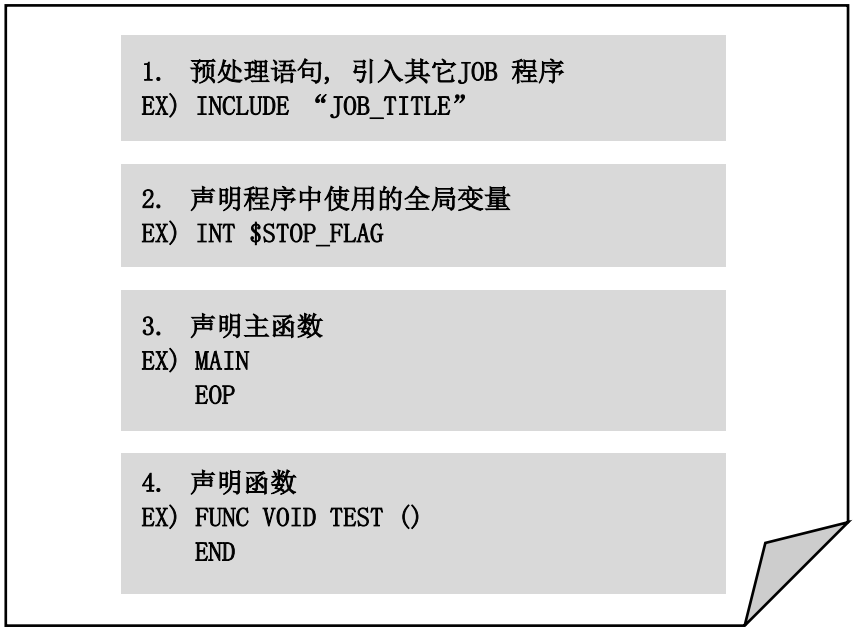


Figure 1-1 基本程序结构

1.1 主函数声明

Robostar Robot 语言程序必须仅有一个主函数. 如Figure 1-2 MAIN 函数例子，声明MAIN函数后, 在MAIN 函数末端需要写表示程序结束的EOP 语句.

1	MAIN	1. 声明主函数开始
2	SVON	2. 伺服 ON
3	JMOV GP1	3. Motion执行及时间延迟
4	DLAY 2000	
5	JMOV GP2	
6	DLAY 2000	
7	SVOF	7. 伺服 OFF
8	EOP	8. 主函数结束

Figure 1-2 主函数制作例子

1.2 一般函数声明

除了主函数，也可以声明其它函数后调用. 请参考第 4章5.2FUNC … END(函数开始/结束)章节 了解更详细的内容.

Syntax

```
FUNC <变量> <函数名> (参数, 参数, …)
                ⋮
    (RET)
END
```

1	FUNC INT TEST()	1. 声明函数开始，返回变量及函数名，变量声明
2	SVON	2. 伺服 ON
3	JMOV GP1	3. Motion执行及时间延迟
4	DLAY 2000	
5	JMOV GP2	
6	DLAY 2000	
7	SVOF	7. 伺服 OFF
8	RET 1	8. 返回值
9	END	9. 函数结束

Figure 1-3 一般函数制作例子

2. 程序调用及子程序

如果有重复特定Motion或功能的情况时，可以避免重复出现在程序中，必要时调用特定Motion或功能使用. 将这种区间放入同样程序内的方法称为子程序. 如果此区间保存在另外的程序，此程序调用相应部分则称为程序调用.

一系列作业由子程序或其它程序为单位组成时，不需要重复制作同样内容. 这种方式能减少修改说明制作程序的时间，便于阅读程序内容，为制作程序提高效率.

子程序必须保存在调用程序所在的文件中. 子程序结束后回到调用子程序的主程序的下一行. 子程序无法被其它子程序调用. 子程序可以使用全局变量，可以传递参数.

由单独文件组成的程序可以被多种程序调用，也是最常用的方法. 调用程序时通过JCALL语句指定程序名使用. JCALL语句将调用程序并移动到程序.被调用的程序最后一句执行后，回到调用程序的下一行. 被调用程序可以调用其他程序.但无法调用调用自己的程序或自己的程序. 一般可以使用全局变量，但无法传达参数.

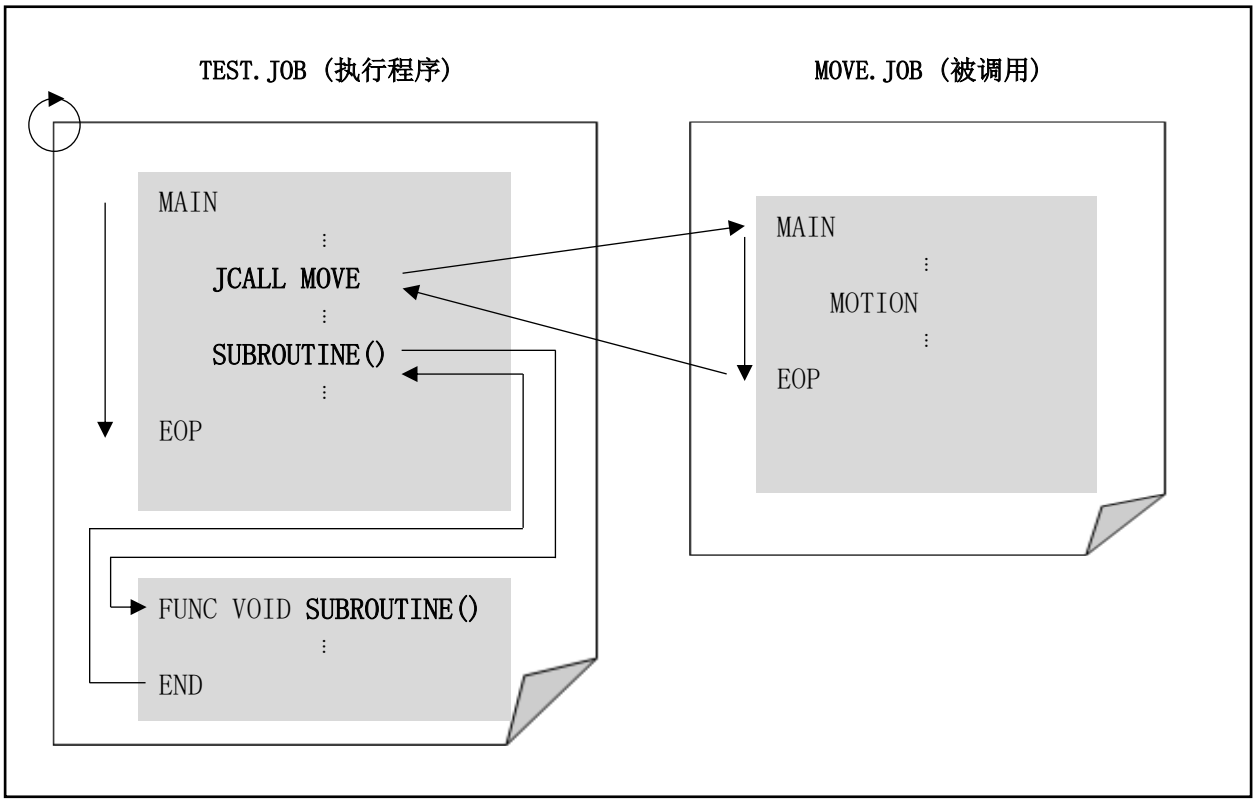


Figure 1-4 子程序和程序调用的差异

3. Multithreading

Robostar Robot语言 (RRL, Robostar Robot Language)提供可以在一个程序中同时处理多个任务的多线程功能.

如果用一个线程完成一个作业中的两个作业，则用一个线程完成一个作业后再进行下一个作业. 相反，使用多线程方式，可以同时利用多个线程完成多个作业. 多线程作业控制Robot时，可以并行执行一个JOB程序或函数，从而优化功能. 一个程序部分中断或执行长作业的情况，使用多线程功能也能继续执行其它程序使得用户体验较好. 但，两个语句同时使用相同数据时，无法预测执行结果或无法保证执行顺序，因此需要格外注意使用方式.

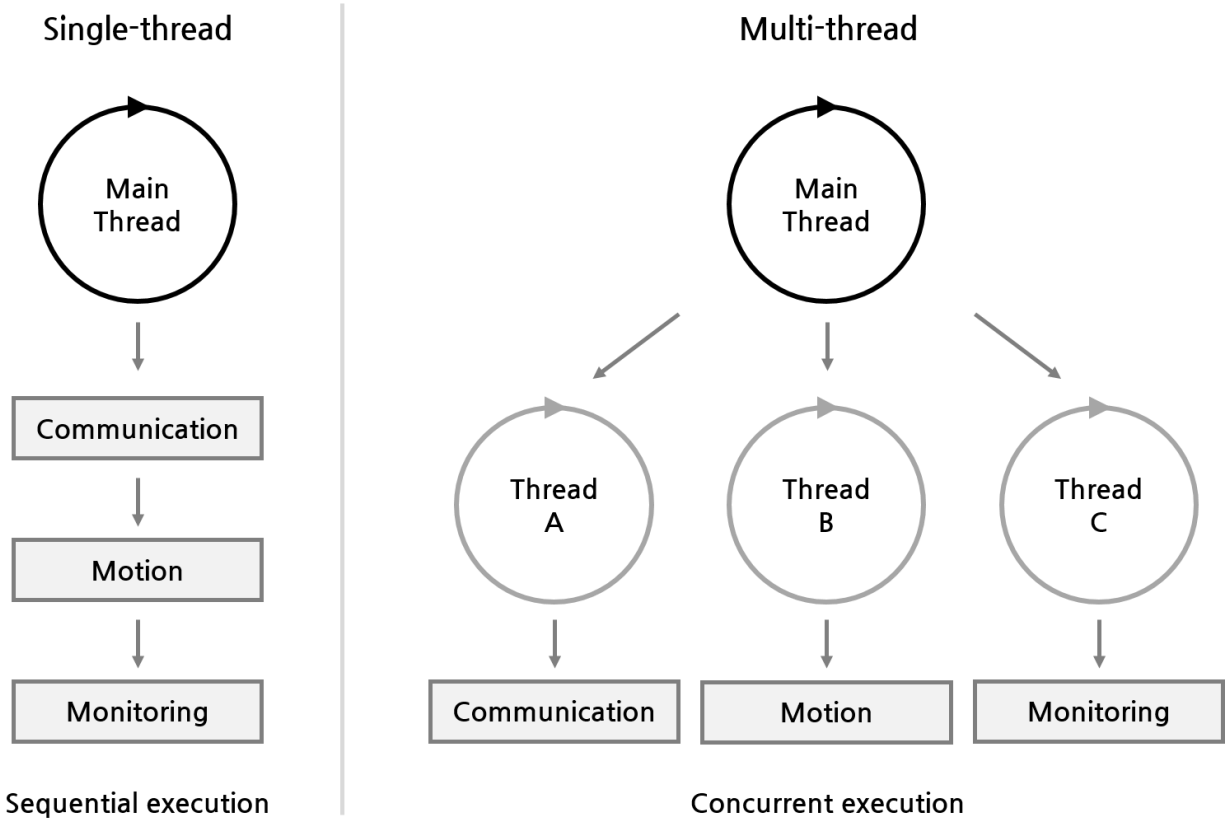


Figure 1-5 单线程和多线程的差异

System模式或Robot模式中可以追加分配处主线程以外最多3个线程同时工作，最多可以使用供4个线程.

一般将单独作业制作成另外的程序，根据特定条件选择主程序分配线程，作业完成后关闭线程.

	System模式	Robot模式
TH1	SYSTEM THREAD 1(主线程)	ROBOT THREAD 1(主线程)
TH2	SYSTEM THREAD 2	ROBOT THREAD 2
TH3	SYSTEM THREAD 3	ROBOT THREAD 3
TH4	SYSTEM THREAD 4	ROBOT THREAD 4

Figure 1-6 System及Robot模式线程结构

多线程功能可以通过执行窗口监控功能确认，使用多线程功能的函数及详细说明请参考 第 4章 9.Multithreading章节.

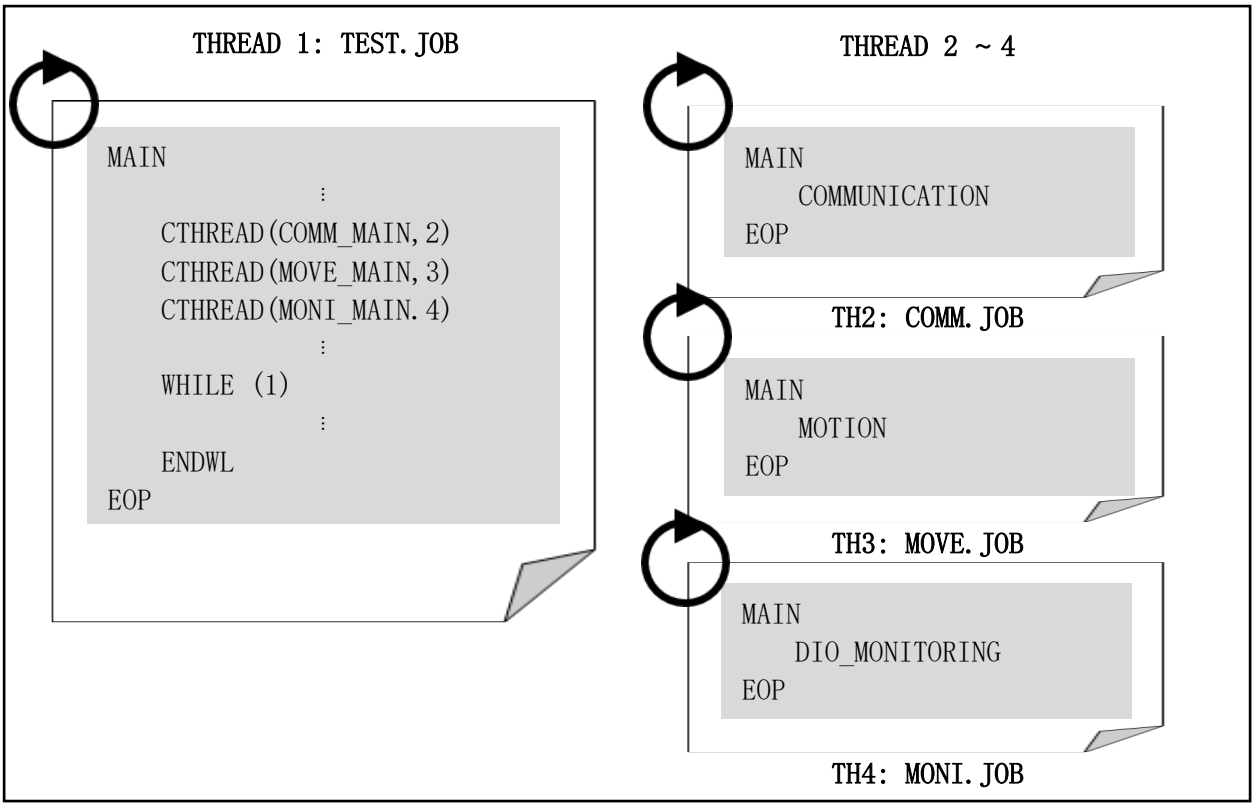


Figure 1-7 多线程执行例子

1) 目前，Robostar对于System模式的主JOB程序仅限于使用Digital I/O专用 BGTD.JOB或现场总线专用BGTF.JOB.
2) System或Robot模式相互独立，因此System模式的主JOB程序（BGTD.JOB, BGTF.JOB）无法在Robot模式使用.

第2章 语言组成因素及语法

1. 语言组成因素

Robostar Robot语言 (RRL, Robostar Robot Language)由以下因素组成.

- 标识符(Identifier): 可以识别组成因素的名称
- 变量(Variable): 由文字和数字组成，临时保存数据的空间
- 常数(Constant): 程序执行过程中无法变更的值
- 运算符(Operator): 执行特定数学或伦理运算的指令符号
- 表达式(Expression): 为获取固有值使用的组成因素的组合
- 命令(Command): 为执行特定功能内置的命令

2. 名称

命令, 变量, 函数, 标签及程序名称需要遵守以下规则.

- 名称只允许以文字开始，只能使用英文大写字母.
- 名称中可以使用文字、数字以及下划线.
- 不可以使用作为运算符使用的文字.

3. 变量

变量为程序中的数据创造保存的空间. 变量包含全局变量、局部变量以及系统变量. 系统变量可用于所有程序, 全局变量分为可以在所有程序中使用的全局变量和只能被从属程序内使用的全局变量. 局部变量仅在声明变量的相应程序块内有效.

3.1 全局变量

全局变量包括整数型及是属性、joint坐标系位置型变量. 全局变量为英文字母开头，加上整数的形式. 全局变量名称由系统定义，因此可以无声明使用. 全局变量可以使用以下类型.

项目	所有程序可以通用的全局变量
I	Integer type (4 bytes, range: -2,147,483,648 ~ 2,147,438,647) 可使用个数: 1,000个 (0 ~ 999)
	Examples) I0, I[1]
F	Real type (4 bytes, range: 3.402,823E-38 ~ 3.402,823E+38) 可使用个数: 1,000个 (0 ~ 999)
	Examples) F100, F[101]
GP	Global joint position type, 可使用个数: 2,000个 (0 ~ 1,999)
	Examples) GP0, GP[1]

项目	从属于程序的全局变量
P	Program joint position type (程序中无法修改值) 可使用个数: 1,000个 (0 ~ 999)
	Examples) P0, P[1]

上述变量中 P 变量拥有程序中无法修改的特征.

全局变量指定后，变量编号可以使用表达式指定. 这种指定方式成为间接参照. 如果执行间接参照，则变量编号由 “[”, “]” 符号包住.

<Examples>

```
I0 = I[2*3]                 // I0号变量中带入I6变量
F1 = F[I12]                 // F1中带入I12号F变量
GP1 = GP[I5+2]              // GP1中带入I5号和2之和号码的GP变量
JMOV P[0]                   // 从当前位置JMOV移动到P0
```

3.2 局部变量

局部变量包括整数型、实数型、字符串、Joint坐标系位置型、Base坐标系位置型、用户坐标系位置型变量. 局部变量需要声明变量后使用，其名称不会由系统定义，因此一定要预先声明. 但, 无法声明的同时初始化. 参照局部变量时，需要提前给局部变量（变量初始化）赋值，否则执行发生错误. 类型声明命令及方法请参照第4章 2.变量 章节.

<Examples>

```
INT AA                          // 变量声明
AA = 0                          // 变量初始化
```

如下表使用 **“$”** 符号声明变量后, 也可以作为全局变量使用. 即, 相应程序内均可使用, 且如果利用INCLUDE语句参照声明变量的程序，在其它程序也可以使用.

<Examples>

```
INT $AA                         // 整数型全局变量声明
REAL $BB                        // 实数型全局变量声明

MAIN                            // 主函数声明
    $AA = 10                    // 整数型全局变量初始化
    $BB = 12.4                  // 实数型全局变量初始化
    INIT()                      // 调用函数
EOP                             // 主函数结束

POS $ACUR                       // Joint坐标系位置型全局变量声明
FUNC VOID INIT()                // 无返回值的 (VOID) 函数定义
    $ACUR = <0,0,0,0,0,0>       // Joint坐标系位置型全局变量初始化
END                             // 函数结束
```

但, 如果将局部变量作为全局变量使用时，需要在主函数或函数外声明变量，变量的初始化需要在函数内部完成.

3.3 系统变量

系统变量用于确认系统状态. 变量名称由系统指定，因此不需要声明变量. 详细说明及使用方法请参靠第 4章 2.11 TMR ~ 2.14 HERE_REF章节.

项目	内容
TMR	系统中预声明的Timer变量，指定计数器值或读取当前计数值时使用的变量
MVR	系统中预声明的变量，返回整体移动距离中占的移动距离比例.
HERE	系统中预声明的变量，当前位置返回为Joint角度值.
HERE_REF	系统中预声明的变量，当前参照位置返回为Joint角度值

4. 函数

函数用于利用预先决定的动作方式获得传递因子的结果. 根据函数类型也有其返回值或传递因子不存在的情况. 用户可以自定义函数使用，此时可以使用所有数据类型的传递因子及返回值. 无返回值的函数声明时使用VOID.

5. 无条件转移

无条件转移表示程序内特定位置的文章. 指定转移位置的名称后，通过命令可以移动到相应位置. 同一个文件内无法使用相同名称的无条件转移.

6. 程序

程序内可以调用其它程序名使用.

<Examples>

```
MAIN                                    // 主函数开始
    JCALL MOTION                        // 调用MOTION.JOB 执行
EOP                                     // 主函数结束
```

7. 数据类型及转换

7.1 数据类型

用户声明并可以使用的数据类型如下.

变量类型		大小及范围	
		说明	可使用个数
数字	INT	4 bytes, -2,147,483,648 ~ 2,147,438,647	
		Integer type	内存范围内
	REAL	4 bytes, 3.402,823E-38 ~ 3.402,823E+38	
		Real type	内存范围内
字符串	DEFSTR	100字以内	
		Integer type	1,000个
位置型	POS	48 bytes (J1, J2, J3, J4, J5, J6, E1, E2)	
		Joint position type	内存范围内
	XPOS	48 bytes (X, Y, Z, RX, RY, RZ, E1, E2, ARM, T)	
		Base position type	内存范围内
	UPOS	48 bytes (Xu, Yu, Zu, RX, RY, RZ, E1, E2, ARM, T, U)	
		User position type	内存范围内

用户使用局部变量时可声明的个数与固定的内存大小相关. 如下表，使用局部变量时可以声明1Mbytes 数字, 位置型变量, 1,000个字符串变量. 使用“$” 符号作为全局变量使用时，使用同样大小的另外一个内存区域. 因此，最多可以使用2Mbyes的数字, 位置型变量, 2,000个字符串变量.

项目	局部变量	全局变量	合计
数字, 位置型变量	1Mbytes	1Mbytes	2Mbytes
字符串变量	1,000个 (100位以下)	1,000个 (100位以下)	2,000个

1) 超出内存大小时发生Alarm.

对于位置型类型，在变量名上使用 “.” 符号代表各参数值. 可代表的位置数据个数与轴个数和自由度 (Degree of freedom, 以下称为DOF) 值不同. 与使用常数初始化不同，可以读写下表中所有值.

分类	例子
J1轴 (或 X轴)	<变量名>.1
J2轴 (或 Y轴)	<变量名>.2
J3轴 (或 Z轴)	<变量名>.3
J4轴 (或 RX轴)	<变量名>.4
J5轴 (或 RY轴)	<变量名>.5
J6轴 (或 RZ轴)	<变量名>.6
J7轴 (E1轴)	<变量名>.7
J8轴 (E2轴)	<变量名>.8
Robot姿势	<变量名>.ARM
Tool 编号	<变量名>.T
User 编号	<变量名>.U
是否完成变量初始化	<变量名>.USED

如使用6轴垂直多关节Robot，1 ~ 6轴代表Robot的各Joint轴. 7 ~ 8轴根据Robot安装环境需要使用附加轴时使用.

Robot的姿势如下输入. 输入0 值表示以当前Robot的JOINT 值为基准设置姿势. 各姿势相关详细说明请参 操作及应用说明书 (N2-OM-C□□).

Value	Shoulder	Elbow	Wrist
000	NOT DEFINED	NOT DEFINED	NOT DEFINED
001	NOT DEFINED	NOT DEFINED	FLIP
002	NOT DEFINED	NOT DEFINED	NON FLIP
010	NOT DEFINED	DOWN	NOT DEFINED
011	NOT DEFINED	DOWN	FLIP
012	NOT DEFINED	DOWN	NON FLIP
020	NOT DEFINED	UP	NOT DEFINED
021	NOT DEFINED	UP	FLIP
022	NOT DEFINED	UP	NON FLIP
100	REAR	NOT DEFINED	NOT DEFINED
101	REAR	NOT DEFINED	FLIP
102	REAR	NOT DEFINED	NON FLIP
110	REAR	DOWN	NOT DEFINED
111	REAR	DOWN	FLIP
112	REAR	DOWN	NON FLIP
120	REAR	UP	NOT DEFINED
121	REAR	UP	FLIP
122	REAR	UP	NON FLIP
200	FRONT	NOT DEFINED	NOT DEFINED
201	FRONT	NOT DEFINED	FLIP
202	FRONT	NOT DEFINED	NON FLIP
210	FRONT	DOWN	NOT DEFINED
211	FRONT	DOWN	FLIP
212	FRONT	DOWN	NON FLIP
220	FRONT	UP	NOT DEFINED
221	FRONT	UP	FLIP
222	FRONT	UP	NON FLIP

Tool 及 User 编号分别可以输入0 ~ 15供16个值.

最后输入变量是否初始化 (0: 未初始化, 1: 已初始化). 但, 用常数初始化变量时, 变量是否初始化的值自动设置为1. 变量初始化情况为0的位置型变量用于移动命令时发生Alarm.

1) 对各项初始化时，请务必向USED 项输入初始化完成 (1) 值告知初始化完成

7.2 数据类型转换

根据以下规则相互转换，如果无法满足规则则发生Alarm.

1) 整数和实数值

- 赋给其它类型变量时, 数值转换成满足变量类型的值.
- 实数转换为整数时四舍五入并赋值.

2) 字母和数值

转换前 \ 转换后	字母	整数
字母		ASC
整数	BINS, CHR	

- 例) 字母数据使用ASC 函数转换为整数.

3) 位置型类型

转换前 \ 转换后	POS	XPOS	UPOS
POS		JTOB	JTOU
XPOS	BTOJ		BTOU
UPOS	UTOJ	UTOB	

- 例) POS 数据利用JTOB函数转换为XPOS 数据.

8. 常数

常数拥有固定值且如下表分类.

分类		说明
整数	Binary	仅使用0 或1. 前面加0B表示
		Example) 0B11
	Decimal	使用0 ~ 9 范围的数字表示
		Example) 100
	Hexadecimal	使用0 ~ 9, A ~ F 表示, 前面加 0H
		Example) 0HFF
实数		利用“.” 符号表示小数点以下
		Example) 2.3456
字符串		使用双引号表示字符串
		Example) “String statement”
位置型	POS	使用“<” 及 “>” 符号表示
		Example) <100, 100, 100, 100, 100, 100>
	XPOS	输入ARM (姿势), T(Tool 编号)
		Example) <100, 100, 100, -30, 30, 60, ARM = 0, T = 2>
	UPOS	输入ARM (姿势), T(Tool 编号), U(User 编号)
		Example) <0, 0, 0, -30, 30, 60, ARM = 0, T = 1, U = 3>

9. 表达式和运算符

表达式用于返回值. 将多个独立的表达式和运算符相连接形成表达式.

9.1 运算符优先级

运算符	说明	优先级
(,)	括号	High
~, !	比特及逻辑NOT	
*, /, %	乘法, 除法, 取余	
+, -	加法, 减法	
<<, >>	左Shift, 右Shift	
<, <=, >, >=	小于, 小于等于, 大于, 大于等于	
==, !=	等于, 不等于	
&	位AND	
^	位 XOR	
|	位 OR	
&&	逻辑 AND	
||	逻辑 OR	
=	代入	Low

9.2 代入运算符

使用“=” 符号向左侧变量代入值或表达式的结果值. 根据变量类型可能发生类型转换.

<Examples>

```
INT A, B, C, D
IN0 = 1                                    // IN 命令
A = 100
B = I5
C = F1
D = A*(B+C)
```

9.3 计算运算符

求加法(+) 减法(-), 乘法(*), 除法(/), 取余 (%)的值. 求取余值遵循以下表的规则.

Divisor / Dividend	(+)	0	(-)
(+)	(+)	Error	(+)
0	0	Error	0
(-)	(-)	Error	(+)

1) 除以0或取余时发生 “[1236] Interpreter error” Alarm.
2) 使用计算运算符可能导致溢出. 虽然不发生A拉拉人么，但请留意最大可用范围值.

9.4 关系运算符

关系运算符用于比较两个数值，其运算结果为逻辑值 (1: true, 0: false). 主要用于流向控制命令语句的条件表达式中.

关系运算符	说明
<	小于
>	大于
<=	小于等于
>=	大于等于
==	等于
!=	不等于

9.5 逻辑及位运算符

逻辑运算符用于连接一个或多个条件表达式. 位运算符以比特为单位进行逻辑处理. 各符号的含义如下表所示.

	逻辑	位
Negation	!	~
Product	&&	&
Addition	||	|
Exclusive addition		^

9.6 数组

Robostar Robot 语言(RRL, Robostar Robot Language)支持1级数组. 数组是相同类型的多个变量的集合. 组成数组的各值称为数组因素 (Element), 代表数组中位置的值称为索引(Index). 索引始终从0开始，且仅可以使用包含0的整数.

<Syntax>
<变量类型> <数组名>’ [‘<数组长度>’]’
<Examples>
INT AA[10] AA[0] = 10

1) 无法使用C语言中指针.
2) 无法声明数组的同时初始化.
3) 请在不超过整体内存大小 (1MB)的范围使用.

9.7 字符串应用

两个字符串可以使用加法符号 “+”相加. 加法符号“+”可以用于常数或数组.

<Syntax>
<字符串变量> = <字符串> + <字符串> <字符串变量> = <字符串变量> + <字符串> <字符串变量> = <字符串变量> + <字符串变量>
<Examples>
DEFSTR STR1, STR2, STR3 STR1 = “TEST “ + “STATEMENT ” STR2 = STR1 + “COMPLETE” STR3 = STR1 + STR2

1) 请注意相加后的字符串不超过100位.

9.8 位置型应用

位置型数据可以进行如下作业.

<Syntax>

<位置型变量> = <位置型常数或变量> + <位置型常数或变量>
<位置型变量> = <位置型常数或变量> - <位置型常数或变量>
<位置型变量> = <位置型常数或变量> * <数字变量或常数>
<位置型变量> = <位置型常数或变量> / <数字变量或常数>

<Examples>

```
POS RES_P, AP, BP
REAL A, B

A = 10.1
B = -7.2
AP = <50, 50, 50, 50, 50, 50>
BP = <30, 30, 30, 30, 30, 30>

RES_P = <100, 100, 100, 100, 100, 100> + <100, 100, 100, 100, 100, 100>
PRINT(0,RES_P)
// <200.000,200.000,200.000,200.000,200.000,200.000,0.000,0.000>

RES_P = AP + <100, 100, 100, 100, 100, 100>
PRINT(1,RES_P)
// <150.000,150.000,150.000,150.000,150.000,150.000,0.000,0.000>

RES_P = <100, 100, 100, 100, 100, 100> + BP
PRINT(2,RES_P)
// <130.000,130.000,130.000,130.000,130.000,130.000,0.000,0.000>

RES_P = AP + BP
PRINT(3,RES_P)
// <80.000,80.000,80.000,80.000,80.000,80.000,0.000,0.000>

RES_P = <100, 100, 100, 100, 100, 100> * A
PRINT(4,RES_P)
// <1010.000,1010.000,1010.000,1010.000,1010.000,1010.000,0.000,0.000>

RES_P = <100, 100, 100, 100, 100, 100> / B
PRINT(5,RES_P)
// <-13.889,-13.889,-13.889,-13.889,-13.889,-13.889,-0.000,-0.000>

RES_P = AP * A
PRINT(6,RES_P)
// <505.000,505.000,505.000,505.000,505.000,505.000,0.000,0.000>

RES_P = BP / B
PRINT(7,RES_P)
// <-4.167,-4.167,-4.167,-4.167,-4.167,-4.167,-0.000,-0.000>
```

1) 无法用于相互不同的位置型变量. 请务必使用相同类型的位置型变量.
2) 如果想变为其它位置型变量，请参考第 4章 10.3 JTOB ~ 10.8 UTOB 章节的转换函数.

第3章 命令列表

1. 变量及常数

1.1 一般变量

分类		说明 格式	使用例子	Ref.
数字	INT	声明整数型变量 (4 bytes) INT <变量名>	INT AA AA = 0	2.1
	REAL	声明实数型变量 (4 bytes) REAL <变量名>	REAL BB BB = 0.0	2.2
DEFSTR		声明字符串变量 DEFSTR <变量名>	DEFSTR SS SS = “STRING”	2.3
POS		声明Joint坐标系位置型变量 POS <变量名>	POS AP AP = <10,10,10,10,10,10>	2.4
XPOS		声明Base坐标系位置型变量 XPOS <变量名>	XPOS BP BP = <0,0,0,0,0,0,ARM = 0,T = 0>	2.5
UPOS		声明用户坐标系位置型变量 UPOS <变量名>	UPOS CP CP = <0,0,0,0,0,0,ARM = 0,T = 0,USED = 0>	2.6
P		预定义的Joint坐标系位置型变量，保存在程序文件，在相应程序内作为全局变量使用 **※ JOB 文件内保存的程序Joint位置型变量 P无法在程序内直接编辑** P<编号> or P[编号]	JMOV P0 JMOV P[1]	2.7
I		预定义的全局整数变量 I<编号> or I[编号]	I0 = 100 I[1] = 50	2.8
F		预定义的全局实数变量 F<编号> or F[编号]	F0 = 3.14 F[1] = 10.54	2.9
GP		预定义的全局Joint坐标系位置型变量 GP<编号> or GP[编号]	GP0 = <0,0,0,0,0,0> GP[1] = <10,10,10,10,10,10>	2.10

1) 变量名不可以以数字开头.
2) 变量名由大写英文字母和数字组成.
3) 位置型变量在变量名后使用 ‘.’ 指定轴编号或偏移读写值.
4) 变量相关详细说明请参考第2章 3. 变量 及 第 2章 7. 数据类型及转换, 第 4章 2. 变量 章节.
5) 变量名前使用 $符号可以作为全局变量使用.

1.2 系统变量

分类	说明 格式	使用例子	Ref.
TMR(0) **TMR(1)**	Timer函数, 输入整数值的瞬间开始赋值，以10ms 为间隔加一 <编号>: 选择Timer编号 (0 或 1) TMR(<编号>) = <值> <整数型变量> = TMR(<编号>)	INT AA TMR(0) = -50 TMR(1) = 0 AA = TMR(0)	2.11

MVR	确认移动距离的百分比
	<Robot编号>: 输入编号指定Robot, 不输入时使用JOB程序中保存的信息 (1 ~ 3)
	
	<实数型变量> = MVR[(<Robot编号>)]
	MVR
		VALUE1 = MVR
		VALUE2 = MVR(2)
		IF MVR <50 THEN
		ENDIF
			2.12

HERE	当前位置返回为Joint值 <Robot编号>: 输入编号指定Robot, 不输入时使用JOB程序中保存的信息 (1 ~ 3) <位置型变量> = HERE[(<Robot编号>)]	POS AP, BP AP = HERE BP = HERE(2)	2.13
HERE_REF	当前参照位置返回为joint值 <Robot编号>: 输入编号指定Robot, 不输入时使用JOB程序中保存的信息 (1 ~ 3) <位置型变量> = HERE_REF[(<Robot编号>)]	POS AP, BP AP = HERE_REF BP = HERE_REF(2)	2.14

1.3 常数

分类	说明 格式	使用例子	Ref.
数字常数	表示整数, 实数, 2进制整数, 16进制整数 [OH/OB]<数字> 数字前没有OH 或OB表示10进制数	INT AA AA = OH12 // 16进制数 AA = OB11 // 2进制数	-
字符串常数	表示字符串, 最大长度100以内 “<字符串>”	DEFSTR STR STR = “STRING CONSTANT”	-
位置型常数	表示位置型变量各别值 使用XPOS 时 ARM(姿势), T(Tool编号) 值, 使用UPOS 时 ARM(姿势), T(Tool编号), U(User 编号) 值需要追加输入值 需要指定<是否设置变量值>时使用 USED 指定值 **※ 利用位置型常数初始化位置型变量值时，可以 省略是否设置变量值 (Default: 1, 0: 不设定值, 1: 设定值)** <位置值(DOF), [<Arm form>], [<Tool 编号>], [<User 编号>], [<是否设置变量值>]>	XPOS XP XP = <0,0,0,0,0,0,ARM = 0,T = 1,USED = 1> UPOS UP UP = <1,2,3,4,5,6,ARM = 0,T = 1,USED = 1>	-

2. 预处理机及注释

分类	说明 格式	使用例子	Ref.
INCLUDE	将特定程序文件引入当前位置，作为一个程序使用 INCLUDE “<JOB 文件名>”	INCLUDE “TEST”	3.1
/*, */	多行注释 /*<内容>*/	/* JMOV GP0 DLAY 1000 */	3.2
//	一行注释 //<内容>	// JMOV GP0 DLAY 1000	

3. 运算符

分类	说明 格式	使用例子	Ref.
=	带入运算符 <变量> = <变量> or <表达式>	B = A	4.1
+, -, *, /, %	计算运算符 (加, 减, 乘, 除, modulus) <变量> = <A项><计算运算符><B项>	C = A+B	4.2
>, <, >=, <=, !=, ==	关系运算符 (超出, 未满, 以上, 以下, 不等, 相等) [<数字变量> =]<A项><关系运算符><B项>	IF A > B THEN ENDIF	4.3

&&, ||, !	逻辑运算符
	(AND, OR, Negation)
	[<数字变量> =]<A项><逻辑运算符><B项>
		C = !A
		IF (A == 1) && (B == 0) THEN
		ENDIF
			4.4

&, I, ~,
^, <<, >>
	位运算符
	(BAND, BOR, Complement,
	BXOR, left shift, right shift)
	[<数字变量> =]<A项><位运算符><B项>
		C = A << 0H01
		IF A & B THEN
		ENDIF
			4.5

( )	括号 (<表达式>)	D = (A+B)*C	4.6

4. 程序控制

分类	说明 格式	使用例子	Ref.
MAIN … EOP	主程序开始及结束 MAIN … EOP	MAIN JMOV P1 JMOV P2 EOP	5.1
FUNC … END	函数声明及结束， 返回类型(包括VOID) 及参数声明. 可以返回特定值 FUNC <返回类型> <函数名>(<参数>) [RET <变量>] END	FUNC VOID TEST() VEL 200 JMOV P0 END	5.2
RET	返回函数种类及值 RET <返回值>	FUNC INT TEST() RET 1 END	5.3
FOR … TO … [BY …] NEXT	按照指定次数生成循环. BY <增量>省略时默认为‘1’. FOR <初始化语句> TO <结束条件> [BY <增量>] … NEXT	… FOR AA = 1 TO 5 [BY 2] BB = BB+AA NEXT …	5.4
WHILE … ENDWL	执行条件重复 WHILE <条件表达式> … ENDWL	… WHILE IN0 == 1 JMOV P1 ENDWL …	5.5

IF … THEN …
[ELSE IF … THEN]
[ELSE …] ENDIF
	执行条件判断
	IF <条件表达式> THEN
	…
	[ELSEIF <条件表达式> THEN]
	…
	[ELSE]
	…
	ENDIF
		…
		IF IN3 == 1 THEN
		GOTO A0
		ELSE
		JMOV P1
		ENDIF
		LABL A0
		…
			5.6

LABL … GOTO	指定无条件转移位置并移动 LABL <无条件转移名> … GOTO <无条件转移名>	… LABL A1 JMOV P1 JMOV P2 GOTO A1 …	5.7

CONTINUE	回到重复语句的首部
	WHILE <条件表达式>
	…
	CONTINUE
	ENDWL
		WHILE 1
		IF IN(1) == 1 THEN
		CONTINUE
		ENDIF
		JMOV P0
		ENDWL
			5.8

分类	说明 格式	使用例子	Ref.

BREAK	停止重复语句并退出
	WHILE <条件表达式>
	…
	BREAK
	ENDWL
		WHILE 1
		IF IN(1) == 1 THEN
		BREAK
		ENDIF
		JMOV P0
		ENDWL
			5.9

JCALL	调用JOB 文件并立即执行 JCALL <JOB 文件名>	JCALL INC	5.10

STOP	停止Robot动作
	STOP
		IF IN0 == 1 THEN
		STOP
		ENDIF
			5.11

EXIT	退出JOB 程序
	EXIT
		IF IN0 == 1 THEN
		EXIT
		ENDIF
			5.12

5. Robot动作

分类	说明 格式	使用例子	Ref.
SVON	伺服电源ON, <Robot编号>: 输入编号指定Robot，未输入时使用JOB程序中保存的信息 (1 ~ 3) SVON[(<Robot编号>)]	SVON SVON(2)	6.1
SVOF	伺服电源 OFF, <Robot编号>: 输入编号指定Robot，未输入时使用JOB程序中保存的信息 (1 ~ 3) SVOF[(<Robot编号>)]	SVOF SVOF(2)	6.2
JMOV	从当前位置移动到目标点PTP JMOV <Point变量>[, <VEL = 值>, <ACC = 值>, <DEC = 值>, <FOS = 值>, <T = 值>, <U = 值>]	JMOV P1 JMOV P2, VEL = 90, ACC = 20, DEC = 20	6.3
LMOV	从当前位置线性插补移动到目标点 LMOV <Point变量>[, <VEL = 值>, <ACC = 值>, <DEC = 值>, <FOS = 值>, <T = 值>, <U = 值>]	LMOV P1 LMOV P2, VEL = 90, ACC = 50, DEC = 50	6.4
CMOV	从当前位置画出连接经由点 1, 2的圆并移动 CMOV <经由点1> <经由点2>[, <VEL = 值>, <ACC = 值>, <DEC = 值>, <T = 值>, <U = 值>]	CMOV P1 P2 CMOV P1 P2, VEL = 100	6.5
AMOV	从当前位置画出连接经由点1和目标点的圆并移动 AMOV <经由点1> <目标点> [, <VEL = 值>, <ACC = 值>, <DEC = 值>, <T = 值>, <U = 值>]	AMOV P1 P2, VEL = 100	6.6

6. Robot动作条件

分类	说明 格式	使用例子	Ref.
WITH … ENDWT	Robot动作过程中同时处理程序块内的执行语句 WITH … ENDWT	WITH JMOV P1 WHILE MVR < 80 ENDWL OUT0 = 1 ENDWT	7.1
VEL	设置轴移动速度的千分比 (%) 轴移动速度 = 初始速度 X 额定速度 X 千分率值 X 0.001 **※ 额定速度及初始速度在参数中设置** VEL <千分率值(%)>	VEL 200 JMOV P1	7.2
ACC	设置加速时间的百分比 (%) 加速时间 = 额定加速时间X 百分比值 X 0.01 **※ 额定减速时间在参数中设置** ACC <百分比值(%)>	ACC 50 JMOV P1	7.3
DEC	设置减速时间的百分比 (%) 减速时间 = 额定减速时间 X 百分比值 X 0.01 **※ 额定减速时间在参数中设置** DEC <百分比值(%)>	DEC 200 JMOV P1	7.4
FOS	轴线段到达目标点前，参考到达下一个目标点为止的距离比 (%)变更轨道 距离比为整体移动距离的百分比 (%) 可用于JMOV, LMOV FOS <距离比(%)>	FOS 5 JMOV P1 FOS 0 // 放行	7.5
DLAY	延迟时间设置，单位为1ms DLAY <延迟时间>	JMOV P1 DLAY 1000	7.6
OFFS	目标点移动指定的距离只可以输入位置型变量（不可以直接代入常数） OFFS <偏差值>	OFFS P1 JMOV P2	7.7
OFFSCLR	清除设置的OFFS 值 OFFSCLR	OFFSCLR	7.8
LIMT	限制各轴的移动范围只可以输入位置型变量(不可以直接代入常数) LIMT <(-)移动范围, (+)移动范围>	LIMT P0 P1	7.9
TRQ	设置扭矩限定值，超出设定值时发出Alarm TRQ <Robot编号> <轴编号> <扭矩限定值>	TRQ 1 2 100	7.10
GAIN	切换Gain组 GAIN <组编号>	GAIN 1 JMOV P0	7.11
IPE	设置到达目标位置的允许时间最大值， 单位 1 ms IPE <允许时间>	IPE 100	7.12

7. 输入输出控制

7.1 一般Digital输入输出

分类	说明 格式	使用例子	Ref.
IN	读取以1比特为单位指定的比特编号值 <比特编号>: 0 ~ 19 (连接扩展Board可以最大83) IN <比特编号> 或 IN(<比特编号>) = <0/1> - 直到满足条件为止等待 <整数型变量> = IN<比特编号> - 输入状态值保存到变量	INT AA IN10 = 1 AA = IN(0)	8.1
OUT	以1比特为单位指定的比特编号的输出 ON(= 1), OFF(= 0) <比特编号>: 0 ~ 19 (连接扩展Board可以最大83) <脉冲幅度>: 单位 1ms, 输入时超出有效时间, 则恢复到前一个状态 <脉冲周期>: 单位 1ms, 输入脉冲周期时间, 需要比脉冲幅度大 OUT<比特编号> 或 OUT(<比特编号>) = <0/1> OUT<比特编号> = <0/1>[<脉冲幅度> <脉冲周期>] OUT<比特编号> = <0/1>[<脉冲幅度> <脉冲周期>] - 输出值 <整数型变量> = OUT<比特编号> - 确认当前输出中的数据值	INT AA OUT0 = 1 OUT0 = 1 100 OUT0 = 1 100 200 AA = OUT(0)	8.2
PIN	读取以端口为段位指定的输入端口值 <端口编号>: 0 ~ 1 (连接扩展Board时最大5) <整数型变量> = PIN<端口编号> 或 <整数型变量> = PIN(<端口编号>)	INT AA, BB AA = PIN(0) BB = PIN1	8.3
POUT	输出以端口为段位指定的输入端口值 <端口编号>: 0 ~ 1 (连接扩展Board时最大5) POUT<端口编号> = <输出指定值> 或 POUT(<端口编号>) = <输出指定值> <整数型变量> = POUT<端口编号>	INT AA POUT0 = 0 POUT0 = 20 POUT1 = 0H000F AA = POUT(0)	8.4

1) 使用OUT 或 COUT 命令生成脉冲洗脑或脉冲周期波形时，输入的 <脉冲幅度> 及 <脉冲周期> 值的输入单位为1ms. 但, 2ms 以下的值将被忽略，请输入2倍数的值.

7.2 现场总线输入输出

分类	说明 格式	使用例子	Ref.
CIN	读取以1比特为单位指定的现场总线比特编号值 <整数型变量> = CIN<比特编号> 或 <整数型变量> = CIN(<比特编号>)	INT AA, BB AA = CIN0 BB = CIN(1)	8.5
COUT	以1比特为单位指定的比特编号输出ON(= 1), OFF(= 0) <脉冲幅度>: 单位 1ms, 输入时超过有效时间则恢复到前一个状态 <脉冲周期>: 单位 1ms, 输入脉冲周期时间, 需要始终大于脉冲幅度 COUT<比特编号> = <0/1> 或 COUT(<比特编号>) = <0/1> COUT<比特编号> = <0/1>[<脉冲幅度>] COUT<比特编号> = <0/1>[<脉冲幅度> <脉冲周期>] - 输出值 <整数型变量> = COUT<比特编号> - 确认当前输出中的数据值	INT AA, BB COUT0 = 1 COUT0 = 1 100 COUT0 = 1 100 200 AA = COUT0 BB = COUT(1)	8.6
CBIN	读取字节为单位指定的现场总线字节值 <整数型变量> = CBIN<端口编号> 或<整数型变量> = CBIN(<端口编号>)	INT AA, BB AA = CBIN1 BB = CBIN(0)	8.7
CBOUT	向以字节为单位指定的现场总线字节指定输出值 CBOUT<端口编号> = <输出指定值> 或CBOUT(<端口编号>) = <输出指定值) <整数型变量> = CBOUT<端口编号>	INT AA CBOUT0 = 0HFF AA = CBOUT(1)	8.8
CDIN	以整数为单位读取双单词输入 <整数型变量> = CDIN<双单词编号> 或<整数型变量> = CDIN(<单词编号>)	INT AA, BB AA = CDIN0 BB = CDIN(0)	8.9
CDOUT	以整数为单位输出双单词输出 CDOUT<双单词编号> = <输出指定值> 或CDOUT(<双单词编号>) = <输出指定值) <整数型变量> = CDOUT<双单词编号>	INT AA CDOUT1 = 0HFFFFFFFF AA = CDOUT(0)	8.10

分类	说明 格式	使用例子	Ref.
CWIN	读取以单词为单位指定的现场总线单词编号的值 <整数型变量> = CWIN<单词编号> 或<整数型变量> = CWIN(<单词编号>)	INT AA, BB AA = CWIN0 BB = CWIN(1)	8.11
CWOUT	向以单词为单位指定的现场总线单词编号输出指定值 CWOUT<单词编号> = <输出指定值> 或CWOUT(<单词编号>) = <输出指定值) <整数型变量> = CWOUT<单词编号>	INT AA CWOUT0 = 0HFFFF AA = CWOUT(1)	8.12
CFIN	以实数为单位读取双单词输入 <实数型变量> = CFIN<双单词编号> 或<实数型变量> = CFIN(<单词编号>)	REAL AA, BB AA = CFIN0 BB = CFIN(0)	8.13
CFOUT	以实数为单位输出双单词输出 CFOUT<双单词编号> = <输出指定值> 或 CFOUT(<双单词编号>) = <输出指定值) <实数型变量> = CFOUT<双单词编号>	REAL AA CFOUT1 = 0HFFFFFFFF AA = CFOUT(0)	8.14

7.3 CNET 输入输出

分类	说明 格式	使用例子	Ref.
XIN	保存CNET 设备区域和比特编号，读取输入的比特 <整数型变量> = XIN(<设备区域>, <比特编号>)	INT DATA DATA = XIN(“M”, 0)	8.15
XOUT	指定CNET 设备区域和比特编号，输出比特或读取输出的比特数据值 XOUT(<设备区域>, <比特编号>) = <0/1> 或 <整数型变量> = XOUT(<设备区域>, <比特编号>)	INT DATA XOUT(“M”, 0) = 1 DATA = XOUT(“M”, 0)	8.16
XBIN	指定CNET 设备区域和字节编号，读取输入的字节 <整数型变量> = XBIN(<设备区域>, <字节编号>)	INT DATA DATA = XBIN(“D”, 0)	8.17
XBOUT	指定CNET 设备区域和字节编号，输出字节或读取输出的字节数据值 XBOUT(<设备区域>, <字节编号>) = <输出值>, 或 <整数型变量> = XBOUT(<设备区域>, <字节编号>)	INT DATA XBOUT(“D”, 0) = 0HFF DATA = XBOUT(“D”, 0)	8.18

分类	说明 格式	使用例子	Ref.
XWIN	指定CNET 设备区域和Word编号，读取输入的Word <整数型变量> = XWIN(<设备区域>, <Word编号>)	INT DATA DATA = XWIN(“M”, 0)	8.19
XWOUT	指定CNET 设备区域和Word编号，输出Word或读取输出的Word数据值 XWOUT(<设备区域>, <Word编号>) = <输出值>, 或 <整数型变量> = XWOUT(<设备区域>, <Word编号>)	INT DATA XWOUT(“M”, 0) = 0HFFFF DATA = XWOUT(“M”, 0)	8.20
XDIN	指定CNET 设备区域和Double Word编号，以整数形式读取值 <整数型变量> = XDIN(<设备区域>, <Double Word编号>)	INT DATA DATA = XDIN(“D”, 0)	8.21
XDOUT	指定CNET 设备区域和 Double Word编号输出Double Word或读取输出的Double Word数据值 XDOUT(<设备区域>, <Double Word 编号>) = <输出值>, 或 <整数型变量> = XDOUT(<设备区域>, <Double Word 编号>)	INT DATA XDOUT(“D”, 0) = 0HFFFFFFFF DATA = XDOUT(“D”, 0)	8.22
XFIN	指定CNET 设备区域和Double Word编号，以实数形式读取输入值 <实数型变量> = XFIN(<设备区域>, <Double Word编号>)	REAL DATA DATA = XFIN(“M”, 0)	8.23
XFOUT	指定 CNET 设备区域和 Double Word 编号，输出实数值或读取输出的实数数据值 XFOUT(<设备区域>, <Double Word 编号>) = <实数值>, 或 <实数型变量> = XFOUT(<设备区域>, <Double Word 编号>)	REAL DATA XFOUT(“M”, 0) = 12.45 DATA = XFOUT(“M”, 0)	8.24

8. Multithreading 控制

分类	说明 格式	使用例子	Ref.
CTHREAD	生成thread <thread 编号>: 只能使用2 ~ 4 <thread Alarm停止>: 发生Alarm时, 停止Thread的功能 (1: ON, 0: OFF) [<整数型变量> =] CTHREAD(<函数名>, <Thread 编号>, [<Thread Alarm停止>])	INCLUDE “AA” MAIN INT T_RET T_RET = CTHREAD(AA_MAIN, 2) EOP	9.1
ETHREAD	结束thread <thread 编号>: 只能使用2 ~ 4 [<整数型变量> =] ETHREAD(<thread 编号>)	ETHREAD(2)	9.2
TH_STATE	读取thread 状态信息 <执行模式>: 输入System模式 (1) 或Robot模式 (0) <Thread 编号>: 只能使用1 ~ 4 - 返回值: 0 (TH_ERROR): Thread 编号错误 1 (TH_IDLE): 可以使用thread 2 (TH_STOP): thread停止 3 (TH_RUN): thread执行中 <整数型变量> = TH_STATE(<执行模式>, <Thread 编号>)	INT T_STATE T_STATE = TH_STATE(1,1)	9.3
TH_SUSPEND	停止执行中的 thread <Thread 编号>: 只能使用2 ~ 4 [<整数型变量> =] TH_SUSPEND(<Thread 编号>)	TH_SUSPEND(2)	9.4
TH_RESUME	执行停止中的thread <Thread 编号>: 只能使用2 ~ 4 [<整数型变量> =] TH_RESUME(<Thread 编号>)	TH_RESUME(2)	9.5

9. 函数

9.1 坐标转换

分类	说明 格式	使用例子	Ref.
TOOL	选择要使用的Tool编号(0 ~ 15) TOOL <编号>	TOOL 3 LMOV P1	10.1
USER	选择使用的用户坐标系编号(0 ~ 15) USER <编号>	USER 10 LMOV P1	10.2
JTOB	Joint坐标系位置变量变为Base坐标系位置变量值. <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <Base坐标系位置变量> = JTOB([<Robot编号>,] <Joint坐标系位置变量>)	POS AP XPOS XP1, XP2 AP = <0,0,0,0,0,0> AP.T = 2 XP1 = JTOB(AP) XP2 = JTOB(2,AP)	10.3
BTOJ	Base坐标系位置变量变为Joint坐标系位置变量值 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <Joint坐标系位置变量> = BTOJ([<Robot编号>,] <Base坐标系位置变量>)	POS AP1, AP2 XPOS XP XP = <100,100,200,30,5,45,ARM = 0,T = 1> AP1 = BTOJ(XP) AP2 = BTOJ(2,XP)	10.4
JTOU	Joint坐标系位置变量变为用户坐标系位置变量值 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <用户坐标系位置变量> = JTOU([<Robot编号>,] <Joint坐标系位置变量>)	POS AP UPOS UP1, UP2 AP = <0,0,0,0,0,0> AP.U = 1 UP1 = JTOU(AP) UP2 = JTOU(2, AP)	10.5
UTOJ	用户坐标系位置变量值变为Joint位置变量值 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <Joint坐标系位置变量> = UTOJ([<Robot编号>,] <用户坐标系位置变量>)	POS AP1, AP2 UPOS UP UP = <10,10,10,3,6,0,ARM = 0,T = 0,U = 1> AP1 = UTOJ(UP) AP2 = UTOJ(2, UP)	10.6
BTOU	Base坐标系位置变量值变为用户坐标系位置变量 值 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <用户坐标系位置变量> = BTOU([<Robot编号>,] <Base坐标系位置变量>)	XPOS XP UPOS UP1, UP2 XP = <100,100,200,30,5,45,ARM = 0,T = 1> XP.U = 2 UP1 = BTOU(XP) UP2 = BTOU(2, XP)	10.7

分类	说明 格式	使用例子	Ref.
UTOB	用户坐标系位置变量值变为Base坐标系位置变量 值 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <Base坐标系位置变量> = UTOB([<Robot编号>,]<用户坐标系位置变量>)	XPOS XP1, XP2 UPOS UP UP = <10,10,10,0,0,0,ARM = 0,T = 0,U = 1> XP1 = UTOB(UP) XP2 = UTOB(2, UP)	10.8
RELBASE	以Base坐标系为基准，从基准位置移动代入移动量的相对移动位置值的函数 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <基准位置>: Base坐标系位置变量 <移动量>: 所有坐标系位置变量 <Base坐标系位置型变量> = RELBASE([<Robot编号>,]<基准位置>, <移动量>)	POS AP, DATA XPOS XP, RP1, RP2 DATA = <100,0,0,0,0,50> AP = HERE AP.T = 0 XP = JTOB(AP) RP1 = RELBASE(XP, DATA) RP2 = RELBASE(1,XP, DATA)	10.9
RELTOOL	以Tool坐标系为基准，从基准位置移动代入移动量的相对移动位置值的函数 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <基准位置>: Base坐标系位置变量 <移动量>: 所有坐标系位置变量 <Base坐标系位置型变量> = RELTOOL([<Robot编号>,]<基准位置>, <移动量>)	POS AP, DATA XPOS XP, RP1, RP2 DATA = <100,0,0,0,0,50> AP = HERE AP.T = 1 XP = JTOB(AP) RP1 = RELTOOL(XP, DATA) RP2 = RELTOOL(1,XP, DATA)	10.10
RELUSER	以用户坐标系为基准，从基准位置移动代入移动量的相对移动位置值的函数 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <基准位置>: 用户坐标系位置变量 <移动量>: 所有坐标系位置变量 <用户坐标系位置型变量> = RELUSER([<Robot编号>,]<基准位置>, <移动量>)	POS AP, DATA UPOS UP, RP1, RP2 DATA = <100,0,0,0,0,50> AP = HERE AP.T = 0 AP.U = 1 UP = JTOU(AP) RP1 = RELUSER(UP, DATA) RP2 = RELUSER(1,UP, DATA)	10.11
GBASEDISP	求两个坐标间Base坐标系为基准的移动量的函数, 疏忽如坐标时 <位置1>和 <位置2>为相同类型 <Robot编号>: 可以指定编号选择Robot, 不输入时默认使用保存在JOB 程序的信息 <位置1>: 所有位置型变量 <位置2>: 所有位置型变量 <Base坐标系位置型变量> = GBASEDISP([<Robot编号>,] <位置1>, <位置2>)	POS AP1, AP2 XPOS XP1, XP2, D_X1, D_X2 AP1 = GP0 AP1.T = 0 AP2 = GP1 AP2.T = 0 D_X1 = GBASEDISP(AP1, AP2) D_X2 = GBASEDISP(1,AP1, AP2)	10.12

分类	说明 格式	使用例子	Ref.
GTOOLDISP	求两个坐标间Tool坐标系基准的移动量的函数， 输入坐标时<位置1>和 <位置2>为同样类型 <Robot编号>: 可以指定编号选择Robot， 不输入时默认使用保存在JOB 程序的信息 <位置1>: 所有位置型变量 <位置2>: 所有位置型变量 <Base坐标系位置型变量> = GTOOLDISP([<Robot编号>,] <位置1>, <位置2>)	POS AP1, AP2 XPOS XP1, XP2, D_X1, D_X2 AP1 = GP0 AP1.T = 1 XP1 = JTOB(AP1) AP2 = GP1 AP2.T = 1 XP2 = JTOB(AP2) D_X1 = GTOOLDISP(XP1, XP2) D_X2 = GTOOLDISP(1,XP1, XP2)	10.13
GUSERDISP	求两个坐标间用户坐标系基准的移动量的函数， 输入坐标时<位置1>和 <位置2>为同样类型 <Robot编号>: 可以指定编号选择Robot， 不输入时默认使用保存在JOB 程序的信息 <位置1>: 所有位置型变量 <位置2>: 所有位置型变量 <用户坐标系位置型变量> = GUSERDISP([<Robot编号>,] <位置1>, <位置2>)	POS AP1, AP2 UPOS UP1, UP2, D_U1, D_U2 AP1 = GP0 AP1.T = 1 AP1.U = 1 UP1 = JTOU(AP1) AP2 = GP1 AP2.T = 1 AP2.U = 1 UP2 = JTOU(AP2) D_U1 = GUSERDISP(UP1, UP2) D_U2 = GUSERDISP(1,UP1, UP2)	10.14

9.2 数学

分类	说明 格式	使用例子	Ref.
ABS	取绝对值 <实数型变量> = ABS(<输入值>)	REAL AA AA = ABS(-1.5)	10.15
DEG	弧度角变为角度值 <实数型变量> = DEG(<输入值>)	REAL AA AA = DEG(3.14)	10.16
RAD	角度值变为弧度角 <实数型变量> = RAD(<输入值>)	REAL AA AA = RAD(180)	10.17
POW	幂函数 <实数型变量> = POW (<底数>, <指数值>)	REAL AA AA = POW(2,4)	10.18
RND	小数部分四舍五入将实数值变为整数值 <实数型变量> = RND(<输入值>)	REAL AA AA = RND(14.8)	10.19
EXP	指数 e^x <实数型变量> = EXP(<输入值>)	REAL AA AA = EXP(3)	10.20
LN	自然对数 $\log_e X$ <实数型变量> = LN(<输入值>)	REAL AA AA = LN(15)	10.21
LOG	常用对数 $\log_{10} X$ <实数型变量> = LOG(<输入值>)	REAL AA AA = LOG(100)	10.22
SQRT	求平方根 <实数型变量> = SQRT(<输入值>)	REAL AA AA = SQRT(16)	10.23
SIN	正弦函数 <实数型变量> = SIN(<输入值>)	REAL AA AA = SIN(RAD(30))	10.24
ASIN	反正弦函数 <实数型变量> = ASIN(<输入值>)	REAL AA AA = ASIN(0.5)	10.25
COS	余弦函数 <实数型变量> = COS(<输入值>)	REAL AA AA = COS(0)	10.26
ACOS	反余弦函数 <实数型变量> = ACOS(<输入值>)	REAL AA AA = ACOS(0.5)	10.27
TAN	正切函数 <实数型变量> = TAN(<输入值>)	REAL AA AA = TAN(RAD(45))	10.28
ATAN	反正切函数 <实数型变量> = ATAN(<输入值>)	REAL AA AA = ATAN(-1.0)	10.29
ATAN2	第2反正切函数 <实数型变量> = ATAN2(<Y 比例>,<X 比例>)	REAL AA AA = ATAN2(1,-1)	10.30

分类	说明 格式	使用例子	Ref.
MIN	最小值函数 <实数型变量> = MIN(<输入值1>, <输入值2>)	REAL AA AA = MIN(1,2)	10.31
MAX	最大值函数 <实数型变量> = MAX(<输入值1>, <输入值2>)	REAL AA AA = MAX(1.1,2.2)	10.32
RAND	随机数生成函数 <数字变量> = RAND()	REAL AA AA = RAND()	10.33

9.3 字符串

分类	说明 格式	使用例子	Ref.
ASC	字符串第一个文字变为字符代码 <整数型变量> = ASC(<字符串>)	INT VAL VAL = ASC(“STATEMENT”)	10.34
BINS	整数变为二进制字符串 <字符串变量> = BINS(<整数值>)	DEFSTR STR STR = BINS(16)	10.35
CHR	整数变为文字 ASCII Code文字保存到字符串变量. <字符串变量> = CHR(<整数值>)	DEFSTR STR STR = CHR(65)	10.36
FLUSH	清除通信输入，输出缓存 <端口编号> 0 : 串口通信端口 10 ~ 12 : TCP I/P Server 端口 ~ 65,535: TCP I/P Client 端口 <选择清除缓存> : 1: 清除输入缓存 2: 清除输出缓存 3: 清除输入输出缓存 FLUSH <端口编号> <选择清除缓存>	FLUSH 0 1	10.37
FTOS	整数或实数变为字符串 <字符串变量> = FTOS(<输入值>)	DEFSTR STR STR = FTOS(10.5)	10.38
HTOS	整数变为16进制字符串 <字符串变量> = HTOS(<输入值>)	DEFSTR STR STR = HTOS(100)	10.39
SLEFT	字符串左侧开始取出输入的位数保存到字符串变量 <字符串变量> = SLEFT(<字符串>, <取出位数>)	DEFSTR STR STR = SLEFT(“TEST”,2)	10.40
SLEN	返回字符串长度 <整数型变量> = SLEN(<字符串>)	INT LEN LEN = SLEN(“STATEMENT”)	10.41

分类	说明 格式	使用例子	Ref.
SMID	从字符串指定位置开始取出输入的位数 <字符串变量> = SMID(<字符串>, <指定位置>, <取出位数>)	DEFSTR STR STR = SMID(“STATEMENT”, 2, 3)	10.42
SPOS	返回字符串1 内含有字符串2的位置 <整数型变量> = SPOS(<字符串1>, <字符串2>)	INT VAL VAL = SPOS(“TEST LINE”, “LINE”)	10.43
SRIGHT	字符串的右侧开始取出输入的位数保存到字符串变量 <字符串变量> = SRIGHT(<字符串>, <取出位数>)	DEFSTR STR STR = SRIGHT(“TEST”,2)	10.44
STRIN	向指定端口发送字符串 <端口编号>: RS-232(0), Ethernet(10, 11, 12) <等待时间>: 单位 1ms <字符串变量> = STRIN(<端口编号>, <等待时间>)	DEFSTR STR STR = STRIN(0,1000)	10.45
STROUT	发送字符串, 返回未传送的文字个数 <端口编号>: RS-232(0), Ethernet(10, 11, 12) <整数型变量> = STROUT(<端口编号>, <字符串>)	INT VAL VAL = STROUT(0, “TEST”)	10.46
SVAL	字符串变为数字 <数字变量> = SVAL(<字符串>)	INT VAL VAL = SVAL(“1234”)	10.47

10. 系统

10.1 错误

分类	说明 格式	使用例子	Ref.

SETERR	用户发出任意Alarm
	SETERR(<用户Alarm编号> [, <用户Alarm信息>])
		IF IN(0) == 1 THEN
		SETERR(1, “User Err 1”)
		ENDIF
			11.1

RESET	解除Alarm RESET[(<Robot编号>)]	RESET RESET(1)	11.2
RERROR	返回最后Alarm编号 <Robot编号>: 指定编号选择Robot (1 ~ 3), 输入0 代表所有Robot <整数型变量> = RERROR([<Robot编号>])	INT ECODE ECODE = RERROR(0) ECODE = RERROR(1)	11.3
RERRCNT	返回当前发生的Alarm个数 <Robot编号>: 指定编号选择Robot (1 ~ 3), 输入0 代表所有Robot <整数型变量> = RERRCNT([<Robot编号>])	INT ECOUNT ECOUNT = RERRCNT(0) ECOUNT = RERRCNT(1)	11.4
RERRCODE	返回相应索引编号的Alarm编号 <Robot编号>: 指定编号选择Robot (1 ~ 3), 输入0 代表所有Robot <索引编号>表示Alarm的发生顺序 <整数型变量> = RERRCODE([<Robot编号>,] <索引编号>)	INT ECODE ECODE = RERRCODE(0,1) ECODE = RERRCODE(1,1)	11.5
RERRTEXT	返回索引编号对应的Alarm内容 <Robot编号>: 指定编号选择Robot (1 ~ 3), 输入0 代表所有Robot <索引编号>表示Alarm的发生顺序 <字符串变量> = RERRTEXT([<Robot编号>,] <索引编号>)	DEFSTR ETEXT ETEXT = RERRTEXT(0,1) ETEXT = RERRTEXT(1,2)	11.6
READ_DERR	读取驱动Alarm编码 <Robot编号>: 指定编号选择Robot (1 ~ 3) <整数型变量> = READ_DERR([<Robot编号>,] <轴编号>)	INT VALUE VALUE = READ_DERR(1,1)	11.7

10.2 状态信息

分类	说明 格式	使用例子	Ref.
RSTATE	确认Robot状态. - <状态索引>的含义 0: 返回ALL, 所有状态信息 1: ALARM 状态 2: MC Status (Ready) (0: OFF, 1: ON) 3: Origin 完成状态 4: Inposition 状态 5: Reserved 6: SERVO state (ON/OFF) 状态 7 ~ 8: Reserved 9: System Mode 状态(1: System, 0: Robot) 10: Reserved 11: Emergency 状态 12 ~ 14: Reserveded 15: Robot Motion执行情况 16: 确认Fan Alarm情况 <整数型变量> = RSTATE(<Robot编号>, <状态索引>)	INT R_ST R_ST = RSTATE(1,0)	11.8
SWLIMIT	确认各轴的最小或最大移动范围. <最大/最小>: 0: 移动范围最小值, 1: 移动范围最大值) <实数型变量> = SWLIMIT(<Robot编号>, <轴编号>, <最大/最小>)	REAL L_LIMIT, R_LIMIT L_LIMIT = SWLIMIT(1,1,0) R_LIMIT = SWLIMIT(1,1,1)	11.9
READ_RPM	确认当前RPM 值. <实数型变量> = READ_RPM(<Robot编号>, <轴编号>)	REAL VALUE VALUE = READ_RPM(1,1)	11.10
READ_LOAD	确认当前负荷率值. <实数型变量> = READ_LOAD(<Robot编号>, <轴编号>)	REAL VALUE VALUE = READ_LOAD(1,2)	11.11
READ_TRQMAX	确认最大负荷率值. <实数型变量> = READ_TRQMAX(<Robot编号>, <轴编号>)	REAL VALUE VALUE = READ_TRQMAX(1,3)	11.12
READ_VEL	确认当前速度值. <实数型变量> = READ_VEL(<Robot编号>, <轴编号>)	REAL VALUE VALUE = READ_VEL(1,4)	11.13
READ_TRQ	确认当前扭矩值 <实数型变量> = READ_TRQ(<Robot编号>, <轴编号>)	REAL VALUE VALUE = READ_TRQ(1,1)	11.14
READ_MTEMP	确认当前点击编码器温度值 <实数型变量> = READ_MTEMP(<Robot编号>, <轴编号>)	REAL VALUE VALUE = READ_MTEMP(1,1)	11.15
READ_CTEMP	确认当前控制器内部温度值 <实数型变量> = READ_CTEMP()	REAL VALUE VALUE = READ_CTEMP()	11.16

10.3 日志及数据保存

分类	说明 格式	使用例子	Ref.
PRINT	保存监控画面中可以queen的日志 <索引> 范围: 0 ~ 999 PRINT(<索引>, <日志内容1> [{, <日志内容2>, … <日志内容N>}])	PRINT(0, “PRINT LOG”)	11.17
WLOG	按顺序将包括时间信息的用户日志保存到文件 WLOG(<日志内容1> [{, <日志内容2>, … <日志内容N>}]))	WLOG(“WLOG TEST”)	11.18
CLOG	※ 目前不支持.		11.19
WGPNT	全局Point变量保存到文件 <同步>: 选择文件同步到磁盘为止情况 (0: 不等待, 1: 等待) - 返回值: 0(正常), 1(写入错误) [<整数型变量> =] WGPNT(<同步>)	WGPNT(1) INT VAL VAL = WGPNT(1)	11.20
WGINT	全局整数变量保存到文件 <同步>: 选择文件同步到磁盘为止情况 (0: 不等待, 1: 等待) - 返回值: 0(正常), 1(写入错误) [<整数型变量> =] WGINT(<同步>)	WGINT(1) INT VAL VAL = WGINT (1)	11.21
WGFLT	全局实数变量保存到文件 <同步>: 选择文件同步到磁盘为止情况 (0: 不等待, 1: 等待) - 返回值: 0(正常), 1(写入错误) [<整数型变量> =] WGFLT(<同步>)	WGFLT(1) INT VAL VAL = WGFLT (1)	11.22
TACT_START	设置为测量工程时间的测量开始位置 TACT_START()	INT VAL TACT_START() DELAY 1000 VAL = TACT_STOP()	11.23
TACT_STOP	设置为测量工程时间的测量结束位置 [<整数型变量> =] TACT_STOP()		11.24

11. System模式专用命令

分类	说明 格式	使用例子	Ref.
PSEL	匹配指定Robot Thread 编号中输入的JOB 程序 <Thread 编号>: 可以使用1 ~ 4 [<整数型变量> =] PSEL(<Thread 编号>, “<JOB 文件名>”)	INT R_VAL R_VAL = PSEL(1,” TEST”)	12.1
PSTART	执行指定Robot Thread 编号的JOB程序 <Thread 编号>: 可以使用1 ~ 4 [<整数型变量> =] PSTART(<Thread 编号>)	PSTART(1)	12.2
PSTOP	停止指定Robot Thread 编号的 JOB程序 <Thread 编号>: 可以使用1 ~ 4 [<整数型变量> =] PSTOP(<Thread 编号>)	PSTOP(1)	12.3
PSTATE	读取指定Robot Thread 编号状态 <Thread 编号>: 可以使用1 ~ 4 <整数型变量> = PSTATE(<Thread 编号>)	INT STATE STATE = PSTATE(1)	12.4
PRESTART	重新执行指定Robot Thread 编号的 JOB 程序 <Thread 编号>: 可以使用1 ~ 4 [<整数型变量> =] PRESTART(<Thread 编号>)	PRESTART(1)	12.5
PEXIT	退出指定Robot Thread 编号的 JOB 程序 <Thread 编号>: 可以使用1 ~ 4 [<整数型变量> =] PEXIT(<Thread 编号>)	PEXIT(1)	12.6

12. 特定于TCP / IP Client的命令

分类	说明 格式	使用例子	Ref.
NET_OPEN	创建TCP I/P Client 套接字，请求访问Server 的函数 <IP 编号>: 输入Server的 IP 编号 <端口编号>: 输入Server的 端口编号 <等待时间>: 输入请求后接收等待时间(msec) <整数型变量> = NET_OPEN(<IP 编号>, <端口编号>, <等待时间>)	INT FD_NUM DEFSTR IP_NUM INT PORT_NUM INT WAIT_TIME IP_NUM = “192.168.1.196” PORT_NUM = 10000 WAIT_TIME = 1000 FD_NUM = NET_OPEN(IP_NUM, PORT_NUM, WAIT_TIME)	13.1
NET_CLOSE	结束控制器的TCP I/P 通信，删除Client 套接字的函数 <文件描述编号>: 用于访问套接字信息的文件描述 (File descriptor) 编号 [<整数型变量> =] NET_CLOSE(<文件描述编号>)	INT FD_NUM …… RET_VEL = NET_CLOSE(FD_NUM)	13.2
NET_STATUS	确认控制器的TCP I/P 通信状态的函数 <文件描述编号>: 用于访问套接字信息的文件描述 (File descriptor) 编号	INT FD_NUM …… RET_VEL = NET_STATUS(FD_NUM)	13.3

	<整数型变量> = NET_STATUS(<文件描述编号>)		
NET_KEEPALIVE	用于设置检查控制器TCP I/P Client 套接字通信状态的可选设定函数 <文件描述编号>: 用于访问套接字信息的文件描述 (File descriptor) 编号 <维持时间>: 开始传送最后数据包到开始发送探索包的时间 (sec) <探索最大次数>: 探索包无应答时最大重复次数 <探索周期>: 发送探索包的间隔 (sec) <整数型变量> = NET_KEEPALIVE(<文件描述编号>, <维持时间>, <探索最大次数>, <探索周期>)	INT FD_NUM …… INT RET_VEL INT KEEP_IDLE INT KEEP_CNT INT KEEP_INTERVAL FD_NUM = 10000 KEEP_IDEL = 10 KEEP_CNT = 20 KEEP_INTERVAL = 3 RET_VEL = NET_KEEPALIVE(FD_NUM, KEEP_IDLE, KEEP_CNT, KEEP_INTERVAL)	13.4
NET_DELIMITER	设置TCP I/P 通信输入/输出末尾字符串的函数 <文件描述编号>: 用于访问套接字信息的文件描述 (File descriptor) 编号 <末尾字符串类型>: 设置末尾字符串 (0: CRLF, 1: LF, 2: CR) <整数型变量> = NET_DELIMITER(<文件描述编号>, <末尾字符串类型>)	INT FD_NUM …… INT RET_VEL INT DELI DELI = 2 RET_VEL = NET_DELIMITER(<FD_NUM>,<DELI>)	13.5
NET_FD	返回访问TCP I/P Server 的 Client的Socket File Descriptor的函数 <索引>: Server的 端口编号指定索引 <整数型变量> = NET_FD(<索引>)	INT FD_NUM INT INDEX INDEX = 10 FD_NUM = NET_FD(<INDEX>)	13.6
NET_STRIN	发送到分隔符为止的字符串 <文件描述编号>: 用于访问套接字信息的文件描述 (File descriptor) 编号 <等待时间>: 单位 1ms <字符串变量> = NET_STRIN(<文件描述编号>, <等待时间>)	DEFSTR STR INT FD_NUM …… STR = NET_STRIN(FD_NUM,1000)	13.7
NET_STROUT	传送字符串, 返回没有传送的文字数 接收到分隔符为止的字符串 <文件描述编号>: 访问套接字信息的文件 <字符串变量>: 要输出的字符串 <整数型变量> = NET_STRIN(<端口编号>, <等待时间>)	INT VAL DEFSTR STR INT FD_NUM …… VAL = NET_STROUT(FD_NUM, STR)	13.8

第4章 命令说明

1. 命令解释格式

按照以下格式详细说明命令含义及其使用方法.

(1)<章节> (2)<命令名称>	
(3) Usage	
	说明命令用途相关内容.
(4) Syntax	
	说明正确使用命令的方法. 命令包括传递参数时使用 “<” 及 “>”表示. 传递多个参数时使用 “,”. 可以省略的因素用大括号“[“ 及 “]”包围. 数组使用 ‘[‘ 及 ‘]’表示. 因素集合中只需要一个因素时，集合写在大括号 “{” 及 “}” 内，各传递参数以分隔符”|”分割. 必须传递参数或函数等特殊用途时，在命令后面小括号“(“ 及 “)”内写入传递参数.
(5) Description	
	详细说明命令的特征及功能.
(6) Return value	
	如果使用命令后有作为其结果返回的值时，说明其返回值类型及用途.
(7) Arguments	
	说明包含的传递参数类型及用途.
(9) Examples	
	给出使用命令方法的例子.
(10) Related terms	
	其它相关命令或说明.

Figure 4-1 命令说明格式

2. 变量

2.1 INT

Usage

声明程序内部使用的整数型变量.

Syntax

INT [$]<变量名>['[' <数组个数> ']'] [, <变量名>['[' <数组个数> ']'], …]

Description

声明4字节大小的整数型变量.
声明后一定要初始化. 但, 无法声明的同时初始化.
Range: -2,147,483,648 ~ 2,147,438,647

1) 变量名前面使用"$" 符号可以作为全局变量使用. 作为全局变量使用时，需要在函数外部声明.
2) <变量名>可以使用英文字母和数字, 变量的第一位只能是英文字母或 "_" 符号.
3) 使用"," 符号声明2个以上的变量.
4) 指定'['<数组个数>']'可以将多个数据组合为一组使用.

Examples

```
INT $AA                          // 全局变量声明

MAIN                             // 主函数声明
    INT D_1, D_2, D_3[2]         // 局部变量声明
    $AA = 15                     // 全局变量初始化
    D_1 = 0                      // D_1中代入常数值
    D_2 = I100                   // D_2中代入全局变量 I100值
    D_3[0] = $AA                 // D_3中代入全局变量 $AA值
EOP                              // 主函数结束
```

Related terms

I, REAL, DEFSTR

1) 声明变量名时，以 "P"， "GP"， "I"， "F"开头的文字与系统中预声明的全局变量重复无法使用.
2) 变量的声明和初始化无法同时进行，请注意.

2.2 REAL

Usage

声明程序内部可以使用的实数型变量.

Syntax

REAL [$]<变量名>['[' <数组个数> ']'] [, <变量名>['[' <数组个数> ']'], …]

Description

声明4字节大小的实数型变量.
声明后必须初始化. 但, 无法声明的同时初始化.
Range: 3.402,823E-38 ~ 3.402,823E+38

1) 变量名前面使用"$" 符号可以作为全局变量使用. 作为全局变量使用时，需要在函数外部声明.
2) <变量名>可以使用英文字母和数字, 变量的第一位只能是英文字母或 "_" 符号.
3) 使用"," 符号声明2个以上的变量.
4) 指定'['<数组个数>']'可以将多个数据组合为一组使用.

Examples

```
REAL $AA                              // 全局变量声明

MAIN                                  // 主函数声明
    REAL D_1, D_2, D_3[2]             // 局部变量声明
    $AA = 15.15                       // 全局变量初始化
    D_1 = 0.58                        // D_1中代入常数值
    D_2 = F100                        // D_2中代入全局变量 F100值
    D_3[0] = $AA                      // D_3中代入全局变量 $AA值
EOP                                   // 主函数结束
```

Related terms

F, INT, DEFSTR

1) 声明变量名时，以 "P"，"GP"，"I"，"F"开头的文字与系统中预声明的全局变量重复无法使用.
2) 变量的声明和初始化无法同时进行，请注意.

2.3 DEFSTR

Usage

声明程序内部使用的字符串变量.

Syntax

DEFSTR [$]<变量名>['[' <数组个数> ']'] [, <变量名>['[' <数组个数> ']'], …]

Description

做声明字符串变量.
声明后必须初始化. 但, 无法声明的同时初始化.
最大字数不可以超过100字.

1) 变量名前面使用"$" 符号可以作为全局变量使用. 作为全局变量使用时, 需要在函数外部声明.
2) <变量名>可以使用英文字母和数字, 变量的第一位只能是英文字母或 "_" 符号.
3) 使用"," 符号声明2个以上的变量.
4) 指定'['<数组个数>']'可以将多个数据组合为一组使用.

Examples

```
DEFSTR $STR                          // 全局变量声明

MAIN                                 // 主函数声明
    DEFSTR D_1, D_2[2]               // 局部变量声明
    $STR = "STRING"                  // 全局变量初始化
    D_1 = "CHECK"                    // D_1中代入常数值
    D_2[0] = $STR                    // D_2中代入全局变量 $STR的值
EOP                                  // 主函数结束
```

Related terms

INT, REAL

1) 声明变量名时, 以 "P", "GP", "I", "F"开头的文字与系统中预声明的全局变量重复无法使用.
2) 变量的声明和初始化无法同时进行, 请注意.

2.4 POS

Usage

声明程序内部使用的Joint坐标系位置型变量.

Syntax

POS [$]<变量名>[‘[’ <数组个数> ‘]’] [, <变量名>[‘[’ <数组个数> ‘]’], …]

Description

声明Joint坐标系位置型变量. 最大可以保存DOF为止的位置值. 声明后必须初始化. 但, 无法声明的同时初始化. 初始化可以通过赋值位置型常数或各因素的方式进行.

1) 变量名前面使用“$” 符号可以作为全局变量使用. 作为全局变量使用时，需要在函数外部声明.
2) <变量名>可以使用英文字母和数字, 变量的第一位只能是英文字母或 “_” 符号.
3) 使用“,” 符号声明2个以上的变量.
4) 指定 ‘[’<数组个数>‘]’可以将多个数据组合为一组使用.

Examples

```
POS $AP                                          // 全局变量声明

MAIN                                             // 主函数声明
    POS D_1, D_2[2]                              // 局部变量声明
    $AP = <100,100,100,100,100,100 >             // 全局变量初始化, 6轴
    D_1.1 = 100.0                                // 局部变量各因素赋值方式
    D_1.2 = 120.1
    D_1.3 = 130.2
    D_1.4 = 140.3
    D_1.5 = 150.4
    D_1.6 = 160.5
    D_1.USED = 1                                 // D_1 初始化完成
    D_2[0] = $AP                                 // D_2中代入全局变量 $AP值
EOP                                              // 主函数结束
```

Related terms

XPOS, UPOS, P, GP

1) 声明变量名时，以 “P”, “GP”, “I”, “F”开头的文字与系统中预声明的全局变量重复无法使用.
2) 变量的声明和初始化无法同时进行，请注意.

2.5 XPOS

Usage

声明程序内部使用的Base坐标系位置型变量.

Syntax

XPOS [$]<变量名>[‘[’ <数组个数> ‘]’] [, <变量名>[‘[’ <数组个数> ‘]’], …]

Description

声明Base坐标系位置型变量. 最大可以保存DOF为止的位置值. 声明后必须初始化. 但, 无法声明的同时初始化. 初始化可以通过赋值位置型常数或各因素的方式进行.

1) 变量名前面使用“$” 符号可以作为全局变量使用. 作为全局变量使用时，需要在函数外部声明.
2) <变量名>可以使用英文字母和数字, 变量的第一位只能是英文字母或 “_” 符号.
3) 使用“,” 符号声明2个以上的变量.
4) 指定‘[’<数组个数>‘]’可以将多个数据组合为一组使用.

Examples

```
XPOS $AP                                                    // 全局变量声明
MAIN                                                        // 主函数声明
    XPOS D_1, D_2[2]                                        // 局部变量声明
    $AP = <100,100,100,100,100,100,ARM = 0,T = 1>           // 全局变量初始化, 6轴
    D_1.1 = 100.0                                           // 局部变量各因素赋值方式
    D_1.2 = 120.1
    D_1.3 = 130.2
    D_1.4 = 140.3
    D_1.5 = 150.4
    D_1.6 = 160.5
    D_1.ARM = 111                                           // 代入Robot姿势
    D_1.T = 2                                               // 代入Robot Tool 编号
    D_1.USED = 1                                            // D_1 初始化完成
    D_2[0] = $AP                                            // D_2中代入全局变量 $AP值
EOP                                                         // 主函数结束
```

Related terms

POS, UPOS, P, GP

1) 声明变量名时，以 “P”, “GP”, “I”, “F”开头的文字与系统中预声明的全局变量重复无法使用.
2) 变量的声明和初始化无法同时进行，请注意.

2.6 UPOS

Usage

声明程序内部使用的用户坐标系位置型变量.

Syntax

UPOS [$]<变量名>['[' <数组个数> ']'] [, <变量名>['[' <数组个数> ']'], …]

Description

声明用户坐标系位置型变量. 最大可以保存DOF为止的位置值. 声明后必须初始化. 但, 无法声明的同时初始化. 初始化可以通过赋值位置型常数或各因素的方式进行.

1) 变量名前面使用"$" 符号可以作为全局变量使用. 作为全局变量使用时, 需要在函数外部声明.
2) <变量名>可以使用英文字母和数字, 变量的第一位只能是英文字母或 "_" 符号.
3) 使用"," 符号声明2个以上的变量.
4) 指定'['<数组个数>']'可以将多个数据组合为一组使用.

Examples

```
UPOS $AP                                        // 全局变量声明
MAIN                                            // 主函数声明
    UPOS D_1, D_2[2]                            // 局部变量声明
    $AP = <100,100,100,100,100,100 >            // 全局变量初始化, 6轴
    D_1.1 = 100.0                               // 局部变量各因素赋值方式
    D_1.2 = 120.1
    D_1.3 = 130.2
    D_1.4 = 140.3
    D_1.5 = 150.4
    D_1.6 = 160.5
    D_1.ARM = 111                               // 代入Robot姿势
    D_1.T = 2                                   // 代入Robot Tool 编号
    D_1.U = 3                                   // 代入Robot User 编号
    D_1.USED = 1                                // D_1 初始化完成
    D_2[0] = $AP                                // D_2中代入全局变量 $AP值
EOP                                             // 主函数结束
```

Related terms

POS, XPOS, P, GP

1) 声明变量名时, 以 "P", "GP", "I", "F"开头的文字与系统中预声明的全局变量重复无法使用.
2) 变量的声明和初始化无法同时进行, 请注意.

2.7 Program Position P

Usage

系统中预声明的Joint坐标系位置型变量作为位置值保存到文件中，从属于程序. 相应程序内作为全局变量使用.

Syntax

P<编号> 或 P '[' <编号> ']'

Description

与声明为POS的Joint坐标系位置型变量相同，最大可以保存DOF为止的位置值
与GP不同，各JOB 程序文件可以保存不同的值，仅在相应程序内可以使用的全局变量.
与局部变量不同，程序中可以不声明直接使用.
执行程序时无法编辑，各别因素只读.

1) <编号>可以如数组输入， 可以直接输入使用. 文字P和 <编号> 之间不可以有空白.
2) <编号>的范围为0 ~ 999，可以使用供1,000个.

Examples

```
REAL AA, BB                 // 声明实数型变量
AA = P0.1                   // AA变数中代入P0.1 值
BB = P[0].2                 // BB 变数中代入 P[0].2 值
SVON                        // 伺服ON
JMOV P0                     // JMOV移动到P0位置
```

Related terms

POS, XPOS, UPOS, GP

1) 程序执行过程中无法变更值. 仅在Teaching模式下可以变更数据.

2.8 Global Integer I

Usage

系统中预声明的整数型全局变量，在整体程序通用的变量.

Syntax

I<编号> 或 I ‘[’<编号>‘]’

Description

与声明为INT的整数型变量类似，可以保存4字节的整数值.
Range: -2,147,483,647 ~ 2,147,438,647

与局部变量不同，程序中不声明直接可以使用.
程序执行过程中可以更改值. 修改值后可以使用WGINT 函数将更改的值保存到文件中.

1) <编号>可以如数组输入， 可以直接输入使用. 文字I和 <编号> 之间不可以有空白.
2) <编号>的范围为0 ~ 999，可以使用供1,000个.

Examples

```
I[0] = 100                    // 100代入到 第0个 I 变量
I[999] = -1                   // -1代入到 第999个 I 变量
```

Related terms

INT, F, WGINT

1) 程序执行过程中为了保存变更的值，一定要使用WGINT 函数.
2) 程序执行过程中变更全局变量后没有执行保存命令，则电源切断或重启后变更数据保存.
3) 文件保存中突然断电时，有可能发生数据丢失.

2.9 Global Real F

Usage

系统中预声明的实数型全局变量，在整体程序通用的变量.

Syntax

F<编号> 或F ‘[’<编号>‘]’

Description

与声明为REAL的实数型变量类似，可以保存4字节的整数值.
Range: 3.402,823E-38 ~ 3.402,823E+38

与局部变量不同，程序中不声明可以直接使用.
程序执行过程中可以更改值. 修改值后可以使用WGFLT 函数将更改的值保存到文件中.

1) <编号>可以如数组输入， 可以直接输入使用. 文字F和 <编号> 之间不可以有空白.
2) <编号>的范围为0 ~ 999，可以使用供1,000个.

Examples

```
F[0] = 100                  // 常数值代入第0个 F 变量
F[999] = -1                 //常数值代入第999个F变量
```

Related terms

REAL, I, WGFLT

1) 程序执行过程中为了保存变更的值，一定要使用WGFLT 函数.
2) 程序执行过程中变更全局变量后没有执行保存命令，则电源切断或重启后变更数据保存.
3) 文件保存中突然断电时，有可能发生数据丢失.

2.10 Global Position GP

Usage

系统中预声明的Joint坐标系位置型变量，在整体程序通用的变量.

Syntax

GP<编号> 或GP ‘[’<编号>‘]’

Description

与声明为POS的Joint坐标系位置型变量相同，最大可以保存DOF为止的位置值.
与局部变量不同，程序中可以不声明直接使用.
程序执行过程中可以更改值. 修改值后可以使用WGPNT 函数将更改的值保存到文件中.
可以代入位置型常数的值，也可以对各别因素修改值.

1) <编号>可以如数组输入，可以直接输入使用. 文字GP和 <编号> 之间不可以有空白.
2) <编号>的范围为0 ~ 1,999，可以使用供2,000个.

Examples

```
GP[20] = <100,100,100,100,100,100>          // 常数值代入第20个 GP 变量
GP[20].1 = 200                              // 为GP 第20个因素赋值
GP[20].2 = 50
GP[20].3 = GP[20].1+GP[20].2
GP[20].USED = 1
```

Related terms

POS, XPOS, UPOS, P, WGPNT

1) 程序执行过程中为了保存变更的值，一定要使用WGPNT 函数.
2) 程序执行过程中变更全局变量后没有执行保存命令，则电源切断或重启后变更数据保存.
3) 文件保存中突然断电时，有可能发生数据丢失.

2.11 TMR

Usage

系统中预声明的Timer变量，指定计数器值或读取当前计数值时使用的变量.

Syntax

TMR(<编号>) = <值>
<变量> = TMR(<编号>)

Description

与局部变量不同，程序中不声明可以直接使用.
输入整数值的瞬间开始赋值，以10ms 间隔加一.
遇到Public参数1.HW_CONF(2/3)→3.TMR 项目中设定的设定值则初始化.

1) <编号>直接写入数字 “(” 和 “)” 之间使用.
2) 输入值只能使用整数型变量及常数.

Return value

<变量>	Data type: <整数型> <整数型>的范围为0 ~ 60,000

Arguments

<编号>	Data type: <整数型> <编号>的范围为0 ~ 1.

Examples

```
INT AA, BB                  // 声明整数型变量
AA = -100                   // AA 变量中代入初始值
TMR(0) = 50                 // TMR 0号中代入Count值50
TMR(1) = AA                 // TMR 1号中输入变量AA 值
DLAY 100                    // 时间延迟
BB = TMR(0)                 // BB 变量中保存TMR 0号Count值
```

2.12 MVR

Usage

系统中预声明的变量，返回整体移动距离中占的移动距离比例.

Syntax

[<变量> =] MVR[(<Robot编号>)]
MVR[(<Robot编号>)] = <值>

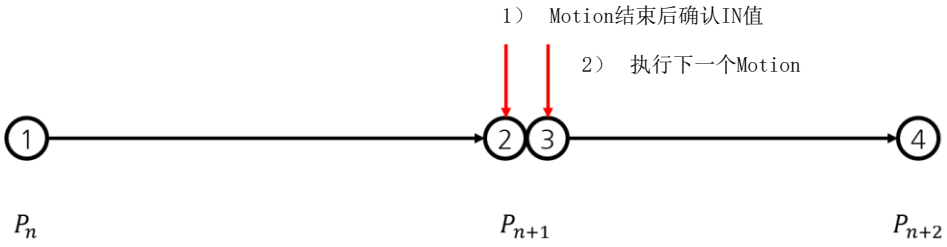
Description

与局部变量不同，程序中不声明可以直接使用..
返回整体移动距离中占据的百分比值.

- MVR = (当前Robot移动距离/ Robot整体移动距离) * 100

常用于Robot动作中条件分段及 I/O 并行处理中. 此时，与WITH … ENDWT语句一起使用.
如下图，位置(P_n, P_{n+1})的移动中同时执行下一个位置 (P_{n+1}, P_{n+2}) 移动时，预先处理必要条件，缩短整体程序执行速度.

1) 一般使用例子



2) MVR 使用例子

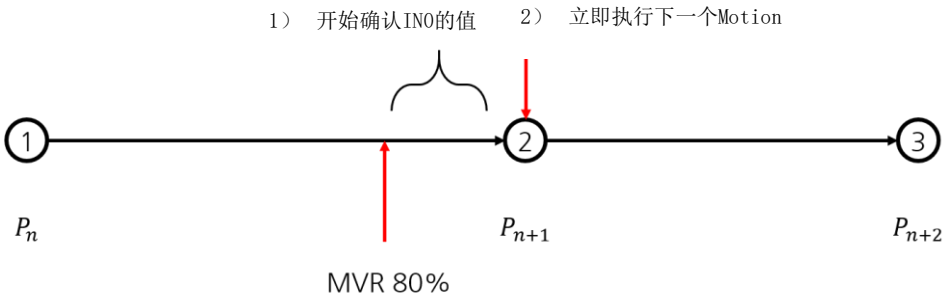


Figure 4-2 一般使用及MVR 使用比较例子

上图中通过例子说明位置(P_n, P_{n+1})的移动过程中，预先确认下一个位置 (P_{n+1}, P_{n+2}) 移动执行条件IN0的值，提高了整体程序的执行速度.

1) 输入<Robot编号>保存 JOB 程序时，可以获取非选择的Robot的其它Robot移动距离比.

Return value

<变量>	Data type: <实数型>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.

Examples

Example 1

```
REAL VALUE                          // 声明实数型变量
VALUE = MVR                         // 代入MVR 值
VALUE = MVR(2)                      // 代入Robot 2号的 MVR 值
```

Example 2

```
MVR = 0                             // MVR 值设置为0
WITH                                // 开始WITH语句
     JMOV P0                        // 从当前位置JMOV到P0
     WHILE MVR < 100                // MVR 值小于100时重复执行
       IF (MVR >80)&&(IN0 == 1) THEN // 移动距离80%以上, IN0为 ON(1)时
         OUT0 = 1                   // OUT0的信号变更为ON(1)
         GOTO BB                    // 移动到无条件转移BB
       ENDIF                        // 条件语句结束
     ENDWL                          // 条件重复语句结束
ENDWT                               // WITH 语句结束
LABL BB                             // 声明无条件转移BB
     JMOV P1                        // 从当前位置JMOV到P1
```

Related terms

HERE, HERE_REF, WITH … ENDWT, WHILE … ENDWL, IF …(ELSE IF …ELSE…) ENDIF

1) 移动距离且高速运行时，MVR 值快速变化有可能无法确认条件. 请注意设置适当MVR值. 此外，此时请降低Robot速度使用.

2.13 HERE

Usage

系统中预声明的变量，当前位置返回为Joint角度值.

Syntax

<变量> = HERE
<变量> = HERE(<Robot编号>)

Description

将Robot的当前位置返回为Joint角度值. 向其它位置型变量保存值时发生Alarm.

1) 输入<Robot编号>保存 JOB 程序时，也可以获取非选择的其它Robot位置值.

Return value

<变量>	Data type: <Joint坐标系位置型>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为1 ~ 3.

Examples

Example 1	POS CUR_POS CUR_POS = HERE	// 声明Joint坐标系位置型变量 // 代入当前Joint角度值
Example 2	POS R2_POS R2_POS = HERE(2)	// 声明Joint坐标系位置型变量 // 代入第2个Robot的当前Joint角度值

Related terms

MVR, POS, P, GP, HERE_REF

2.14 HERE_REF

Usage

系统中预声明的变量，当前参照位置返回为Joint角度值.

Syntax

<变量> = HERE_REF
<变量> = HERE_REF (<Robot编号>)

Description

将Robot的当前参照位置返回为Joint角度值. 向其它位置型变量保存值时发生Alarm.

1) 输入<Robot编号>保存 JOB 程序时，也可以获取非选择的其它Robot位置值.

Return value

<变量>	Data type: <Joint坐标系位置型>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为1 ~ 3.

Examples

```
Example 1
POS CUR_POS                    // 声明Joint坐标系位置型变量
CUR_POS = HERE_REF             // 代入当前参照Joint角度值

Example 2
POS R2_POS                     // 声明Joint坐标系位置型变量
R2_POS = HERE_REF(2)           // 代入第2个Robot的当前参照Joint角度值
```

Related terms

MVR, POS, P, GP, HERE

3. 预处理机及注释

3.1 INCLUDE

Usage

将特定程序文件引入到当前位置，当成一个程序使用.

Syntax

INCLUDE “<JOB 文件名>”

Description

INCLUDE 命令在写主函数前使用.
可以使用引入的JOB 文件内部的函数及全局变量.
执行引入文件的JOB 主程序时，需要在JOB 文件后加上 “_MAIN()”.

1) <JOB 文件>中只允许有除文件结束符 “.” 和扩展名 “JOB”的其它文件名.

Examples

Example 1

```
INCLUDE “INC”                    // 引入文件INC.JOB

MAIN                             // 主函数开始
    …
EOP                              // 主函数结束
```

Example 2

```
INCLUDE “INC”                    // 引入文件INC.JOB

MAIN                             // 主函数开始
    INC_MAIN()                   // 执行INC 文件的主函数
EOP                              // 主函数结束
```

Related terms

JCALL

3.2 注释 (Comment)

Usage

对程序执行没有任何作用.

Syntax

```
// [<注释>]
/* [<注释>] */
```

Description

一般注释用于表达记述或参照、说明语句使用.
执行时没有任何影响.

1) 使用“//” 符号时注释相应符号右侧一行.
2) 使用“/*” 及 “*/” 符号时，注释两个符号之间的所有字符串. 用于注释多行.

Arguments

<注释>	所有字符串

Examples

Example 1

```
MAIN                    // 声明主函数
    SVON                // 伺服ON
    JMOV P0             // 从当前位置JMOV到P0
    //DLAY 1000         // 注释一行
    JMOV P1             // 从当前位置JMOV到P1
    DLAY 1000           // 1秒延迟
EOP                     // 主函数结束
```

Example 2

```
MAIN                    // 声明主函数
    SVON                // 伺服 ON
    /*                  // 注释多行开始
    JMOV P0
    DLAY 1000
    */                  // 注释多行结束
    JMOV P1             // 从当前位置JMOV到P1
    DLAY 1000           // 1秒延迟
EOP                     // 主函数结束
```

4. 运算符

4.1 代入运算符

Usage

用于对变量赋新值. 此运算符可以用到常数到表达式的所有格式.

Syntax

〈变量〉 = 〈所有常数, 变量或表达式〉

Description

转换成符合保存变量类型的数据保存.
无法转换类型时发生Alarm.

Return value

〈变量〉	Data type: 〈所有变量〉

Arguments

Data type: 与值或表达式结果值类型相同的值

Examples

Example 1

```
INT AA, BB, CC                  // 声明整数型变量
AA = 1                          // 变量中代入常数值
BB = AA                         // 变量中代入变量
CC = (AA+BB)*5                  // 变量中代入表达式
```

Example 2

```
DEFSTR STR1, STR2               // 声明字符串变量
STR1 = “CONSTANT”               // 变量中代入常数值
STR2 = STR1+” ASSIGN”           // 变量中代入表达式
```

Example 3

```
POS AP, BP, CP, DP              // 声明位置型变量
AP = P0                         // 变量中代入程序变量
BP = GP0                        // 变量中代入全局变量
CP = <0,0,0,0,0,0>              // 变量中代入常数
DP = AP+BP+CP                   // 变量中代入表达式
```

4.2 计算运算符

Usage

执行计算运算的运算符.

Syntax

<变量> = <A项><计算运算符><B项>

Description

用于指定计算运算类型. 遵循下表规则.

符号	含义	可使用的数据类型	
		A项	B项
+	加法	所有数据类型	
-	减法	除字符串的所有数据类型	
*	乘法	除字符串的所有数据类型	整数或实数
/	除法	除字符串的所有数据类型	整数或实数
%	取余	整数(输入实数也会转换为整数)	

取余 (%)时的符号规则请参考第2章9.3计算运算符 章节.

Return value

<变量> Data type: <所有变量>

Arguments

Data type: 值或表达式结果值与返回值相同类型值

Examples

Example 1
```
I2 = I0 + I1                    // 整数加法
GP2 = P0 + P1                   // 位置型加法
STR = “ADD” + “STRING”          // 字符串加法
```

Example 2
```
F2 = F0 - F1                    // 实数减法
GP2 = GP0 - GP1                 // 位置型减法
GP3 = GP0 - 1                   // 位置型各别数据均减一
```

Example 3
```
I2 = I0 * I1                    // 整数乘法
GP2 = P0 * 1.5                  // 位置型各别数据均乘以1.5
```

Example 4
```
F2 = F0 / F1                    // 实数除法
GP2 = <100,100,100,100,100,100> / 2   // 位置型各别数据均除 2
```

Example 5
```
I2 = I0 % I1                    // 整数取余
```

4.3 关系运算符

Usage

比较被运算值，根据指定的关系判断有效性的运算符. 主要用于条件表达式.

Syntax

[<变量> =]<A项><关系运算符><B项>

Description

主要以整数型和实数型资料为对象, 用于IF 语句或WHILE 语句条件表达式. 下表收货名各符号的含义.

符号	含义
<	小于
>	大于
<=	小于等于
>=	大于等于
==	等于
!=	不等于

Return value

<变量> Data type: <整数型变量>
关系运算结果返回逻辑值(true: 1, false: 0).

Arguments

Data type: <整数或实数型>

Examples

Example 1

```
IF IO > I1 THEN                 // IO的值比I1 大时,
    OUTO = 1                    // OUTO的信号输出为1
ENDIF                           // 条件语句结束
```

Example 2

```
INT VALUE                       // 声明整数型变量
VALUE = FO > F1                 // FO和 F1值相比后FO大时返回1 F1大时返回0
```

4.4 逻辑运算符

Usage

为确认一个或两个被运算值的逻辑关系使用的运算符.

Syntax

[<变量> =]<A项><逻辑运算符><B项>
[<变量> =]<逻辑运算符><A项>

Description

逻辑运算符不执行一般计算转换. 反而以逻辑值 (true: 1, false: 0)概念评价各被运算值.

符号	含义
&&	逻辑 AND 运算符在两个被运算值非0时返回1. 被运算值中任何一个为0，则结果为0.
||	逻辑OR 运算符在哪个被运算值均为0时返回0. 被运算值中任意一个为非0，则结果为1.
!	逻辑非 (逻辑 NOT) 运算符在相应被运算值true(非0值) 时返回0， false(0)时返回1.

Return value

<变量> Data type: <整数型>
逻辑运算的结果返回逻辑值(true: 1, false: 0).

Arguments

Data type: <整数或实数型>

Examples

Example 1

```
IF I0 < I1 && I2 > I3 THEN          // I0的值小于I1且I2的值大于I3时
    OUT(0) = 1                      // OUT0信号赋值1
ENDIF                               // 条件语句结束
```

Example 2

```
IF IN0 == 1 || IN1 == 1 THEN        // IN0 值为 1或IN1 值为1时
    JMOV P0                         // 从当前位置JMOV到P0
ENDIF                               // 条件语句结束
```

Example 3

```
INT VALUE                           // 声明整数型变量
VALUE = 0                           // 初始化值
IF !VALUE THEN                      // VALUE 变量的值非1(true)时
    OUT(0) = 1                      // OUT0信号赋值1
ENDIF                               // 条件语句结束
```

4.5 位运算符

Usage

对1个或两个被运算符应用比特单位的运算.

Syntax

[<变量> =]<A项><位运算符><B项>
[<变量> =]<位运算符><A项>

Description

位运算符与逻辑运算符类似，但最大的差距我使用比特单位的运算. 此外用于将整体比特向左或向右以比特单位移动时使用.

符号	含义
&	比特单位的AND 运算符在两侧比特均为1时结果比特设置为1. 否则结果比特设置为0.
^	比特单位的XOR 运算符在一个比特为0，另一个比特为1时相应比特设置为1. 否则相应结果比特设置为0.
|	比特单位的OR 运算符在任意一侧比特为1时，相应结果比特设置为1. 否则相应结果比特设置为0.
<<	对第一个被运算值向左移动第二个被运算值指定的位数.
>>	对第一个被运算值向右移动第二个被运算值指定的位数.
~	比特 NOT 运算符返回被运算符比特的相反值. 比特为1时返回0，比特为0时返回1.

Return value

<变量> Data type: <整数型>
关系运算结果返回逻辑值 (true: 1, false: 0).

Arguments

Data type: <整数型>
被运算值均为整数值.

Examples

Example 1

```
IF IO & I1 THEN                    // IO和 I1的比特AND 运算为真时
    OUTO = 1                       // OUTO的信号赋值为1
ENDIF                              // 条件语句结束
```

Example 2

```
INT VALUE                          // 声明整数型变量
VALUE = IO ^ I1                    // 保存IO和 I1的比特XOR 运算
```

Example 3

```
IF IO | I1 THEN                    // IO和I1的比特OR运算为真时
    JMOV PO                        // 从当前位置JMOV到PO
ENDIF                              // 条件语句结束
```

Example 4

```
INT A, B                           // 声明整数型变量
A = OB1011                         // 变量初始化
B = A<<2                           // 向左移动2位
```

Example 5

```
INT A, B                           // 声明整数型变量
A = OB1011                         // 变量初始化
B = A>>1                           // 向右移动1位
```

Example 6

```
INT A, B                           // 声明整数型变量
A = OB1011                         // 变量初始化
B = ~A                             // 保存A 变量值的反转值
```

4.6 括号

Usage

表达式或下位表达式周边利用一对括号表示，忽略一般运算符优先顺序，赋予更高优先级的运算符.

Syntax

(<表达式>)

Description

组合括号内内容.

Examples

```
INT A, B, C, RES          // 声明整数型变量
A = 1                     // 变量初始化
B = 2
C = 3

RES = (A+B)*C             // 比起一般有限级高的乘法(*)运算，先处理加法(+)运算
```

5. 程序控制

5.1 MAIN … EOP (程序开始/结束)

Usage

表示主程序的开始及结束.

Syntax

```
MAIN
    …
EOP
```

Description

所有程序只允许有一个主函数, 必须有一个主函数时才可以执行.

1) MAIN 和 EOP配对使用.
2) EOP(End of Program)表示主函数程序块的结束，执行 JOB 程序时遇到此语句JOB将结束. 但, 被JCALL调用的JOB遇到 EOP 语句时结束被调用的JOB的执行，回到调用的JOB.

Examples

```
MAIN                    // 主函数开始
    SVON                // 伺服 ON
    JMOV P1             // 从当前位置JMOV到P1
    JMOV P2             // 从当前位置JMOV到P2
EOP                     // 主函数结束
```

Related terms

FUNC … END

5.2 FUNC … END (函数开始/结束)

Usage

表示函数开始及结束.

Syntax

```
FUNC <返回类型> <函数名> ([<参数>])
    …
    [RET <变量>]
END
```

Description

经常重复使用或程序代码长时，可以分割为其它函数制作.
主程序内任何位置都可以声明多个函数使用.
主程序中输入函数名和参数，可以执行相应函数， 也可以递归调用 (Recursive Call). 但, 函数的调用深度不可以超过100.

1) FUNC和 END 必须配对使用.
2) END表示函数快的结束，执行函数块时遇到此命令则结束函数的执行.
3) <返回类型>可以使用包括 VOID(无返回值)的所有数据类型 (INT 及 REAL, DEFSTR, POS, XPOS, UPOS). 但无法使用系统预声明的全局变量(P, GP, I, F) 以及系统变量 (TMR 等).
4) <函数名>可以由英文字母和数字组成, 无法使用预声明的全局变量.
5) <参数>可以省略，根据需求可以接收函数内部运算中使用的各种变量的参数使用. <参数>可以使用所有数据类型 (INT 及 REAL, DEFSTR, POS, XPOS, UPOS). 但无法使用系统预声明的全局变量(P, GP, I, F) 以及系统变量 (TMR 等).

Arguments

<参数>	Data type: <所有数据类型, 没有时省略>

Examples

Example 1

- **返回类型 (VOID)及无参数类型**

```
FUNC VOID TEST()                    // 声明返回值及无参数的函数
    SVON                            // 伺服 ON
    JMOV P0                         // 执行Motion
    JMOV P1
END                                 // 函数结束

MAIN                                // 主函数声明
    TEST()                          // 调用函数
EOP                                 // 主函数结束
```

Example 2

- **无返回类型 (VOID)，有参数(Joint坐标系位置型)类型**

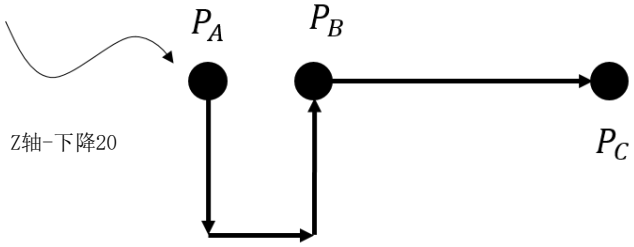


Figure 4-3 函数动作例子

```
MAIN                                    // 主函数声明
    MOVE(GP[0], GP[1])                  // A, B positional variable as function call
    LMOV GP[2]                          // LMOV to GP [2] (C position)
EOP                                     // 主函数结束

FUNC VOID MOVE(POS A, POS B)            // 无返回值，有位置型参数的函数声明
    XPOS TEMP                           // Base坐标系位置型变量声明
    SVON                                // 伺服ON
    JMOV A                              // JMOV 到A 位置
    TEMP = JTOB(A)                      // A 位置转换为Base坐标系位置值保存
    TEMP.3 = TEMP.3 - 20                // 保存A 位置沿着Z轴下降20 的位置值
    LMOV TEMP                           // LMOV 到TEMP 位置
    TEMP = JTOB(B)                      // B位置转换为Base坐标系位置值保存
    TEMP.3 = TEMP.3 - 20                // 保存B位置沿着Z轴下降20 的位置值
    LMOV TEMP                           // LMOV 到TEMP 位置
    LMOV B                              // LMOV 到B 位置
END                                     // 函数结束
```

Example 3

- **有返回类型 (实数)，无参数类型**

```
MAIN                                    // 主函数声明
    REAL VALUE                          // 声明实数型变量
    VALUE = TEST()                      // 调用函数及VALUE 变量中保存返回值
    PRINT(0, VALUE)                     // 保存至监控日志0号索引中
EOP                                     // 主函数结束

FUNC REAL TEST()                        // 返回实数值且无参数的函数声明
    REAL RES                            // 声明实数型变量
    RES = F[0]+F[1]                     // 实数值的加法
    RET RES                             // 返回结果值
END                                     // 函数结束
```

Example 4

- **返回类型（字符串）及参数(整数，字符串)均有的类型**

```
MAIN                                    // 主函数声明
    INT R_VAL                           // 声明整数型变量
    DEFSTR STR                          // 字符串变量声明
    WHILE 1                             // 重复语句开始
        STR = TEST(“IN(0)”, IN(0))      // 调用函数及STR 变量中保存返回值
        PRINT(0, STR)                   // 保存至监控日志0号索引中
        STROUT(0,STR)                   // 通过RS-232 端口传送字符串
        DLAY 100                        // 0.1秒延迟
    ENDWL                               // 重复语句结束
EOP                                     // 主函数结束
                                        // 返回字符串，拥有字符串及整数参数的函
FUNC DEFSTR TEST(DEFSTR A, INT B)       //数声明
    DEFSTR DATA, RES                    // 字符串变量声明
    DATA = HTOS(B)                      // 整数值转换为 16进制字符串
    RES = A + “:” + DATA                // 相加各字符串保存到变量
    RET RES                             // 返回字符串
END                                     // 函数结束
```

Example 5

- **返回类型及参数均有的类型（Base坐标系位置型例子）**

```
MAIN                                    // 主函数声明
    XPOS A_P, B_P, X_T                  // Base位置型变量声明
    A_P = JTOB(GP[0])                   // 获得Base坐标系位置值
    B_P = JTOB(GP[1])
    X_T = MD_X(A_P, B_P)                // 调用函数及 X_T 变量中保存返回值
    SVON                                // 伺服ON
    LMOV X_T                            // LMOV至返回位置
EOP

FUNC XPOS MD_X(XPOS A, XPOS B)          // 拥有Base坐标系位置型参数和返回值的函
    XPOS RES                            //数声明
    RES = (A+B)/2                       // Base位置型变量声明
    RET RES                             // 保存两个Base坐标值相加除2的结果
END                                     // 返回Base坐标系位置值
                                        // 函数结束
```

Example 6

- **递归函数例子**

```
INT $COUNT                          // 全局变量声明
MAIN                                // 主函数声明
    $COUNT = 0                      // 变量初始化
    RECALL()                        // 调用函数
EOP                                 // 主函数结束

FUNC VOID RECALL()                  // 函数声明
    $COUNT = $COUNT+1               // 全局变量 $COUNT值加一
    PRINT(0,” COUNT”,$COUNT)        // 输出监控日志
    IF $COUNT < 50 THEN             // 全局变量 $COUNT值小于 50时
        RECALL()                    // 递归调用
    ENDIF                           // 条件语句结束
END                                 // 函数结束
```

Related terms

MAIN … EOP, RET

5.3 RET

Usage

结束函数的执行返回值，恢复到调用函数的位置.

Syntax

RET <返回值>

Description

无法在主函数使用.

<返回值>作为函数的返回值，由函数声明决定.
函数声明时 <返回类型>为 VOID时无法使用 RET 命令.
<返回值>可以在除 VOID 的所有数据类型 (INT 及 REAL, DEFSTR, POS, XPOS, UPOS) 使用. 但无法使用系统预声明的全局变量(P, GP, I, F) 以及系统变量 (TMR 等).

Arguments

Data type: 由函数声明决定, VOID时省略

Examples

```
FUNC INT TEST(INT A, INT B)     // 返回整数型值，拥有参数的TEST 函数声明
     INT VALUE                  // 声明整数型变量
     VALUE = A*B
     RET VALUE                  // 返回计算值
END                             // 函数结束
```

Related terms

FUNC … END

5.4 FOR … TO … [BY] … NEXT (重复执行语句)

Usage

直到条件变为false(0)重复执行程序块.

Syntax

```
FOR <初始化语句> TO <结束条件> [BY <增量>]
    …
NEXT
```

Description

主要形成执行指定次数的循环. 设置初始值后, 直到满足结束条件为止重复执行FOR程序块.

1) FOR … TO … NEXT必须一起使用.
2) <初始化语句>用于初始化循环索引. 只执行1次. 可以使用表达式或变量.
3) <结束条件>用于确认循环结束基准.
4) <增量>可以指定. 省略时索引每增加1.

Examples

Example 1

```
INT AA, SUM                    // 声明整数型变量
SUM = 0                        // 变量初始化
FOR AA = 1 TO 10 BY 2          // 从1到 10每增加 2
    SUM = SUM+AA               // SUM 变量和 AA 变量相加保存到SUM 变量
    JMOV P[SUM]                // JMOV 到SUM 值相应的变量位置
NEXT                           // FOR语句结束
PRINT(0, SUM)                  // 输出监控日志
```

Example 2

```
REAL AA, SUM                   // 声明实数型变量
SUM = 0.0                      // 变量初始化
FOR AA = 0.0 TO 1 BY 0.1       // 从0到 1每增加0.1
  SUM = SUM+AA                 // SUM变量和 AA 变量相加保存到SUM 变量
NEXT                           // FOR语句结束
PRINT(0,SUM)                   // 输出监控日志
```

Related terms

WHILE… ENDWL

5.5 WHILE … ENDWL (条件重复执行语句)

Usage

条件表达式为真(true)时重复执行程序块.

Syntax

```
WHILE <条件表达式>
    …
ENDWL
```

Description

WHILE 语句在满足条件 (条件表达式的结果为true, 即, 非0值)期间重复执行程序块. 条件表达式的结果始终为真时无线重复执行.

1) WHILE … ENDWL必须要配对使用.
2) <条件表达式>中含有表达式时, 其结果需要为逻辑值 (true: 1, false: 0).
3) <条件表达式> 输入时请注意运算优先级.

Examples

Example 1	WHILE 1 ENDWL	// WHILE ~ ENDWL 程序块无限执行
Example 2	WHILE IN0 == 1 ENDWL	// IN0为 1时重复执行程序块
Example 3	WHILE IN1 == 1 && IN0 == 1 ENDWL	// IN0和 IN1为 1时重复执行程序块
Example 4	WHILE MVR > 10 ENDWL	// MVR 变量值大于10时重复执行程序块

Related terms

FOR … TO … [BY] … NEXT

5.6 IF …THEN … ENDIF (条件分段命令)

Usage

根据条件表达式的运算结果执行程序块.

Syntax

```
IF <条件表达式> THEN
    …
[ELSE IF <条件表达式> THEN]
    …
[ELSE]
    …
ENDIF
```

Description

根据条件表达式的运算结果决定是否执行程序块.

1) IF … THEN … ENDIF必须一起使用.
2) ELSEIF, ELSE用户可以选择性使用.
3) 条件表达式的运算结果为true(0 以外值)时执行 THEN 以下语句, false(0)时执行ELSE 以下语句.
4) IF 语句内无法再使用IF语句. 此时，请注意IF和ENDIF需要逻辑上成对.

Examples

Example 1

```
IF IN0 == 1 THEN                    // IN0为 1时
    JMOV P0                         // 从当前位置JMOV到P0
ENDIF                               // IF语句结束
```

Example 2

```
INT BB                              // 声明整数型变量
IF I0 == 1 THEN                     // I0值为 1时
    STOP                            // Robot停止
ELSEIF I0 == 2 THEN                 // I0值为 2时
    EXIT                            // 程序结束
ELSE                                // 不满足任何条件式时
    BB = IN0                        // IN0的值保存到整数型变量
    IF BB == 1 THEN                 // BB 变量为1时
        DLAY 10                     // 等待10ms
        JMOV P0                     // 从当前位置JMOV到P0
    ENDIF                           // IF语句结束
ENDIF                               // IF语句结束
```

5.7 LABL … GOTO（无条件转移命令）

Usage

指定跳转位置，移动到相应转移位置的命令.

Syntax

```
LABL <无条件转移名>
    …
GOTO <无条件转移名>
```

Description

GOTO 命令将无条件分段，仅在相同块内使用.
例如，MAIN 块内无法跳转到FUNC 块，FUNC 块内无法跳转到其它FUNC 块.
IF 及 FOR，WHILE 块使用时，可以从内部跳转到外部，反之不可.

1) <为条件转移名>可以使用英文字母和数字，声明时第一位不可以使用数字. 此外，无法使用预声明的全局变量. 同一个文件中无法重复声明相同无条件转移名.

Examples

Example 1

```
MAIN                        // 主函数声明
    SVON                    // 伺服 ON
    VEL 500                 // 速度变更
LABL REPEAT                 // 声明跳转点
    JMOV P0                 // 从当前位置JMOV到P0
    JMOV P1                 // 当前位置JMOV到P1
    GOTO REPEAT             // 移动到跳转点
EOP                         // 主函数结束
```

Examp12

```
FUNC VOID TEST()            // 函数声明
    SVON                    // 伺服 ON
    OUT0 = 1                // OUT0赋值为 1
LABL REPEAT                 //声明跳转点
    JMOV P0                 // 从当前位置JMOV到P0
    JMOV P1                 // 当前位置JMOV到 P1
    OUT0 = 0                // OUT0赋值为0
    GOTO REPEAT             //移动到跳转点
END                         // 函数结束
```

5.8 CONTINUE

Usage

重复语句内不执行后续语句，立即执行重复语句后面部分及利用条件式分段时使用.

Syntax

```
WHILE <条件表达式>
    IF <条件表达式> THEN
        CONTINUE
    ENDIF
ENDWL
```

Description

遇到CONTINUE语句时移动到重复语句的第一行.

Examples

```
WHILE 1                                 // 无限重复
    SVON                                // 伺服 ON
    JMOV P1                             // 从当前位置JMOV到P1
    IF IN(1) == 1  THEN                 // IN1值为 1时
        CONTINUE                        // 移动到第一句 (→ WHILE 1)
    ENDIF                               // 条件式结束
    JMOV P0                             // 从当前位置JMOV到P0
ENDWL                                   // WHILE语句结束
```

Related terms

FOR … TO … [BY] … NEXT, WHILE … ENDWL, BREAK

5.9 BREAK

Usage

重复语句内，发生特定条件时退出重复语句使用.

Syntax

```
WHILE <条件表达式>
    IF <条件表达式> THEN
        BREAK
    ENDIF
ENDWL
```

Description

遇到BREAK语句退出重复语句.

Examples

```
WHILE 1                                 // 无限重复
    SVON                                // 伺服 ON
    JMOV P1                             // 从当前位置JMOV到P1
    IF IN(1) == 1  THEN                 // IN1的值为1时
        BREAK                           // 退出重复语句
    ENDIF                               // 条件式结束
    JMOV P0                             // 从当前位置JMOV到P0
ENDWL                                   // WHILE语句结束
```

Related terms

FOR … TO … [BY] … NEXT, WHILE … ENDWL, CONTINUE

5.10 JCALL (程序调用)

Usage

调用JOB 文件立即执行.

Syntax

JCALL <JOB 文件名>

Description

调用的JOB 程序遇到EOP 命令，回到调用程序的JCALL 命令下一句.
与INCLUDE命令相同，可以使用被调用文件内部全局变量及函数.
制作程序时需要考虑JCALL不会被递归调用.

1) <JOB 文件名>只包含除文件结束符“.” 和扩展名 “JOB”的文件名.

Examples

```
MAIN                                // 主函数声明
    JCALL TEST                      // TEST.JOB 程序调用并立即执行
EOP                                 // 主函数结束
```

Related terms

INCLUDE

5.11 STOP (Robot停止)

Usage

停止Robot动作.

Syntax

STOP

Description

停止移动中Robot, 处理下一个STEP的命令. 即, 停止执行中Motion后继续执行下一个STEP.

Examples

```
MAIN                                    // 主函数声明
    SVON                                // 伺服 ON
    WITH                                // 执行WITH 语句
        JMOV P0                         // 从当前位置JMOV到P0
        WHILE MVR < 90                  // MVR 值小于90 时重复执行
            IF IN0 == 1 THEN            // IN0为 1时
                GOTO AA                 // 跳转到跳转点AA
            ENDIF                       // IF语句结束
        ENDWL                           // WHILE 语句结束
LABL AA                                 // 声明无条件转移点AA
        STOP                            // Robot停止
    ENDWT                               // WITH语句结束
EOP                                     // 主函数结束
```

Related terms

EXIT

5.12 EXIT (JOB 程序停止)

Usage

停止JOB 程序执行.

Syntax

EXIT

Description

EXIT 语句将停止执行JOB 程序. 此时, 控制器发出 “EXIT Instruction” Alarm.

Examples

```
MAIN                                    // 主函数声明
    SVON                                // 伺服 ON
    WITH                                // 执行WITH 语句
        JMOV P0                         // 从当前位置JMOV到P0
        WHILE MVR < 90                  // MVR 值小于90 时重复执行
            IF INO == 1 THEN            // INO为 1时
                GOTO AA                 // 跳转到跳转点AA
            ENDIF                       // IF语句结束
        ENDWL                           // WHILE 语句结束
LABL AA                                 // 声明无条件转移点AA
        EXIT                            // 程序停止及发出EXIT Alarm
    ENDWT                               // WITH语句结束
EOP                                     // 主函数结束
```

Related terms

STOP

6. Robot动作

6.1 SVON (伺服 ON)

Usage

开启电机电源.

Syntax

SVON[(<Robot编号>)]

Description

开启JOB 程序中分配的Robot所有轴的伺服电机电源.
伺服电机电源未开启的状态下使用移动命令将发出Alarm.

1) 输入<Robot编号>保存 JOB 程序时，可以开启非选择的其它Robot伺服电机电源.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.

Examples

Example 1

```
MAIN                                // 主函数开始
    SVON                            // 伺服 ON
    JMOV P0                         // 从当前位置JMOV到P0
    SVOF                            // 伺服 OFF
EOP                                 // 主函数结束
```

Example 2

```
MAIN                                // 主函数开始
    INT STATE                       // 声明整数型变量
    STATE = RSTATE(2,6)             // 保存Robot2号伺服状态
    IF STATE == 0 THEN              // 伺服OFF时
        SVON(2)                     // Robot2号伺服 ON
    ENDIF                           // 条件语句结束
EOP                                 // 主函数结束
```

Related terms

SVOF

6.2 SVOF (伺服 OFF)

Usage

关闭伺服电机电源.

Syntax

SVOF[(<Robot编号>)]

Description

关闭分配到JOB 程序的Robot所有轴的伺服电机电源.

1) 输入<Robot编号>保存 JOB 程序时，可以关闭非选择的其它Robot的伺服电机电源.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.

Examples

Example 1

```
MAIN                    // 主函数开始
    SVON                // 伺服 ON
    JMOV P0             // 从当前位置JMOV到P0
    SVOF                // 伺服 OFF
EOP                     // 主函数结束
```

Example 2

```
MAIN                    // 主函数开始
    INT STATE           // 声明整数型变量
    STATE = RSTATE(2,6) // 保存Robot2号伺服状态
    IF STATE == 1 THEN  // 伺服ON时
        SVOF(2)         // Robot2号伺服 OFF
    ENDIF               // 条件语句结束
EOP                     // 主函数结束
```

Related terms

SVON

6.3 JMOV (PTP 移动)

Usage

从当前位置移动到目标点.

Syntax

JMOV <目标位置> [, FOS = <值>, VEL = <值>, ACC = <值>, DEC = <值>, T = <值>, U = <值>]

Description

点到点移动的PTP(Point to Point) 移动.
移动的路径与Robot姿势相关, 其路径不仅限于直线动作.

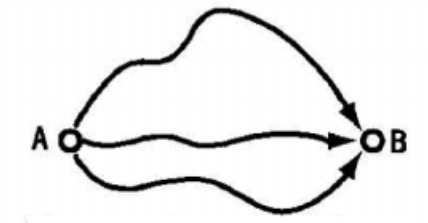


Figure 4-4 PTP 移动例子

使用命令时可以追加输入FOS, VEL, ACC, DEC, T, U的附加项目.
指定如FOS, VEL的关键字代入值, 可以使Motion命令有所变化. 各关键字与顺序无关, 也可以部分使用.

1) <目标位置>可以使用所有位置型变量. 使用JMOV 命令时, 输入的位置型变量转换为Joint坐标系位置型变量移动.
2) FOS(Factor of smooth path)为不在中间经由点停止, 直接通过连续到达下一个目标位置的功能. 移动时参考下一个目标点的距离比 (%)更改轨迹. 距离比是整体移动距离的百分比 (%).
3) VEL代表轴移动速度, 以千分比 (%)表示. 如下公式计算移动速度. 额定速度及初始速度在MOTION 参数中设置.
   - 轴移动速度 = 初始速度 X 额定速度 X 千分比值 X 0.001
4) ACC代表轴加速时间, 以百分比 (%)表示. 如下公式计算加速时间.
   额定加速时间在 MOTION 参数中设置.
   - 加速时间 = 额定加速时间 X 百分比值 X 0.01
5) DEC代表轴减速时间, 以百分比 (%)表示. 如下公式计算减速时间
   额定减速时间在MOTION 参数中设置.
   - 减速时间 = 额定减速时间 X 百分比值 X 0.01
6) T表示Tool编号, U表示用户坐标系编号. 执行Motion时选择使用的Tool或选择坐标系时使用.

Arguments

<目标位置>	Data type: <所有位置型变量>
FOS	Factor of smooth path Data type: <整数型> FOS的范围为 0 ~ 50%.

VEL	Velocity Data type: <整数型> VEL的范围为 1 ~ 1,000%.
ACC	Acceleration Data type: <整数型> ACC的范围为 1 ~ 400%.
DEC	Deceleration Data type: <整数型> DEC的范围为 1 ~ 400%.
T	Tool number Data type: <整数型> T的范围为 0 ~ 16. 与TOOL 命令作用相同.
U	User number Data type: <整数型> U的范围为 0 ~ 16. 与USER 命令作用相同.

Examples

Example 1

```
MAIN                                                    // 主函数声明
    POS AP                                              // 位置型变量声明
    AP = <400.0, 50.0, 10.4, 10.0, 0, 0>                // 变量初始化
    SVON                                                // 伺服 ON
    JMOV GP0                                            // JMOV 到GP0
    JMOV AP                                             // JMOV 到AP
EOP                                                     // 主函数结束
```

Example 2

```
MAIN                                                    // 主函数声明
    XPOS XP                                             // 位置型变量声明
    XP = <250,250,350,-30,30,60,ARM = 0,T = 2>          // 变量初始化
    SVON                                                // 伺服 ON
    JMOV XP                                             // JMOV 到XP
EOP                                                     // 主函数结束
```

Example 3

```
MAIN                                                    // 主函数声明
    SVON                                                // 伺服 ON
    JMOV P0, FOS = 30, VEL = 100, ACC = 200, DEC = 100  // 参照条件JMOV 到P0
    JMOV P1, DEC = 150, VEL = 200, ACC = 150            // 参照条件JMOV 到P1
EOP                                                     // 主函数结束
```

Related terms

LMOV, FOS, VEL, ACC, DEC, TOOL, USER

6.4 LMOV (直线插值移动)

Usage

从当前位置到目标点进行直线插值移动.

Syntax

LMOV <目标位置> [, FOS = <值>, VEL = <值>, ACC = <值>, DEC = <值>, T = <值>, U = <值>]

Description

进行插值移动使从起始位置到目标位置的路径成直线.

Figure 4-5 直线插值移动例子

使用命令时可以追加使用FOS, VEL, ACC, DEC, T, U附加项.
指定如FOS, VEL的关键字代入值，可以使Motion命令有所变化. 各关键字与顺序无关，也可以部分使用.

1) <目标位置>可以使用所有位置型变量. 使用LMOV 命令时，输入的位置型变量转换为Base坐标系位置型变量移动.
2) FOS(Factor of smooth path) 为不在中间经由点停止，直接通过连续到达下一个目标位置的功能. 移动时参考下一个目标点的距离比 (%)更改轨迹. 距离比是整体移动距离的百分比 (%).
3) VEL代表轴移动速度, 以千分比 (%)表示. 如下公式计算移动速度. 额定速度及初始速度在MOTION 参数中设置.
   - 轴移动速度 = 初始速度 X 额定速度 X 千分比值 X 0.001
4) ACC代表轴加速时间, 以百分比 (%)表示. 如下公式计算加速时间.
   额定加速时间在 MOTION 参数中设置.
   - 加速时间 = 额定加速时间 X 百分比值 X 0.01
5) DEC代表轴减速时间, 以百分比 (%)表示. 如下公式计算减速时间
   额定减速时间在MOTION 参数中设置.
   - 减速时间 = 额定减速时间 X 百分比值 X 0.01
6) T表示Tool编号, U表示用户坐标系编号. 执行Motion时选择使用的Tool或选择坐标系时使用.

Arguments

<目标位置>	Data type: <所有位置型变量>
FOS	Factor of smooth path Data type: <整数型> FOS的范围为 0 ~ 50%.
VEL	Velocity Data type: <整数型> VEL的范围为 1 ~ 1,000%.
ACC	Acceleration

Data type: <整数型>
ACC的范围为 1 ~ 400%.

DEC — Deceleration
Data type: <整数型>
DEC的范围为 1 ~ 400%.

T — Tool number
Data type: <整数型>
T的范围为 0 ~ 16. 与TOOL 命令作用相同.

U — User number
Data type: <整数型>
U的范围为 0 ~ 16. 与USER 命令作用相同.

Examples

Example 1

```
MAIN                                                    // 主函数声明
    XPOS XP                                             // 位置型变量声明
    XP = <250,250,350,-30,30,60,ARM = 0,T = 2>          // 变量初始化
    SVON                                                // 伺服 ON
    LMOV XP                                             // LMOV 到XP
EOP                                                     // 主函数结束
```

Example 2

```
MAIN                                                    // 主函数声明
    POS AP                                              // 位置型变量声明
    AP = <30.0, -60.0, 10.4, 10.0, 0, 0>                // 变量初始化
    SVON                                                // 伺服 ON
    LMOV AP, T = 1                                      // 使用T1号LMOV到AP
EOP                                                     // 主函数结束
```

Example 3

```
MAIN                                                    // 主函数声明
    SVON                                                // 伺服 ON
    LMOV P0, FOS = 30, VEL = 80, ACC = 200, DEC = 100, U = 1   // 参照条件LMOV 到P0
    LMOV P1, DEC = 150, VEL = 200, ACC = 150, T = 1     // 参照条件LMOV 到P1
EOP                                                     // 主函数结束
```

Related terms

JMOV, FOS, VEL, ACC, DEC, TOOL, USER

1) 根据Robot的位置和姿势有可能无法动作. 此时, 会发生“[1219] Range Over error”, 或“[1415] ik position error” 等Alarm. 使用Robot动作条件命令时请注意.
2) 请留意作业Point Teaching.

6.5 CMOV (圆形插值移动)

Usage

从当前位置连接经由点1, 2的同时画圆移动.

Syntax

CMOV <经由点1> <经由点2> [, VEL = <值>, ACC = <值>, DEC = <值>, T = <值>, U = <值>]

Description

画出到达动作目标位置的圆进行补插值. 圆形插值动作时需要供3个位置坐标 (起始点, 经由点1, 经由点2).
使用CMOV时建议如 Figure 4-6使用JMOV移动后再使用CMOV.

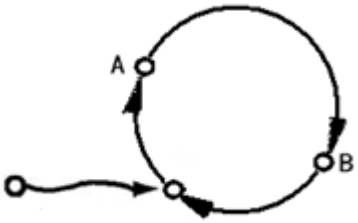


Figure 4-6 圆形插值例子

使用命令时可以追加使用 VEL, ACC, DEC, T, U附加项.
指定如VEL的关键字代入值, 可以使Motion命令有所变化. 各关键字与顺序无关, 也可以部分使用.

1) <经由点1>和 <经由点2>可以使用所有位置型变量. 使用CMOV命令时, 输入的位置型变量转换为Base坐标系位置型变量移动.
2) VEL代表轴移动速度, 以千分比 (%)表示. 如下公式计算移动速度. 额定速度及初始速度在MOTION 参数中设置.
   - 轴移动速度 = 初始速度 X 额定速度 X 千分比值 X 0.001
3) ACC代表轴加速时间, 以百分比 (%)表示. 如下公式计算加速时间. 额定加速时间在 MOTION 参数中设置
   - 加速时间 = 额定加速时间 X 百分比值 X 0.01
4) DEC代表轴减速时间, 以百分比 (%)表示. 如下公式计算减速时间. 额定减速时间在MOTION 参数中设置.
   - 减速时间 = 额定减速时间 X 百分比值 X 0.01
5) T表示Tool编号, U表示用户坐标系编号. 执行Motion时选择使用的Tool或选择坐标系时使用.

Arguments

<经由点1> <经由点2>	Data type: <所有位置型变量>
VEL	Velocity Data type: <整数型> VEL的范围为 1 ~ 1,000%.
ACC	Acceleration Data type: <整数型>

ACC的范围为 1 ~ 400%.

DEC
Deceleration
Data type: <整数型>
DEC的范围为 1 ~ 400%.

T
Tool number
Data type: <整数型>
T的范围为 0 ~ 16. 与TOOL 命令作用相同.

U
User number
Data type: <整数型>
U的范围为 0 ~ 16. 与USER 命令作用相同.

Examples

Example 1

```
MAIN                                                    // 主函数声明
    SVON                                                // 伺服 ON
    JMOV P1                                             // JMOV 到P1
    CMOV P2 P3                                          // 经由P2, P3的圆形插值移动
EOP                                                     // 主函数结束
```

Example 2

```
MAIN                                                    // 主函数声明
    SVON                                                // 伺服 ON
    JMOV P0                                             // JMOV到P0
    CMOV P1 P2, DEC = 50, VEL = 100, ACC = 50, T = 1    // 经由P1, P2的圆形插值移动
EOP                                                     // 主函数结束
```

Related terms

AMOV, VEL, ACC, DEC, TOOL, USER

1) 根据Robot的位置和姿势有可能无法动作. 此时, 会发生“[1219] Range Over error”, 或“[1415] ik position error” 等Alarm. 使用Robot动作条件命令时请注意.
2) 请留意作业Point Teaching.

6.6 AMOV (圆弧插值移动)

Usage

从当前位置连接经由点 1, 2的同时画圆弧移动.

Syntax

AMOV <经由点1> <经由点2> [, VEL = <值>, ACC = <值>, DEC = <值>, T = <值>, U = <值>]

Description

画出到达动作目标位置的圆弧进行补插值. 圆弧形插值动作时需要供3个位置坐标 (起始点, 经由点1, 经由点2)
使用AMOV时建议如 Figure 4-7使用 JMOV移动后再使用 AMOV.

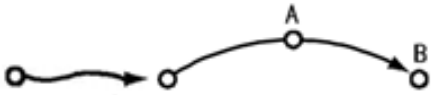


Figure 4-7 圆弧插值例子

使用命令时可以追加使用 VEL, ACC, DEC, T, U附加项.
指定如VEL的关键字代入值，可以使Motion命令有所变化. 各关键字与顺序无关，也可以部分使用.

1) <经由点1>和 <经由点2>可以使用所有位置型变量. 使用AMOV命令时，输入的位置型变量转换为Base坐标系位置型变量移动.
2) VEL代表轴移动速度, 以千分比 (%)表示. 如下公式计算移动速度. 额定速度及初始速度在MOTION 参数中设置.
   - 轴移动速度 = 初始速度 X 额定速度 X 千分比值 X 0.001
3) ACC代表轴加速时间, 以百分比 (%)表示. 如下公式计算加速时间. 额定加速时间在 MOTION 参数中设置.
   - 加速时间 = 额定加速时间 X 百分比值 X 0.01
4) DEC代表轴减速时间, 以百分比 (%)表示. 如下公式计算减速时间. 额定减速时间在MOTION 参数中设置.
   - 减速时间 = 额定减速时间 X 百分比值 X 0.01
5) T表示Tool编号, U表示用户坐标系编号. 执行Motion时选择使用的Tool或选择坐标系时使用.

Arguments

<经由点1> <经由点2>	Data type: <所有位置型变量>
VEL	Velocity Data type: <整数型> VEL的范围为 1 ~ 1,000%.
ACC	Acceleration Data type: <整数型> ACC的范围为 1 ~ 400%.
DEC	Deceleration

Data type: <整数型>
DEC的范围为 1 ~ 400%.

T Tool number
Data type: <整数型>
T的范围为 0 ~ 16. 与TOOL 命令作用相同.

U User number
Data type: <整数型>
U的范围为 0 ~ 16. 与USER 命令作用相同.

Examples

Example 1

```
MAIN                                                    // 主函数声明
    SVON                                                // 伺服 ON
    JMOV P1                                             // JMOV到P1
    AMOV P2 P3                                          // 经由P2到达P3的圆弧插值移动
EOP                                                     // 主函数结束
```

Example 2

```
MAIN                                                    // 主函数声明
    SVON                                                // 伺服 ON
    JMOV P0                                             // JMOV到P0
    AMOV P1 P2, DEC = 50, VEL = 80, ACC = 50, T = 1     // 经由P1到达P2的圆弧插值移动
EOP                                                     // 主函数结束
```

Related terms

CMOV, VEL, ACC, DEC, TOOL, USER

1) 根据Robot的位置和姿势有可能无法动作. 此时, 会发生“[1219] Range Over error”, 或“[1415] ik position error” 等Alarm. 使用Robot动作条件命令时请注意.
2) 请留意作业Point Teaching.

7. Robot动作条件

7.1 WITH … ENDWT (同时处理命令)

Usage

Robot动作的同时处理程序块内执行语句.

Syntax

```
WITH
    …
ENDWT
```

Description

WITH语句内执行第一条移动命令的同时并行处理之后的语句.
WITH语句块内使用重复语句及条件式等，在移动完成前预先进行必要的作业.

Examples

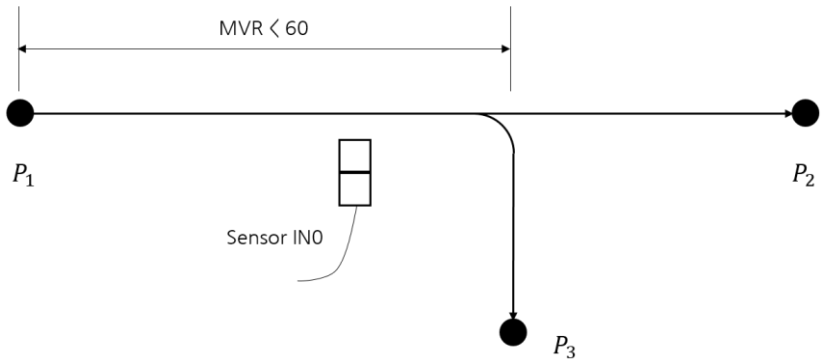


Figure 4-8 WITH语句使用例子

```
MAIN                                    // 主函数声明
    SVON                                // 伺服 ON
    JMOV P1                             // JMOV到P1
    WITH                                // 使用WITH语句, 同时处理
        JMOV P2                         // JMOV到P2
        WHILE MVR < 60                  // MVR小于 60 时重复执行
            IF IN0 == 1 THEN            // IN0为1时
                STOP                    // Robot停止
                GOTO AA                 // 移动到跳转点AA
            ENDIF                       // IF语句结束
        ENDWL                           // WHILE 语句结束
    ENDWT                               // WITH语句结束
LABL AA                                 // 跳转点AA 声明
    JMOV P3                             // JMOV到P3
EOP                                     // 主函数结束
```

Related terms

MVR

1) 执行WITH语句过程中如果用跳转命令跳出WITH … ENDWT 语句且停止Robot时，请在跳转命令前停止Robot.

7.2 VEL (轴移动速度设置)

Usage

设置轴移动速度的千分比 (%).

Syntax

VEL <速递比>

Description

对Robot移动速度的命令，变更整体程序移动命令的速度.
额定速度及初始速度在MOTION参数中设置..

- 轴移动速度 = 初始速度 X 额定速度 X 千分比 X 0.001

移动命令中变更速度时，以命令中插入的速度值为优先.

1) <速度比>表示轴移动速度的比例，以千分比 (%) 为单位设置.

Arguments

<速度比>	Data type: <整数型> <速度比>的范围为 1 ~ 1,000%.

Examples

```
MAIN                    // 执行主函数
    VEL 1000            // 移动速度: 初始速度 X 额定速度 X 0.001 X 1,000
    JMOV P0             // JMOV到P0
    VEL 200             // 移动速度: 初始速度 X 额定速度 X 0.001 X 200
    JMOV P1             // JMOV到P1
EOP
```

Related terms

JMOV, LMOV, CMOV, AMOV, FOS, ACC, DEC

1) 需要超出器具允许的最大RPM时，可能发生噪音及破损.
   请务必确认器具上粘贴的标签后使用.
2) 仅适用于JOB 程序内，不会发生参数值的变更.

7.3 ACC (设置轴加速时间)

Usage

设置加速时间的百分比 (%).

Syntax

ACC <加速时间比>

Description

对Robot加速时间的命令，变更整体程序移动命令的加速时间. 额定加速时间在MOTION 参数中设置，调整百分比变更加速时间.

- 加速时间 = 额定加速时间 X 百分比 X 0.01

移动命令中变更加速时间时，以命令中插入的加速时间为优先.

1) <加速时间比>表示轴加速时间比例，以百分比 (%) 为单位设置

Arguments

<加速时间比>	Data type: <整数型> <加速时间比>的范围为 1 ~ 400%.

Examples

- 参数中额定加速时间设置为0.4秒为例

```
MAIN                    // 执行主函数
    ACC 200             // 加速时间: 额定加速时间(0.4s) X 0.01 X 200 = 0.8s
    JMOV P0             // JMOV到P0
    ACC 50              // 加速时间: 额定加速时间(0.4s) X 0.01 X 50 = 0.2s
    JMOV P1             // JMOV到P1
EOP
```

Related terms

JMOV, LMOV, CMOV, AMOV, FOS, VEL, DEC

1) 加速时间比设置在100%以下时可能发生机械部分噪音及振动，请注意使用.
2) 仅适用于JOB 程序内，不会发生参数值的变更.

7.4 DEC (轴减速时间设置)

Usage

设置减速时间的百分比(%).

Syntax

DEC <加速时间比>

Description

对Robot减速时间的命令，变更整体程序移动命令的减速时间. 额定减速时间在MOTION 参数中设置，调整百分比变更减速时间.

- 减速时间 = 额定减速时间 X 百分比 X 0.01

移动命令中变更减速时间时，以命令中插入的减速时间为优先.

1) <减速时间比>表示轴减速时间比例，以百分比 (%) 为单位设置

Arguments

<减速时间比>	Data type: <整数型> <减速时间比>的范围为 1 ~ 400%.

Examples

- 参数中额定减速时间设置为0.6秒为例

```
MAIN                    // 执行主函数
    DEC 200             // 减速时间: 额定减速时间(0.6s) X 0.01 X 200 = 1.2s
    JMOV P0             // JMOV到P0
    DEC 50              // 减速时间: 额定减速时间(0.6s) X 0.01 X 50 = 0.3s
    JMOV P1             // JMOV到P1
EOP
```

Related terms

JMOV, LMOV, CMOV, AMOV, FOS, VEL, ACC

1) 减速时间比设置在100%以下时可能发生机械部分噪音及振动，请注意使用.
2) 仅适用于JOB 程序内，不会发生参数值的变更.

7.5 FOS (Factor of smooth path 设置)

Usage

在轴线段到达目标点之前改变下一个目标点以此改变轨迹。

Syntax

FOS <距离比例>: 使用单一命令时
LMOV P0, FOS = <距离比例>: 使用移动命令时
JMOV P1, FOS = <距离比例>: 使用移动命令时

Description

开始执行 FOS 命令后，机器人未达到指定位置之前启动移动到下一个位置的命令，可以在不停止机器人连续移动机器人。
设置 FOS 命令，可在当前运作中的移动动作减速时间之内设置下一个移动命令开始时间点。
FOS命令中 JMOV和 JMOV 移动命令是可以连续进行移动的，
LMOV, AMOV, 以及 CMOV之间的移动命令切换也可以连续进行。
但是 JMOV和 LMOV(AMOV, CMOV)的移动坐标系不同，因此不能连续进行移动命令，只能当前移动命令到达之后执行下一个命令。

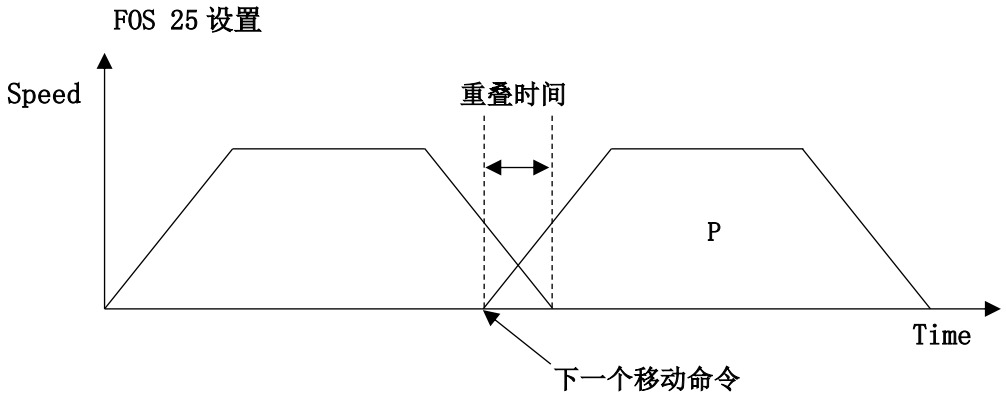
使用 FOS 命令时在相同移动距离内移动速度变更时连接区间的轨迹会改变，
速度加快则连接区间会加长，也需要确认根据速度的移动轨迹。

使用FOS 命令时不会产生 INPOS 信号。

当前 \ 下一个	JMOV	LMOV	AMOV	CMOV
JMOV	连续运动	不可连续运动	不可连续运动	不可连续运动
LMOV	不可连续运动	连续运动	连续运动	连续运动
AMOV	不可连续运动	连续运动	连续运动	连续运动
CMOV	不可连续运动	连续运动	连续运动	连续运动

1) <加减速时间比例>设置当前移动命令语减速时间比例(%)。

例子) 设置 FOS 25 时在减速时间进行到 50% 时开始下一个移动命令。



例子）到达 FOS 50 设置时减速时间的 0% 时间点时开始下一个移动命令

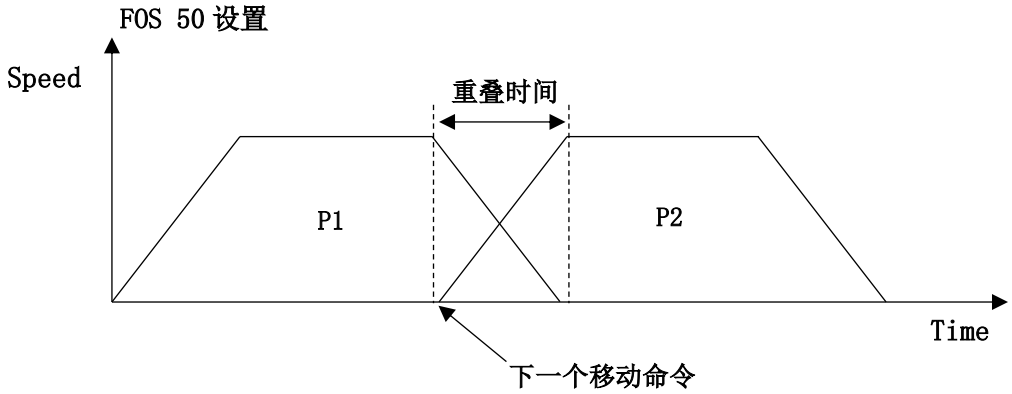


Figure 4-9 FOS 比例

Arguments

<距离比例>	Data type: <整数型> <减速时间比例>的范围为 0 ~ 50%. 设置 1 ~ 50 设置时可以激活 FOS 命令功能，设置为0时为非激活状态

Examples

Example 1

- P1 到达前 FOS 20%

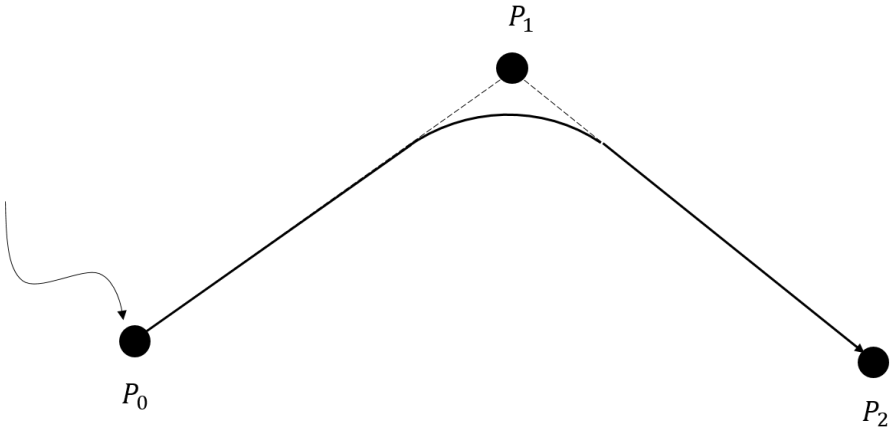


Figure 4-10 FOS 使用基本例子

```
MAIN                                    // 开始主函数
    SVON                                // 私服 ON
    JMOV P0                             // 到 P0为止 JMOV
    FOS 20                              // FOS 20% 为止
    JMOV P1                             // 到 P1 为止 JMOV
    FOS 0                               // FOS 0% 设置
    JMOV P2                             // 到 P2 为止 JMOV
EOP                                     // 结束主函数
```

Example 2

- P1移动到 P2 过程中输出OUT0 信号

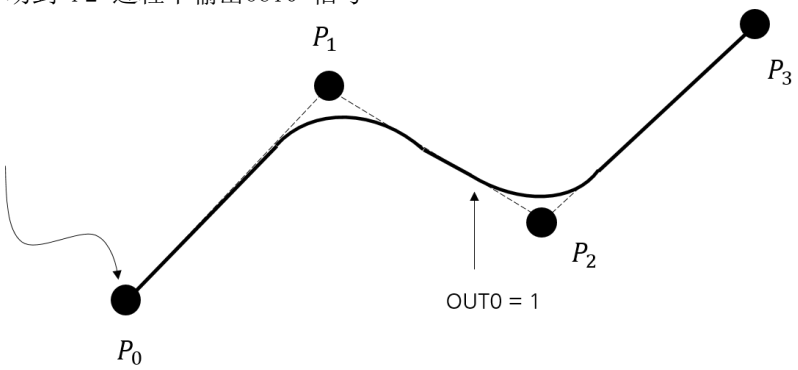


Figure 4-11 移动中信号输出案例

```
MAIN                                    // 开始主函数
    SVON                                // 私服 ON
    JMOV P0                             // 到 P0 为止 JMOV
    FOS 10                              // FOS 10% 设置
    LMOV P1                             // 到 P1 为止 LMOV
    LMOV P2                             // 到 P2 为止 LMOV
    OUT0 = 1                            // OUT0中输出 1
    FOS 0                               // FOS 0% 设置
    JMOV P3                             // 到P3 为止 JMOV
EOP                                     // 结束主函数
```

Example 3

- 从 P0移动到 P1 过程中确认 IN0 信号之后选择路径

Figure 4-12 移动过程中确认信号输入的例子

```
MAIN                                    // 开始主函数
    SVON                                // 私服 ON
    JMOV P0                             // 到 P0为止 JMOV
    FOS 10                              // FOS 10% 设置
    JMOV P1                             // 到P1为止 JMOV
    IF IN0 == 1 THEN                    // IN0为 1时
        JMOV P2                         // 到P2为止 JMOV
    ENDIF                               // 结束 IF语句
EOP                                     // 结束主函数
```

Related terms

JMOV, LMOV, CMOV, AMOV, VEL, ACC, DEC

1) 移动命令之间处移动命令语外有多条延迟命令(DLAY), 计算式以及序列计算等时在移动命令区间的速度会减慢。
2) 当前移动命令和下一个移动命令的加减速时间相同时连接区间的速度会减慢。
3) 使用 FOS 命令时在相同距离内移动速度如果被修改则连接区间的移动轨迹会改变。

7.6 DLAY（延迟时间设置）

Usage

指定时间期间暂停程序处理.

Syntax

DLAY <延迟时间>

Description

直到经过输入的<延迟时间>为止停止程序处理.

1) <延迟时间>的单位为1ms.

Arguments

<延迟时间>	Data type: <整数型>

Examples

```
MAIN                                    // 主函数开始
    SVON                                // 伺服 ON
    JMOV P0                             // JMOV到P0
    DLAY 1000                           // 延迟1秒
    JMOV P1                             // JMOV到P1
    DLAY 2000                           // 延迟2秒
EOP                                     // 主函数结束
```

7.7 OFFS (指定偏差)

Usage

目标点在指定值上加减后移动.

Syntax

OFFS <偏差值>

Description

执行OFFS 命令后使用移动命令，均移动增减偏差值的位置.
如Figure 4-13，从A到B之间虚线为原来的路径， 使用OFFS 命令可以将路径变为A到B的实线路径.

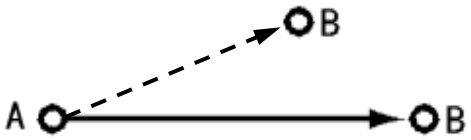


Figure 4-13 OFFS 命令例子

OFFS 命令仅在JOB 程序执行时有效， 对所有程序通用. 因此， 同样适用于通过JCALL, INCLUDE 命令使用的程序.

1) <偏差值>除位置型常数，可以使用所有位置型变量.
2) 参照输入的<偏差值>位置型变量坐标系，更新移动命令的最终目标位置.

Arguments

<偏差值>	Data type: <所有位置型变量>

Examples

Example 1

```
POS PDATA                          // 位置型变量声明
PDATA = <0,0,0,0,30,30>            // 位置型变量初始化
SVON                               // 伺服 ON
OFFS PDATA                         // OFFSET 设置
JMOV P0                            // JMOV到P0
```

Example 2

```
OFFS P0                            // OFFSET 设置
SVON                               // 伺服 ON
JMOV GP0                           // JMOV到GP0
OFFS GP0                           // OFFSET 重新设置
JMOV P0                            // JMOV到P0
```

Example 3

```
XPOS AP                                  // Base坐标系 位置型变量声明
UPOS BP                                  // 用户坐标系位置型变量声明
AP = <10,10,10,10,10,10,ARM = 0,T = 1>   // Base坐标系位置型变量初始化
BP = <5,10,15,0,0,0,ARM = 0,T = 1,U =    // 用户坐标系位置型变量初始化
1>                                       // 伺服 ON
SVON                                     // 利用Base坐标系位置型变量设置OFFSET
OFFS AP                                  // LMOV到GP0
LMOV GP0                                 // 利用用户坐标系位置型变量设置OFFSET
OFFS BP                                  // GJMOV到P1
JMOV GP1
```

Related terms

OFFSCLR

1) 即使Teaching值没有超出S/W 移动限定范围，在执行JOB程序时使用OFFS 命令时有可能发生 “[1219] Range Over error”.
2) 如果解除OFFS请使用OFFSCLR 命令.

7.8 OFFSCLR (解除偏差)

Usage

解除设置的偏差.

Syntax

OFFSCLR

Description

解除用户使用OFFS 命令设置的偏差值，变更为参照原位置值.
与OFFS 命令相同，适用于所有程序.

Examples

```
OFFS P0                 // OFFSET 设置
SVON                    // 伺服 ON
JMOV GP0                // JMOV到GP0
OFFSCLR                 // OFFSET 解除
JMOV P0                 // JMOV到P0
```

Related terms

OFFS

7.9 LIMT (轴移动范围限制)

Usage

限制各轴的移动范围.

Syntax

LIMT <(-)移动范围> <(+)移动范围>

Description

设置JOB 程序内的S/W 移动限制范围.
LIMT 命令仅在JOB 程序执行时有效, 所有程序通用. 因此, 同样适用于通过JCALL, INCLUDE 命令使用的程序.

1) (-)移动范围表示向(-)方向移动的最小位置.
2) (+)移动范围表示向(+)方向移动的最大位置.
3) 输入(-) 及 (+)移动范围时需要使用Joint坐标系位置型变量, 无法使用位置型常数.

Arguments

<(-)移动范围>	Data type: <Joint坐标系位置型变量>
<(+)移动范围>	Data type: <Joint坐标系位置型变量>

Examples

Example 1

```
LIMT P0 GP1                          // S/W 移动限制范围变更
SVON                                 // 伺服 ON
JMOV P1                              // JMOV到P1
```

Example 2

```
POS AP, BP                           // Joint坐标系位置型变量声明
AP = <-100,-100,-100,-100,-100,-100> // Joint坐标系位置型变量初始化
BP = <100,100,100,100,100,100>
LIMT AP BP                           // S/W 移动限制范围变更
SVON                                 // 伺服 ON
JMOV P0                              // JMOV到P0
```

1) 即使Teaching值不超过S/W 移动限制范围, 执行JOB 程序时使用 LIMT 命令有可能发生 "[1219] Range Over error".
2) 请注意(+)移动范围应该不小于(-)移动范围.

7.10 TRQ（冲突感知命令）

Usage

设置扭矩限制值冲突时发生Alarm的命令.

Syntax

TRQ <Robot编号> <轴编号> <扭矩限制值>

Description

Robot动作中设置指定的各轴扭矩限制，限制以上扭矩在参数TOL 时间以上动作时发生“[1434] Over Trq error” Alarm.

1) 输入<Robot编号>指定Robot.
2) 输入<轴编号>指定监控扭矩值的轴.
3) 输入<扭矩限制值>设置发生Alarm的限定值.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ 8.
<扭矩限制值>	Data type: <整数型> <扭矩限制值>的范围为 1 ~ 300.

Examples

Example 1

```
TRQ 1 1 50          // J1 轴的扭矩限制设置为 50%
TRQ 1 2 100         // J2 轴的扭矩限制设置为 100%
LMOV P0             // 从当前位置LMOV到P0
LMOV P1             // 从当前位置LMOV到P1
TRQ 1 1 300         // J1轴的扭矩限制设置为 300%
TRQ 1 2 300         // J2轴的扭矩限制设置为 300%
```

1) JOB START时 TRQ 值初始化为300.
2) 伺服 OFF 状态下不监控扭矩值.

7.11 GAIN (Gain组转换)

Usage

转换Gain组.

Syntax

GAIN <组编号>

Description

选择执行Motion时使用的Gain组.
Gain组包含位置循环Gain, 速度循环Gain, 速度循环积分时间常数, 扭矩命令过滤时间常数.

1) 输入<组编号>选择Gain组.

Arguments

<组编号>	Data type: <整数型> <组编号>的范围为 1 ~ 2.

Examples

Example 1

```
MAIN                    // 主函数声明
    SVON                // 伺服 ON
    GAIN 1              // 应用Gain组1
    JMOV P0             // Gain组1应用的状态下JMOV到P0
    GAIN2               // 应用Gain组2
    JMOV P1             // Gain组2应用的状态下JMOV到P1
EOP                     // 主函数结束
```

1) 请注意Gain切换时有等待时间及切换时间.

7.12 IPE (In-position 允许时间设置)

Usage

设置到达目标位置 (In-Position)为止的允许时间最大值.

Syntax

IPE <允许时间>

Description

设置到达目标位置值附近开始到目标位置允许的最大时间值.
允许时间值内需要移动到目标位置误差允许范围 (IPA)内.

1) <允许时间>的单位为 1ms.

Arguments

<允许时间>	Data type: <整数型> <允许时间>的范围为 0 ~ 5,000(ms).

Examples

Example 1

```
MAIN                    // 主函数声明
    SVON                // 伺服 ON
    IPE 100             // 设置In-Position 允许时间最大值
    JMOV P0             // JMOV到P0
    JMOV P1             // JMOV到P1
EOP                     // 主函数结束
```

1) 允许时间内无法到达目标位置时发生“[1426] Inposition error Alarm”, 请注意设置的值.
2) IPE设置为 0时, JOB驱动中不会确认In-Position 情况.

8. 输入输出控制

8.1 IN（数字比特输入）

Usage

以1比特为单位读取指定比特编号的值.

Syntax

IN<比特编号> = <0/1> 或 IN(<比特编号>) = <0/1>
<变量> = IN<比特编号> 或 <变量> = IN(<比特编号>)

Description

IN 命令提供2中使用方法.
第一种方法为在指定的比特中代入0 或 1 值，等待相应比特输入信号与代入的值相同. 第二种方法为读取指定比特编号的信号值保存到变量中.

1) <比特编号>使用括号输入或直接输入数字使用. 文字 IN和<编号>之间不可以有空白.
2) <比特编号>的范围为 0 ~ 19, 使用扩展板时最大可以输入83.

Return value

<变量> Data type: <整数型>
结果值返回 0(OFF) 或1(ON)值.

Arguments

<0/1> Data type: <整数型>
只允许输入0(OFF) 或1(ON).

<比特编号> Data type: <整数型>

- 一般输入: IN0 ~ IN19
- 扩展1 输入: IN20 ~ IN51
- 扩展2 输入: IN52 ~ IN83

Examples

Example 1

```
SVON                    // 伺服 ON
JMOV P0                 // JMOV到P0
IN0 = 1                 // 等待0号输入信号变为1
JMOV P1                 // JMOV到P1
```

Example 2

```
INT STATE, NUM          // 声明整数型变量
NUM = 0                 // 整数型 变量初始化
STATE = IN0             // 保存IN0 信号值
PRINT(0,STATE)          // 保存到监控日志0号索引中
STATE = IN(0)           // 保存IN0 信号值
PRINT(1,STATE)          // 保存到监控日志1号索引中
STATE = IN(NUM)         // 参照NUM 变量值保存IN 信号
PRINT(2,STATE)          // 保存到监控日志2号索引中
```

Related terms

PIN

1) 等待信号值时请注意使用.

8.2 OUT (数字比特输出)

Usage

输出指定的比特编号中的值.

Syntax

OUT<比特编号> = <0/1> 或OUT(<比特编号>) = <0/1>
<变量> = OUT<比特编号> 或<变量> = OUT(<比特编号>)
OUT<比特编号> = <0/1>[<脉冲幅度>]
OUT<比特编号> = <0/1>[<脉冲幅度> <脉冲周期>]

Description

OUT 命令提供4种使用方法.
第一种方法为在指定比特编号中输出 0 或 1 值.
第二种方法为读取向指定比特编号输出的值保存到变量中.
第三种方法为向指定比特编号输出脉冲信号.
最后方法为向指定比特编号输出脉冲周期波形.

1) <比特编号>使用括号输入或直接输入数字使用. 文字OUT和 <编号>之间不可以有空白.
2) <比特编号>的范围为 0 ~ 19, 使用扩展板时最大可以输入83.
3) 追加输入<脉冲幅度>时, 输出输入时间的脉冲信号后回到原来值.
4) 输入<脉冲周期>时, 输出以输入时间为周期的脉冲周期波形. <脉冲周期>始终与 <脉冲幅度>同时输入, 一定要输入大于 <脉冲幅度>的值.

Return value

<变量> Data type: <整数型>
返回结果值0(OFF) 或1(ON)值.

Arguments

<0/1> Data type: <整数型>
只允许输入0(OFF) 或1(ON).

<比特编号> Data type: <整数型>
- 一般输入: IN0 ~ IN19
- 扩展1 输入: IN20 ~ IN51
- 扩展2 输入: IN52 ~ IN83

<脉冲幅度> Data type: <整数型>
单位为 1ms 但忽略 2ms 以下值, 因此请输入2的倍数.

<脉冲周期> Data type: <整数型>
单位为 1ms 但忽略 2ms 以下值, 因此请输入2的倍数.
脉冲周期一定要大于脉冲幅度.

Examples

Example 1	OUT(0) = 1
	DLAY 1000
	OUT0 = 0
		// 向0号输出信号输出1
		// 等待1秒
		// 向0号输出信号输出0值

Example 2	INT STATE
	STATE = OUT1
	PRINT(0,STATE)
	STATE = OUT(1)
	PRINT(1,STATE)
	STATE = OUT(IO)
	PRINT(2,STATE)
		// 声明整数型变量
		// 保存OUT1 信号值
		// 保存到监控日志0号索引中
		// 保存OUT1 信号值
		// 保存到监控日志1号索引中
		// 参照全局变量 IO号值保存 OUT 信号
		// 保存到监控日志2号索引中

Example 3	OUT0 = 0 OUT(0) = 1 100	// 向0号输出信号输出0值 // 向0号输出信号输出0.1秒长度的脉冲
Example 4	OUT1 = 0 OUT1 = 1 1000 5000	// 向1号输出信号输出0值 // 向1号输出信号输出5秒为周期1秒脉冲幅度的波形

Related terms

POUT

1) 使用OUT命令生成脉冲信号或脉冲周期波形时，输入的<脉冲幅度> 及 <脉冲周期>值的单位为1ms. 但， 2ms 以下值被忽略，因此请输入2倍数的值.
   - OUT0 = 1 1 → 无脉冲输出，继续维持输出值
   - OUT0 = 1 15 → 输出14ms 长度的脉冲
   - OUT0 = 1 10 15 → 输出以14ms 为周期10ms脉冲幅度的波形
   - OUT0 = 1 15 29 → 输出以28ms 为周期14ms脉冲幅度的波形

8.3 PIN（数字端口输入）

Usage

读取端口为单位的指定端口编号的值.

Syntax

<变量> = PIN<端口编号> 或<变量> = PIN(<端口编号>)

Description

一次性读取指定端口编号16比特数据值.

1) <端口编号>使用括号输入或直接输入数字使用. 文字PIN和 <编号>之间不可以有空白.
2) <端口编号>的范围为 0 ~ 1, 使用扩展板时最大可以输入5.

Return value

<变量> Data type: <整数型>
结果值返回16比特整数值. 数据值从右侧增加到左侧.

比特 / 整数值	F E D C	B A 9 8	7 6 5 4	3 2 1 0
31242	0 1 1 1	1 0 1 0	0 0 0 0	1 0 1 0
-63	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 1
0HF419	1 1 1 1	0 1 0 0	0 0 0 1	1 0 0 1
0B11110011	0 0 0 0	0 0 0 0	1 1 1 1	0 0 1 1

Arguments

<端口编号> Data type: <整数型>

- 一般输入: PIN0 ~ 1
  (PIN0: IN0 ~ IN15, PIN1: IN16 ~ 19)
- 扩展1 输入: PIN0 ~ 3
  (PIN0: IN0 ~ IN15, PIN1: IN16 ~ 31, PIN2: IN32 ~ 47, PIN3: IN48 ~ 51)
- 扩展2 输入: PIN0 ~ 5
  (PIN0: IN0 ~ IN15, PIN1: IN16 ~ 31, PIN2: IN32 ~ 47, PIN3: IN48 ~ 63, PIN4: 64 ~ 79, PIN5: 80 ~ 83)

Examples

Example 1

```
INT STATE                    // 声明整数型变量
STATE = PIN0                 // 保存PIN0信号值
PRINT(0,STATE)               // 保存到监控日志0号索引中
STATE = PIN(0)               // 保存PIN0信号值
PRINT(1,STATE)               // 保存到监控日志1号索引中
```

Example 2

- 位运算符应用例子

```
DEFSTR STR                          // 字符串变量声明
INT AA, RES                         // 声明整数型变量
AA = PIN0                           // 保存PIN0信号值
RES = AA & 0H00F0                   // 第4 ~7值位AND 运算
STR = BINS(RES)                     // 保存为二进制字符串
PRINT(0,STR)                        // 二进制字符串保存到监控日志0号索引中
RES = PIN0 | 0H000F                 // 第0 ~3值位 OR 运算
STR = BINS(RES)                     // 保存为二进制字符串
PRINT(1,STR)                        // 二进制字符串保存到监控日志1号索引中
RES = 0B11110000 ^ PIN0             // 第4 ~7值位 XOR 运算
STR = BINS(RES)                     // 保存为二进制字符串
PRINT(2,STR)                        // 二进制字符串保存到监控日志2号索引中
```

Related terms

IN

8.4 POUT (数字端口输出)

Usage

向指定端口编号输出指定值.

Syntax

POUT<端口编号> = <值> 或POUT(<端口编号>) = <值>
<变量> = POUT<端口编号> 或 <变量> = POUT(<端口编号>)

Description

POUT 命令提供2种使用方法.
第一种方法输出指定值.
第二种方法读取向指定端口编号输出的信号值保存到变量中.

1) <端口编号>使用括号输入或直接输入数字使用. 文字 POUT和 <编号>之间不可以有空白.
2) <端口编号>的范围为 0 ~ 1, 使用扩展板时最大可以输入5.

Return value

<变量> Data type: <整数型>
结果值返回16比特整数值. 数据值从右侧增加到左侧.

比特 / 整数值	F E D C	B A 9 8	7 6 5 4	3 2 1 0
31242	0 1 1 1	1 0 1 0	0 0 0 0	1 0 1 0
-63	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 1
0HF419	1 1 1 1	0 1 0 0	0 0 0 1	1 0 0 1
0B11110011	0 0 0 0	0 0 0 0	1 1 1 1	0 0 1 1

Arguments

<值> Data type: <整数型>
可输入的值范围为 16比特-32,768 ~ 32,767.

<端口编号> Data type: <整数型>

- 一般输入: POUT0 ~ 1
  (POUT0: OUT0 ~ OUT15, POUT1: OUT16 ~ OUT19)
- 扩展1 输入: POUT0 ~ 3
  (POUT0: OUT0 ~ OUT15, POUT1: OUT16 ~ OUT31, POUT2: OUT32 ~ OUT47, POUT3: OUT48 ~ OUT51)
- 扩展2 输入: POUT0 ~ 5
  (POUT0: OUT0 ~ OUT15, POUT1: OUT16 ~ OUT31, POUT2: OUT32 ~ OUT47, POUT3: OUT48 ~ OUT63, POUT4: OUT64 ~ OUT79, POUT5: OUT80 ~ OUT83)

Examples

Example 1	POUT(0) = 6245	// 向0号端口输出值
	DLAY 1000	// 等待1秒
	POUT0 = 0	// 向0号端口输出0值
Example 2	INT STATE	// 声明整数型变量
	STATE = POUT1	// 保存输出的POUT1 信号值
	PRINT(0,STATE)	// 保存到监控日志0号索引中
	STATE = POUT(1)	// 保存输出的POUT1 信号值
	PRINT(1,STATE)	// 保存到监控日志1号索引中
	STATE = POUT(I0)	// 参照全局变量I0号值保存到 POUT 信号中
	PRINT(2,STATE)	// 保存到监控日志2号索引中

Related terms

OUT

8.5 CIN（现场总线比特输入）

Usage

从现场总线读取指定比特编号的值.

Syntax

CIN<比特编号> = <0/1> 或 CIN(<比特编号>) = <0/1>
<变量> = CIN<比特编号> 或 <变量> = CIN(<比特编号>)

Description

CIN 命令提供2种使用方法. 第一种方法为, 向指定比特代入0 或 1 值，直到输入信号与代入值相等为止等待. 第二种方法为, 读取指定比特编号信号值保存到变量中.

1) <比特编号>使用括号输入或直接输入数字使用. 文字CIN和 <编号>之间不可以有空白.
2) <比特编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量>	Data type: <整数型> 结果值返回0(OFF) 或 1(ON)值.

Arguments

<0/1>	Data type: <整数型> 只能输入0(OFF) 或1(ON).
<比特编号>	Data type: <整数型> • CC-Link V1/V2, Ethernet IP, DeviceNet: CIN0 ~ CIN127 (远程输入输出域)

Examples

Example 1

```
SVON                    // 伺服 ON
JMOV P0                 // JMOV到P0
CIN0 = 1                // 等待CIN0号输入信号变为1
JMOV P1                 // JMOV到P1
```

Example 2

```
INT STATE, NUM          // 声明整数型变量
NUM = 0                 // 整数型 变量初始化
STATE = CIN0            // 保存CIN0 信号值
PRINT(0, STATE)         // 保存到监控日志0号索引中
STATE = CIN(0)          // 保存CIN0 信号值
PRINT(1,STATE)          // 保存到监控日志1号索引中
STATE = CIN(NUM)        // 参照NUM 变量值保存 CIN 信号
PRINT(2,STATE)          // 保存到监控日志2号索引中
```

Related terms

CBIN, CWIN, CDIN, CFIN

1) 请注意信号值等待的使用方法.

8.6 COUT (现场总线输出)

Usage

向指定现场总线卡的比特编号输出指定值.

Syntax

COUT<比特编号> = <0/1> 或 COUT(<比特编号>) = <0/1>
<变量> = COUT<比特编号> 或 <变量> = COUT(<比特编号>)
COUT<比特编号> = <0/1>[<脉冲幅度>]
COUT<比特编号> = <0/1>[<脉冲幅度> <脉冲周期>]

Description

COUT 命令提供4种使用方法.
第一种方法输出指定值.
第二种方法读取指定向比特编号输出的信号值保存到变量中.
第三种方法向指定比特编号输出脉冲信号.
第四种方法向指定比特编号输出脉冲周期波形.

1) <比特编号>使用括号输入或直接输入数字使用. 文字COUT和 <编号>之间不可以有空白.
2) <比特编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.
3) 追加输入<脉冲幅度>时，输出输入时间的脉冲信号后回到原来值.
4) 输入<脉冲周期>时，输出以输入时间为周期的脉冲周期波形. <脉冲周期>始终与 <脉冲幅度>同时输入, 一定要输入大于 <脉冲幅度>的值.

Return value

<变量>	Data type: <整数型> 返回结果值0(OFF) 或1(ON)值.

Arguments

<0/1>	Data type: <整数型> 只允许输入0(OFF) 或1(ON).
<比特编号>	Data type: <整数型> • CC-Link V1/V2, Ethernet IP, DeviceNet: COUT0 ~ COUT127 (远程输入输出域)
<脉冲幅度>	Data type: <整数型> 单位为 1ms 但忽略 2ms 以下值，因此请输入10的倍数.
<脉冲周期>	Data type: <整数型> 单位为 1ms 但忽略 2ms 以下值，因此请输入10的倍数.. 脉冲周期一定要大于脉冲幅度.

Examples

Example 1	COUT(0) = 1	// 向COUT0号输出信号输出1值
	DLAY 1000	// 等待1秒
	COUT0 = 0	// 向COUT0输出信号输出0值
Example 2	INT STATE	// 声明整数型变量
	STATE = COUT1	// 保存COUT1 信号值
	PRINT(0,STATE)	// 保存到监控日志0号索引中
	STATE = COUT(1)	// 保存COUT1 信号值
	PRINT(1,STATE)	// 保存到监控日志1号索引中
	STATE = COUT(IO)	// 参照全局变量 IO号值保存 COUT
	PRINT(2,STATE)	// 保存到监控日志2号索引中
Example 3	COUT0 = 0	// 向COUT0号输出信号输出0值
	COUT(0) = 1 100	// 向COUT0号输出信号输出0.1秒长度的脉冲
Example 4	COUT1 = 0	// 向COUT1号输出信号输出0值
	COUT1 = 1 1000 5000	// 向COUT1号输出信号输出5秒为周期1秒脉冲幅度的波形

Related terms

CBOUT, CWOUT, CDOUT, CFOUT

1) 使用COUT命令生成脉冲信号或脉冲周期波形时，输入的<脉冲幅度> 及 <脉冲周期> 值的单位为1ms. 但， 2ms 以下值被忽略，因此请输入2倍数的值.
   - COUT0 = 1 1 → 无脉冲输出，继续维持输出值
   - COUT0 = 1 15 → 输出14ms 长度的脉冲
   - COUT0 = 1 10 15 → 输出以14ms 为周期10ms脉冲幅度的波形
   - COUT0 = 1 15 29 → 输出以28ms 为周期14ms脉冲幅度的波形

8.7 CBIN（现场总线字节输入）

Usage

从现场总线读取指定字节编号的值.

Syntax

<变量> = CBIN<字节编号> 或 <变量> = CBIN(<字节编号>)

Description

读取指定字节编号的信号值保存到变量中.

1) <字节编号>使用括号输入或直接输入数字使用. 文字CBIN和 <编号>之间不可以有空白.
2) <字节编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量>	Data type: <整数型> 结果值返回8比特整数值. 数据值从右侧增加到左侧. 请参考PIN（数字端口输入）说明.

Arguments

<字节编号>	Data type: <整数型> • CC-Link V1/V2, Ethernet IP, DeviceNet: CBIN0 ~ CBIN15 (远程输入输出域)

Examples

Example 1

```
INT STATE                  // 声明整数型变量
STATE = CBIN0              // 保存CBIN0 信号值
PRINT(0,STATE)             // 保存到监控日志0号索引中
STATE = CBIN(0)            // 保存CBIN0 信号值
PRINT(1,STATE)             // 保存到监控日志1号索引中
```

Example 2

• 位运算符 应用例子

```
DEFSTR STR                 // 字符串变量声明
INT AA, RES                // 声明整数型变量
AA = CBIN0                 // 保存CBIN0 信号值
RES = AA | 0HF0            // 第4 ~7值比特OR运算
STR = BINS(RES)            // 保存为二进制字符串
PRINT(0,STR)               // 二进制字符串保存到监控日志0号索引中
RES = 0B0110 & CBIN0       // 第1 ~2值比特 AND 运算
STR = BINS(RES)            // 保存为二进制字符串
PRINT(1,STR)               // 二进制字符串保存到监控日志1号索引中
```

Related terms

CIN, CWIN, CDIN, CFIN

8.8 CBOUT (现场总线字节输出)

Usage

向指定现场总线卡的字节编号中输出指定值.

Syntax

CBOUT<字节编号> = <值> 或 CBOUT(<字节编号>) = <值>
<变量> = CBOUT<字节编号> 或 <变量> = CBOUT(<字节编号>)

Description

CBOUT命令提供2种使用方法.
第一种方法输出指定值.
第二种方法读取向指定字节编号输出的信号值保存到变量中.

1) <字节编号>使用括号输入或直接输入数字使用. 文字CBOUT和 <编号>之间不可以有空白.
2) <字节编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量> Data type: <整数型>
结果值返回8比特整数值(0 ~ 255). 数据值从右侧增加到左侧.
请参考POUT(数字端口输出) 说明.

Arguments

<值> Data type: <整数型>
可输入的值的范围为 8比特 -128 ~ 127.

<字节编号> Data type: <整数型>

- CC-Link V1/V2, Ethernet IP, DeviceNet : CBOUT0 ~ CBOUT15 (远程输入输出域)

Examples

Example 1

```
CBOUT(0) = 62                 // 向CBOUT 0号输入字节值
CBOUT0 = 0                    // 向CBOUT 0号输出0值
```

Example 2

```
INT STATE                     // 声明整数型变量
STATE = CBOUT1                // 保存输出的CBOUT1 信号值
PRINT(0,STATE)                // 保存到监控日志0号索引中
STATE = CBOUT(1)              // 保存输出的 CBOUT1 信号值
PRINT(1,STATE)                // 保存到监控日志1号索引中
STATE = CBOUT(I0)             // 参照全局变量 I0号值保存到CBOUT信号中
PRINT(2,STATE)                // 保存到监控日志2号索引中
```

Related terms

COUT, CWOUT, CDOUT, CFOUT

8.9 CDIN (现场总线双单词输入)

Usage

以整数类型读取现场总线卡中指定的双单词编号的值.

Syntax

<变量> = CDIN<双单词编号> 或 <变量> = CDIN(<双单词编号>)

Description

读取指定的双单词编号的信号值保存到变量中.

1) <双单词编号>使用括号输入或直接输入数字使用. 文字CDIN和 <编号>之间不可以有空白.
2) <双单词编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量>	Data type: <整数型> 结果值返回32比特整数值.

Arguments

<双单词编号>	Data type: <整数型> • CC-Link V1/V2, Ethernet IP, DeviceNet : CDIN0 ~ CDIN7 (远程注册域)

Examples

```
INT DATA                    // 声明整数型变量
DATA = CDIN0                // 保存CDIN0 信号值
PRINT(0, DATA)              // 以整数值保存到监控日志0号索引中
DATA = CDIN(0)              // 保存CDIN0 信号值
PRINT(1, DATA)              // 以整数值保存到监控日志1号索引中
```

Related terms

CIN, CBIN, CWIN

8.10 CDOUT (现场总线双单词输出)

Usage

以整数类型向指定现场总线卡的双单词编号输出指定值.

Syntax

CDOUT<双单词编号> = <值> 或 CDOUT(<双单词编号>) = <值>
<变量> = CDOUT<双单词编号> 或 <变量> = CDOUT(<双单词编号>)

Description

CDOUT 命令提供2种使用方法.
第一种方法以整数类型输出指定值.
第二种方法读取输出到指定双单词编号的信号值保存到变量中.

1) <双单词编号>使用括号输入或直接输入数字使用. 文字CDOUT和 <编号>之间不可以有空白.
2) <双单词编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量>	Data type: <整数型> 结果值返回32比特整数值.

Arguments

<值>	Data type: <整数型> 可输入的值的范围为32比特 -2,147,483,648 ~ 2,147,483,647.
<双单词编号>	Data type: <整数型> • CC-Link V1/V2, Ethernet IP, DeviceNet : CFIN0 ~ CFIN7 (远程注册域)

Examples

Example 1

```
CDOUT(0) = 145748          // 向CDOUT 0号输出整数值
CDOUT0 = 0                 // 向CDOUT 0号输出 0值
```

Example 2

```
INT STATE                  // 声明整数型变量
STATE = CDOUT1             // 保存输出的CDOUT1 信号值
PRINT(0,STATE)             // 保存到监控日志0号索引中
STATE = CDOUT(1)           // 保存输出的CDOUT1 信号值
PRINT(1,STATE)             // 保存到监控日志1号索引中
STATE = CDOUT(I0)          // 参照全局变量 I0号值保存 CDOUT 信号
PRINT(2,STATE)             // 保存到监控日志2号索引中
```

Related terms

COUT, CBOUT, CWOUT

8.11 CWIN (现场总线单词输入)

Usage

读取现场总线卡中指定的单词编号的值.

Syntax

<变量> = CWIN<单词编号> 或 <变量> = CWIN(<单词编号>)

Description

读取指定单词编号的信号值保存到变量中.

1) <单词编号>使用括号输入或直接输入数字使用. 文字 CWIN和 <编号>之间不可以有空白.
2) <单词编号>的范围根据使用的现场总线类型不同, 请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量>	Data type: <整数型> 结果值返回16比特整数值. 数据值从右侧增加到左侧. 请参考PIN (数字端口输入) 说明.

Arguments

<单词编号>	Data type: <整数型> • CC-Link V1/V2, Ethernet IP, DeviceNet : CWIN0 ~ CWIN15 (远程注册域)

Examples

Example 1

```
INT STATE                    // 声明整数型变量
STATE = CWIN0                // 保存CWIN0 信号值
PRINT(0,STATE)               // 保存到监控日志0号索引中
STATE = CWIN(0)              // 保存CWIN0 信号值
PRINT(1,STATE)               // 保存到监控日志1号索引中
```

Example 2

• 位运算符应用例子

```
DEFSTR STR                   // 字符串变量声明
INT AA, RES                  // 声明整数型变量
AA = CWIN0                   // 保存CWIN0 信号值
RES = AA | 0HFF00            // 第8 ~15值比特OR运算
STR = BINS(RES)              // 保存为二进制字符串
PRINT(0,STR)                 // 二进制字符串保存到监控日志0号索引中
RES = 0B10000110 & CWIN0     // 第1 ~2, 7值比特AND 运算
STR = BINS(RES)              // 保存为二进制字符串
PRINT(1,STR)                 // 二进制字符串保存到监控日志1号索引中
```

Related terms

CIN, CBIN, CDIN, CFIN

8.12 CWOUT (现场总线单词输出)

Usage

向指定现场总线卡的单词编号输出指定值.

Syntax

CWOUT<单词编号> = <值> 或 CWOUT(<单词编号>) = <值>
<变量> = CWOUT<单词编号> 或 <变量> = CWOUT(<单词编号>)

Description

CWOUT命令提供2种使用方法.
第一种方法输出指定值.
第二种方法读取指定向单词编号输出的信号值保存到变量中.

1) <单词编号>使用括号输入或直接输入数字使用. 文字CWOUT和 <编号>之间不可以有空白.
2) <单词编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量> Data type: <整数型>
结果值返回16比特整数值. 数据值从右侧增加到左侧.
请参考POUT(数字端口输出) 说明.

Arguments

<值> Data type: <整数型>
可输入的值的范围为16比特 -32,768 ~ 32,767.

<单词编号> Data type: <整数型>

- CC-Link V1/V2, Ethernet IP, DeviceNet : CWOUT0 ~ CWOUT15 (远程注册域)

Examples

Example 1

```
CWOUT(0) = -2456          // 向CWOUT 0号输出 16比特整数值
CWOUT0 = 0                // 向CWOUT 0号输出0 值
```

Example 2

```
INT STATE                 // 声明整数型变量
STATE = CWOUT1            // 保存输出的CWOUT1 信号值
PRINT(0,STATE)            // 保存到监控日志0号索引中
STATE = CWOUT(1)          // 保存输出的CWOUT1 信号值
PRINT(1,STATE)            // 保存到监控日志1号索引中
STATE = CWOUT(IO)         // 参照全局变量 IO号值保存 CWOUT 信号
PRINT(2,STATE)            // 保存到监控日志2号索引中
```

Related terms

COUT, CBOUT, CDOUT, CFOUT

8.13 CFIN (现场总线双单词输入)

Usage

以实数类型读取现场总线卡中指定的双单词编号的值.

Syntax

<变量> = CFIN<双单词编号> 或 <变量> = CFIN(<双单词编号>)

Description

读取指定的双单词编号的信号值保存到变量中.

1) <双单词编号>使用括号输入或直接输入数字使用. 文字 CFIN和 <编号>之间不可以有空白.
2) <双单词编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量>	Data type: <实数型> 结果值返回32比特实数值.

Arguments

<双单词编号>	Data type: <整数型> • CC-Link V1/V2, Ethernet IP, DeviceNet : CFIN0 ~ CFIN7 (远程注册域)

Examples

```
REAL DATA                    // 声明实数型变量
DATA = CFIN0                 // 保存CFIN0 信号值
PRINT(0, DATA)               // 以实数值保存到监控日志0号索引中
DATA = CFIN(0)               // 保存CFIN0 信号值
PRINT(1, DATA)               // 以实数值保存到监控日志1号索引中
```

Related terms

CIN, CBIN, CWIN, CDIN

8.14 CFOUT (现场总线双单词输出)

Usage

以实数类型向指定现场总线卡的双单词编号输出指定值.

Syntax

CFOUT<双单词编号> = <值> 或 CFOUT(<双单词编号>) = <值>
<变量> = CFOUT<双单词编号> 或 <变量> = CFOUT(<双单词编号>)

Description

CFOUT命令提供2种使用方法.
第一种方法输出指定的实数值.
第二种方法读取向指定双单词编号输出的信号值保存到变量中.

1) <双单词编号>使用括号输入或直接输入数字使用. 文字 CFOUT和 <编号>之间不可以有空白.
2) <双单词编号>的范围根据使用的现场总线类型不同，请留意. 现场总线类型可以通过Public参数中1.HW_CONF(1/3)→4.OPTCARD设置.

Return value

<变量> Data type: <实数型>
结果值返回32比特实数值.

Arguments

<值> Data type: <实数型>
可输入的值的范围为32比特3.402,823E-38 ~ 3.402,823E+38.

<双单词编号> Data type: <整数型>
- CC-Link V1/V2, Ethernet IP, DeviceNet : CFIN0 ~ CFIN7 (远程注册域)

Examples

Example 1

```
CFOUT(0) = 14.5748          // 向CFOUT 0号输出实数值
CFOUT0 = 0.0                // 向CFOUT 0号输出 0值
```

Example 2

```
REAL STATE                  // 声明实数型变量
STATE = CFOUT1              // 保存输出的CFOUT1 信号值
PRINT(0,STATE)              // 保存到监控日志0号索引中
STATE = CFOUT(1)            // 保存输出的CFOUT1 信号值
PRINT(1,STATE)              // 保存到监控日志1号索引中
STATE = CFOUT(I0)           // 参照全局变量 I0号值保存 CFOUT 信号
PRINT(2,STATE)              // 保存到监控日志2号索引中
```

Related terms

COUT, CBOUT, CWOUT, CDOUT

8.15 XIN (输入CNET 比特)

Usage

参考指定的设备区域和比特编号，从 CNET 接收缓冲读取值.

Syntax

<整数型变量> = XIN(<设备区域>, <比特编号>)

Description

参考指定的设备区域和比特编号，从 CNET 接收缓冲读取比特值.

1) <设备区域>输入字符串 “M” 或 “D”.
2) <比特编号>的范围根据设备区域不同.

Return value

<变量>	Data type: <整数型> 结果值返回0(OFF) 或 1(ON)值.

Arguments

<设备区域>	Data type: <字符串> 只允许输入“M” 或 “D” 字符串.
<比特编号>	Data type: <整数型> • 设备 M 区域: 0 ~ 383 (共 384个) • 设备 D 区域: 0 ~ 2,047 (共 2,048个)

Examples

Example 1

```
INT DATA                          // 整数型变量声明
DATA = XIN(“M”,0)                 // 读取设备 M 区域 0号比特值
```

Example 2

```
IF XIN(“D”, 100) == 1 THEN        // 设备 D 区域100号比特值为1时
    JMOV P1                       // 向P1 位置JMOV
ENDIF                             // 条件语句结束
```

Related terms

XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.16 XOUT (输出CNET 比特)

Usage

参考指定设备及比特编号，向 CNET 设备发信缓冲输出值.

Syntax

XOUT(<设备区域>, <比特编号>) = <0/1>
<整数型变量> = XOUT(<设备区域>, <比特编号>)

Description

XOUT 指令有两种使用方法.
第一种方法为将指定值输出到CENT 发送缓冲中.
第二种方法为读取以指定比特编号输出的信号值保存到变量中.

1) <设备区域>输入字符串 “M” 或 “D”.
2) <比特编号>的范围根据设备区域不同.

Return value

<变量> Data type: <整数型>
结果值返回0(OFF) 或 1(ON)值.

Arguments

<设备区域> Data type: <字符串>
只允许输入“M” 或 “D” 字符串.

<比特编号> Data type: <整数型>

- 设备 M 区域: 0 ~ 383 (共 384个)
- 设备 D 区域: 0 ~ 2,047 (共 2,048个)

Examples

Example 1

```
XOUT(“M”, 0) = 1                // 向设备 M 区域 0号比特输出1
```

Example 2

```
INT DATA                        // 整数型变量声明
XOUT(“D”, 100) = 1              // 向设备 D 区域 100号比特输出1
DATA = XOUT(“D”, 100)           // 读取设备 D 区域100号比特输出值
```

Related terms

XIN, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.17 XBIN (输入CNET 字节)

Usage

参考指定的设备及字节编号，读取CNET 接收缓冲的值.

Syntax

<整数型变量> = XBIN(<设备区域>, <字节编号>)

Description

参考指定设备区域和字节编号，读取 CNET 接收缓冲的字节输入值.

1) <设备区域>输入字符串 “M” 或 “D”.
2) <字节编号>的范围根据设备区域不同.

Return value

<变量> Data type: <整数型>
结果值返回8比特整数值，数据值从右侧向左侧增加.
请参考PIN (数字端口输入) 说明.

Arguments

<设备区域> Data type: <字符串>
只允许输入“M” 或 “D” 字符串.

<字节编号> Data type: <型>

- 设备 M 区域: 0 ~ 47 (供 48个)
- 设备 D 区域: 0 ~ 255 (供 256个)

Examples

Example 1

```
INT DATA                          // 整数型变量声明
DATA = XBIN(“M”, 0)               // 读取设备M 区域0号字节数据
```

Example 2

- 比特运算符应用例子

```
INT DATA                          // 整数型变量声明
DEFSTR STR                        // 字符串变量声明
DATA = XBIN("D",0) & 0H000F       // 设备D区域 0号字节的第0 ~ 3值比特AND
                                  //运算
STR = BINS(DATA)                  // 结果数据转换为二进制字符串
PRINT(0,STR)                      // 监控日志0号索引中输出字符串
DATA = 0B11110000 ^ XBIN("D",0)   // 设备D区域0号字节的第4 ~ 7值比特XOR
                                  //运算
STR = BINS(DATA)                  // 结果数据转换为二进制字符串
PRINT(1,STR)                      // 监控日志2号索引中输出字符串
```

Related terms

XIN, XOUT, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.18 XBOUT (输出CNET 字节)

Usage

参考指定的设备及字节编号，向 CNET 设备发送缓冲输出值.

Syntax

XBOUT(<设备区域>, <字节编号>) = <输出值>
<整数型变量> = XBOUT(<设备区域>, <字节编号>)

Description

XBOUT 指令有两种使用方法.
第一种方法为将指定值输出到CENT 发送缓冲中.
第二种方法为读取以指定字节编号输出的信号值保存到变量中.

1) <设备区域>输入字符串 “M” 或 “D”.
2) <字节编号>的范围根据设备区域不同.

Return value

<变量>	Data type: <整数型> 结果值返回8比特整数值, 数据值从右侧向左侧增加. 请参考PIN (数字端口输入) 说明.

Arguments

<设备区域>	Data type: <字符串> 只允许输入“M” 或 “D” 字符串.
<字节编号>	Data type: <型> • 设备 M 区域: 0 ~ 47 (供 48个) • 设备 D 区域: 0 ~ 255 (供 256个)

Examples

Example 1	XBOUT(“M”, 0) = 0HFF	// 向设备M区域 0号字节写入值
Example 2	INT DATA DATA = XBOUT(“D”, 0)	// 整数型变量声明 // 读取设备D区域 0号字节输出值

Related terms

XIN, XOUT, XBIN, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.19 XWIN (读取CNET Word)

Usage

参考指定设备及Word编号，读取 CNET 接收缓冲的值.

Syntax

<整数型变量> = XWIN(<设备区域>, <Word编号>)

Description

参考指定设备及Word编号，读取 CNET 接收缓冲的Word输入.

1) <设备区域>输入字符串 “M” 或 “D”.
2) <Word编号>的范围根据设备区域不同.

Return value

<变量> Data type: <整数型>
结果值返回16比特整数值，数据值从右侧向左侧增加.
请参考PIN (数字端口输入) 说明.

Arguments

<设备区域> Data type: <字符串>
只允许输入“M” 或 “D” 字符串.

<Word编号> Data type: <型>

- 设备 M 区域: 0 ~ 23 (供 24个)
- 设备 D 区域: 0 ~ 127 (供 128个)

Examples

Example 1

```
INT DATA                          // 整数型变量声明
DATA = XWIN(“M”, 0)               // 读取设备 M 区域 0号Word值
```

Example 2

- 比特运算应用例子

```
INT RES                           // 整数型变量声明
DEFSTR STR                        // 字符串变量声明
RES = XWIN(“D”, 0) | 0HFF00       // 设备D区域 0号Word的 第8 ~ 15值比特
                                  //OR运算
STR = BINS(RES)                   // 保存为二进制字符串
PRINT(0,STR)                      // 监控日志0号索引中输出字符串
RES = 0B10000110 & XWIN(“D”, 0)   // 设备D区域 0号Word的第1 ~ 2, 7值比
                                  //特 AND 运算
STR = BINS(RES)                   // 保存为二进制字符串
PRINT(1,STR)                      // 监控日志1号索引中输出字符串
```

Related terms

XIN, XOUT, XBIN, XBOUT, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.20 XWOUT (输出CNET Word)

Usage

参考指定的设备及Word编号，向 CNET 设备发信缓冲输出值.

Syntax

XWOUT(<设备区域>, <Word编号>) = <输出值>
<整数型变量> = XWOUT(<设备区域>, <Word编号>)

Description

XWOUT 指令有2种使用方法.
第一种方法为将指定值输出到CENT 发送缓冲中..
第二种方法为读取以指定Word编号输出的信号值保存到变量中.

1) <设备区域>输入字符串 “M” 或 “D”..
2) <Word编号>的范围根据设备区域不同.

Return value

<变量> Data type: <整数型>
结果值返回16比特整数值, 数据值从右侧向左侧增加.
请参考PIN (数字端口输入) 说明..

Arguments

<设备区域> Data type: <字符串>
只允许输入“M” 或 “D” 字符串.

<Word编号> Data type: <型>

- 设备 M 区域: 0 ~ 23 (供 24个)
- 设备 D 区域: 0 ~ 127 (供 128个)

Examples

Example 1

```
XWOUT(“M”, 0) = 0HFFFF          // 向设备 M 区域 0号Word中写入值
```

Example 2

```
INT DATA                        // 整数型变量声明
DATA = XWOUT(“D”, 0)            // 读取设备 D 区域 0号Word的输出值
```

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XDIN, XDOUT, XFIN, XFOUT

8.21 XDIN (输入CNET Double Word)

Usage

参考指定设备及Double Word编号，读取CNET 接收缓冲的值.

Syntax

<整数型变量> = XDIN(<设备区域>, <Double Word编号>)

Description

参考指定的设备区域和 Double Word编号，从CNET 接收缓冲读取Double Word输入.

1) <设备区域>输入字符串 “M” 或 “D”..
2) <Double Word编号>的范围根据设备区域不同.

Return value

<变量>	Data type: <整数型> 结果值返回32比特整数值.

Arguments

<设备区域>	Data type: <字符串> 仅允许输入“M” 或 “D” 字符串.
<Double Word编号>	Data type: <型> • 设备 M 区域: 0 ~ 11 (供 12个) • 设备 D 区域: 0 ~ 63 (供 64个)

Examples

Example 1

```
INT DATA                          // 整数型变量声明
DATA = XDIN(“M”, 0)               // 读取设备 M 区域 0号Double Word值
```

Example 2

• 比特运算符应用例子

```
INT RES                           // 整数型变量声明
DEFSTR STR                        // 字符串变量声明
RES = XDIN(“D”, 0) | 0HFFFF0000   // 设备D区域 0号 Double Word的第
                                  //16 ~ 32值比特OR运算
STR = BINS(RES)                   // 保存为二进制字符串
PRINT(0,STR)                      // 监控日志 0号索引中输出字符串
```

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDOUT, XFIN, XFOUT

8.22 XDOUT (输出CNET Double Word)

Usage

参考指定的设备及Double Word编号，向 CNET 设备发送缓冲输出值.

Syntax

XDOUT(<设备区域>, <Double Word编号>) = <输出值>
<整数型变量> = XDOUT(<设备区域>, <Double Word编号>)

Description

XDOUT 指令有两种使用方法.
第一种方法为将指定值输出到CENT 发送缓冲中..
第二种方法为读取以Double Word编号输出的信号值保存到变量中.

1) <设备区域>输入字符串 “M” 或 “D”..
2) <Double Word编号>的范围根据设备区域不同.

Return value

<变量>	Data type: <整数型> 结果值返回32比特整数值.

Arguments

<设备区域>	Data type: <字符串> 仅允许输入“M” 或 “D” 字符串.
<Double Word编号>	Data type: <型> • 设备 M 区域: 0 ~ 11 (供 12个) • 设备 D 区域: 0 ~ 63 (供 64个)

Examples

Example 1	XDOUT(“M”, 0) = 0HFFFFFFFF	// 向设备 M 区域 0号Double Word中输入值
Example 2	INT DATA DATA = XDOUT(“D” ,0)	// 整数型变量声明 // 读取设备 D 区域 0号Double Word输出值

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XFIN, XFOUT

8.23 XFIN (输入CNET 实数)

Usage

参考指定的设备及Double Word编号，从CNET 接收缓冲中以读取实数型值.

Syntax

<实数型变量> = XFIN(<设备区域>, <Double Word编号>)

Description

参考指定的设备区域和 Double Word编号，从 CNET接收缓冲读取实数型输入值.

1) <设备区域>输入字符串 “M” 或 “D”..
2) <Double Word编号>的范围根据设备区域不同.

Return value

<变量>	Data type: <实数型> 结果值返回32比特实数值.

Arguments

<设备区域>	Data type: <字符串> 仅允许输入“M” 或 “D” 字符串.
<Double Word编号>	Data type: <型> • 设备 M 区域: 0 ~ 11 (供 12个) • 设备 D 区域: 0 ~ 63 (供 64个)

Examples

Example

```
REAL DATA1, DATA2              // 声明实数型变量
DATA1 = XFIN(“M”, 0)           // 读取设备 M 区域 0号Double Word区域的
                               //实数值
DATA2 = XFIN(“D”, 63)          // 读取设备 D 区域 63号Double Word区域的
                               //实数值
```

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFOUT

8.24 XFOUT (输出CNET 实数)

Usage

参考指定的设备及Double Word编号，向 CNET 设备发送缓冲输出实数值.

Syntax

XFOUT(<设备区域>, <Double Word编号>) = <实数值>
<实数型变量> = XFOUT(<设备区域>, <Double Word编号>)

Description

XDOUT 命令有两种使用方法.
第一种方法为将指定值输出到CENT 发送缓冲中..
第二种方法为读取指定Double Word编号中输出的实数值保存到变量中.

1) <设备区域>输入字符串 “M” 或 “D”..
2) <Double Word编号>的范围根据设备区域不同.

Return value

<变量>	Data type: <实数型> 结果值返回32比特实数值.

Arguments

<设备区域>	Data type: <字符串> 仅允许输入“M” 或 “D” 字符串.
<Double Word编号>	Data type: <型> • 设备 M 区域: 0 ~ 11 (供 12个) • 设备 D 区域: 0 ~ 63 (供 64个)

Examples

Example 1	XFOUT(“M”, 0) = 12.45	// 向设备 M 区域 0号Double Word写入实数值

Example 2	REAL DATA
	DATA = XFOUT(“D”, 0)
		// 声明实数型变量
		// 读取设备 D 区域 0号Double Word区域的
		//实数值

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN

9. Multithreading 控制

9.1 CTHREAD (Thread 生成)

Usage

同时执行函数或程序.

Syntax

[<变量> =] CTHREAD(<函数名>, <Thread 编号>, [<Thread 停止Alarm>])

Description

将函数或程序分配到Thread同时执行.
分配函数时，返回类型一定是VOID类型且无参数.
分配程序时，需要使用INCLUDE 命令预先引入文件.

1) <函数名>中写入函数或程序名.
2) <Thread 编号>输入启动的 Thread编号. 选择的Thread 编号已被其它Thread占用时不会生成Thread.
3) <Thread 停止Alarm>在发生Alarm时设置Thread是否停止.

Return value

<变量>	Data type: <整数型> 结果值返回负数 (Thread 生成失败) 或 0(Thread 生成成功)值.

Arguments

<函数名>	Data type: <函数或程序名> 输入JOB 程序名时，在程序名后面必须追加 “_MAIN”. • 例: TEST.JOB -> CTHREAD(TEST_MAIN, 2)
<Thread 编号>	Data type: <整数型> Thread 编号的范围为 2 ~ 4(TH2 ~ TH4).
<Thread Alarm 停止>	Data type: <整数型> 选择发生Alarm时Thread 停止(1), 发生Alarm时 Thread 不停止(0) 之一输入.

Examples

Example 1

- 调用程序例子

```
INCLUDE “TEST”                          // 引入TEST.JOB
MAIN                                    // 主函数声明
     INT T_RET                          // 声明整数型变量
     T_RET = CTHREAD(TEST_MAIN,2)       // 将TEST.JOB的主函数分配到 Thread 2号
     IF T_RET == 0 THEN                 // 执行结果等于Thread 生成失败时
          EXIT                          // JOB 程序结束
     ENDIF                              // IF语句结束

     …
     WHILE 1                            // 重复执行
     ENDWL                              // 重复语句结束
EOP                                     // 主函数结束
```

Example 2

- 调用函数例子

```
MAIN                                    // 主函数声明
     CTHREAD(MOVE, 2, 0)                // MOVE函数分配到Thread 2号
                                        //发生Alarm时Thread 不停止
     WHILE 1                            // 重复执行
     ENDWL                              // 重复语句结束
EOP                                     // 主函数结束

FUNC VOID MOVE()                        // MOVE 函数声明 (VOID 类型, 无参数)
     WHILE 1                            // 开始执行重复语句
          JMOV P0                       // JMOV到P0
          DLAY 1000                     // 等待1秒
          JMOV P1                       // JMOV到P1
          DLAY 1000                     // 等待1秒
     ENDWL                              // 重复语句结束
END                                     // 函数结束
```

Related terms

ETHREAD, TH_STATE, TH_SUSPEND, TH_RESUME

9.2 ETHREAD (Thread 结束)

Usage

强制停止Thread.

Syntax

[<变量> =] ETHREAD(<Thread 编号>)

Description

完全停止执行中的Thread.
使用ETHREAD 命令结束Thread后无法使用TH_RESUME 命令重新开始.

1) <Thread 编号>中输入要结束的Thread编号. 使用CTHREAD 命令式输入的Thread编号结束Thread.

Return value

<变量>	Data type: <整数型> 结果值返回0(Thread 结束成功) 或 -1(Thread 结束失败)值.

Arguments

<Thread 编号>	Data type: <整数型> Thread 编号的范围为 2 ~ 4(TH2 ~ TH4).

Examples

```
INCLUDE “TEST”                      // 引入TEST.JOB 文件
MAIN                                // 主函数声明
    INT T_RET                       // 声明整数型变量
    CTHREAD(TEST_MAIN,2,1)          // TEST.JOB的主函数分配到 Thread 2号
                                    // 发生Alarm时停止Thread
    IN0 = 1                         // 等待0号输入信号变为1
    T_RET = ETHREAD(2)              // Thread 强制结束

    IF T_RET < 0 THEN               // 执行结果为Thread 结束失败时
        EXIT                        // JOB 程序结束
    ENDIF                           // IF语句结束
EOP                                 // 主函数结束
```

Related terms

CTHREAD, TH_STATE, TH_SUSPEND, TH_RESUME

9.3 TH_STATE (读取Thread 状态)

Usage

确认Thread的当前状态.

Syntax

<变量> = TH_STATE(<执行模式>, <Thread 编号>)

Description

返回各Thread的当前状态.

1) 指定<执行模式>可以确认System及Robot模式的Thread 状态.
2) 指定<Thread 编号>读取相应Thread的状态.

Return value

<变量>	Data type: <整数型> 返回Thread的当前状态信息. • 0 (TH_ERROR): Thread 编号错误 • 1 (TH_IDLE): 未分配Thread, 可以使用 • 2 (TH_STOP): Thread 停止 • 3 (TH_RUN): Thread 执行中

Arguments

<执行模式>	Data type: <整数型> 输入System模式 (1) 或 Robot模式 (0).
<Thread 编号>	Data type: <整数型> Thread 编号的范围为 1 ~ 4(TH1 ~ TH4).

Examples

```
INCLUDE “MOVE”                          // 引入MOVE.JOB
MAIN                                    // 主函数声明
    INT ST                              // 声明整数型变量
    INT P_ID, T_ID

    P_ID = 0                            // 指定Robot模式
    T_ID = 2                            // 指定Thread 2号

    CTHREAD(MOVE_MAIN,T_ID,1)           // 为MOVE.JOB的主函数分配 Thread
                                        // 发生Alarm时结束Thread
    WHILE 1                             // 无限重复语句开始
        ST = TH_STATE(P_ID, T_ID)       // 确认Robot模式的Thread 2号状态

        IF (ST == 2) THEN               // 如果未TH_STOP 状态
            TH_RESUME(T_ID)             // 重新开始Thread
        ELSEIF (ST == 3 && IN0 == 1) THEN  // TH_RUN状态且0号输入信号为1时
            TH_SUSPEND(T_ID)            // Thread 停止
        ENDIF                           // 条件语句结束

        DLAY 1000                       // 等待1秒

        IF (ST == 1 && ST == 0) THEN    // Thread未分配或发生错误时
```

```
                ETHREAD(T_ID)                   // Thread 结束
            ENDIF                               // 条件语句结束

        ENDWL                                   // 重复语句结束
    EOP                                         // 主函数结束
```

Related terms

CTHREAD, ETHREAD, TH_SUSPEND, TH_RESUME

9.4 TH_SUSPEND (Thread 停止)

Usage

停止执行中的Thread.

Syntax

[<变量> =] TH_SUSPEND(<Thread 编号>)

Description

停止执行中的Thread.
停止状态的Thread可以通过TH_RESUME 命令重新执行.

1) <Thread 编号>中输入要停止的Thread编号.

Return value

<变量>	Data type: <整数型> 结果值返回0(Thread 停止成功) 或 -1(Thread 停止失败)值.

Arguments

<Thread 编号>	Data type: <整数型> Thread 编号的范围为 2 ~ 4(TH2 ~ TH4).

Examples

```
INCLUDE “MOVE”                          // 引入MOVE.JOB
MAIN                                    // 主函数声明
    INT ST                              // 声明整数型变量
    INT P_ID, T_ID

    P_ID = 0                            // 指定Robot模式
    T_ID = 2                            // 指定Thread 2号

    CTHREAD(MOVE_MAIN,T_ID,1)           // 为MOVE.JOB的主函数分配 Thread
                                        // 发生Alarm时结束Thread
    WHILE 1                             // 无限重复语句卡碍事
        ST = TH_STATE(P_ID, T_ID)       // 确认Robot模式的Thread 2号状态

        IF (ST == 3 && IN0 == 1) THEN   // TH_RUN状态且0号输入信号为1时
            TH_SUSPEND(T_ID)            // Thread 停止
        ENDIF                           // 条件语句结束
    ENDWL                               // 重复语句结束
EOP                                     // 主函数结束
```

Related terms

CTHREAD, ETHREAD, TH_STATE, TH_RESUME

9.5 TH_RESUME (Thread 重新开始)

Usage

重新执行停止的Thread.

Syntax

[<变量> =] TH_RESUME(<Thread 编号>)

Description

重新执行停止的Thread.
只能重新开始通过TH_SUSPEND停止的Thread.

1) <Thread 编号>中输入重新开始的Thread编号.

Return value

<变量>	Data type: <整数型> 结果值返回0(Thread 重新开始成功) 或 -1(Thread 重新开始失败)值.

Arguments

<Thread 编号>	Data type: <整数型> Thread 编号的范围为 2 ~ 4(TH2 ~ TH4).

Examples

```
INCLUDE “MOVE”                              // 引入MOVE.JOB
MAIN                                        // 主函数声明
    INT ST                                  // 声明整数型变量
    INT T_ID

    T_ID = 2                                // 指定Thread 2号

    CTHREAD(MOVE_MAIN,T_ID,1)               // 为MOVE.JOB的主函数分配 Thread
                                            // 发生Alarm时结束Thread
    WHILE 1                                 // 无限重复语句开始
        ST = TH_STATE(0, T_ID)              // 确认Robot模式的Thread 2号状态

        IF (ST == 2) THEN                   // TH_STOP 状态时
            TH_RESUME(T_ID)                 // Thread 重新开始
        ELSEIF (ST == 3 && IN0 == 1) THEN   // TH_RUN状态且0号输入信号为1时
            TH_SUSPEND(T_ID)                // Thread 停止
        ENDIF                               // 条件语句结束
    ENDWL                                   // 重复语句结束
EOP                                         // 主函数结束
```

Related terms

CTHREAD, ETHREAD, TH_STATE, TH_SUSPEND

10. 函数

10.1 TOOL (选择Tool坐标系)

Usage

选择Robot上使用的Tool和相应坐标系.

Syntax

TOOL <编号>

Description

指定作业时使用的Tool编号.
变更坐标时参照指定Tool参数值计算末端位置及方向.
通过此命令变更的值有效至程序结束为止.

移动命令执行过程中变更Tool值时以命令中插入的编号优先.

1) <编号>中输入要使用的Tool和选择坐标系. 此Tool信息可以通过COORD-TOOL 参数变更.

Arguments

<编号>	Data type: <整数型> Tool编号可以使用0 ~ 15.

Examples

```
MAIN                        // 主函数声明
    SVON                    // 伺服 ON
    JMOV P0                 // JMOV到P0

    VEL 200                 // 变更速度值
    TOOL 1                  // 指定TOOL 1 号

    LMOV P1                 // LMOV到P1
    LMOV P2                 // LMOV到P2
EOP                         // 主函数结束
```

Related terms

USER

1) TOOL 0表示Robot Hand末点, 表示未安装Tool的状态. 无法修改此值.
2) 作业程序中没有使用TOOL 命令时应用TOOL 0 值.

10.2 USER（选择用户坐标系）

Usage

选择用户坐标系.

Syntax

USER <编号>

Description

指定作业时使用的USER编号.
变更坐标时参照指定USER参数值计算末端位置及方向.
通过此命令变更的值有效至程序结束为止.

移动命令执行过程中变更USER值时以命令中插入的编号优先.

1) <编号>中输入要使用的USER和选择坐标系. 此USER信息可以通过COORD-TOOL 参数变更.

Arguments

<编号>	Data type: <整数型> 用户坐标系编号可以使用0 ~ 15.

Examples

```
MAIN                          // 主函数声明
    SVON                      // 伺服 ON
    JMOV P0                   // JMOV到P0

    VEL 200                   // 变更速度值
    USER 1                    // 指定USER 1 号

    LMOV P1                   // LMOV到P1
    LMOV P2                   // LMOV到P2
EOP                           // 主函数结束
```

Related terms

TOOL

1) 作业程序中没有使用USER命令时应用USER 0 值
2) 用户 0 无法修改

10.3 JTOB (坐标数据计算, JOINT -> BASE)

Usage

从Joint坐标系位置值计算出Base坐标系位置值.

Syntax

<变量> = JTOB([<Robot编号>,] <输入变量>)

Description

参照输入的位置变量变量的Tool信息，通过Joint坐标系位置值计算出Base坐标系位置值.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <输入变量>中输入值时，务必输入变量格式的值. 输入常数发出Alarm.

Return value

<变量>	Data type: <Base坐标系 位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<输入变量>	Data type: <Joint坐标系位置型变量>

Examples

```
MAIN                                    // 主函数声明
  POS PDATA                             // Joint坐标系位置型变量声明
  XPOS XRDATA                           // Base坐标系 位置型变量声明

  PDATA = <30,15,-15,45,-45,60>         // 代入Joint位置值
  PDATA.T = 1                           // 代入Tool信息

  XRDATA = JTOB(PDATA)                  // 获取Base坐标系位置值
  PRINT(0,XRDATA)                       // 向监控日志0号保存坐标值
  XRDATA = JTOB(2,PDATA)                // 获取Robot 2的Base坐标系位置值
  PRINT(1,XRDATA)                       // 向监控日志1号保存坐标值
EOP                                     // 主函数结束
```

Related terms

BTOJ, JTOU, UTOJ, BTOU, UTOB, TOOL, USER

10.4 BTOJ (坐标数据计算, BASE -> JOINT)

Usage

通过Base坐标系位置值计算出Joint坐标系位置值.

Syntax

<变量> = BTOJ([<Robot编号>,] <输入变量>)

Description

参照输入的位置变量的Tool信息，通过Base坐标系位置值计算出Joint坐标系位置值
如果未指定Robot姿势信息，则以当前Robot姿势为基准计算Joint坐标系位置值.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <输入变量>中输入值时，务必输入变量格式的值. 输入常数发出Alarm.

Return value

<变量>	Data type: <Joint坐标系位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<输入变量>	Data type: <Base坐标系 位置型变量>

Examples

```
MAIN                                          // 主函数声明
   POS RDATA                                  // Joint坐标系位置型变量声明
   XPOS XRDATA                                // Base坐标系 位置型变量声明

   XRDATA = <100,100,100,30,45,60,ARM = 0,T = 1>  // 代入Base坐标值

   RDATA = BTOJ(XRDATA)                       // 获取Joint坐标值位置值
   PRINT(0, RDATA)                            // 向监控日志0号保存坐标值
   RDATA = BTOJ(2,XRDATA)                     // 获取Robot 2的Joint坐标位置值
   PRINT(1, RDATA)                            // 向监控日志1号保存坐标值
EOP                                           // 主函数结束
```

Related terms

JTOB, JTOU, UTOJ, BTOU, UTOB, TOOL, USER

10.5 JTOU (坐标数据计算, JOINT -> USER)

Usage

通过Joint坐标系位置值计算出用户坐标系位置值.

Syntax

<变量> = JTOU([<Robot编号>,] <输入变量>)

Description

参照输入的位置变量的Tool及用户坐标系信息，通过Joint坐标系位置值计算出用户坐标系位置值.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <输入变量>中输入值时，务必输入变量格式的值. 输入常数发出Alarm.

Return value

<变量>	Data type: <用户坐标系位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<输入变量>	Data type: <Joint坐标系位置型变量>

Examples

```
MAIN                                    // 主函数声明
  POS PDATA                             // Joint坐标系位置型变量声明
  UPOS URDATA                           // 用户坐标系位置型变量声明

  PDATA = <30,15,-15,45,-45,60>         // Joint坐标系位置型变量
  PDATA.T = 2                           // 代入Tool信息
  PDATA.U = 3                           // 代入用户坐标系信息

  URDATA = JTOU(PDATA)                  // 获取用户坐标系位置值
  PRINT(0,URDATA)                       // 向监控日志0号保存坐标值
  URDATA = JTOU(2,PDATA)                // 获取Robot 2的用户坐标系位置值
  PRINT(1,URDATA)                       // 向监控日志1号保存坐标值
EOP                                     // 主函数结束
```

Related terms

JTOB, BTOJ, UTOJ, BTOU, UTOB, TOOL, USER

10.6 UTOJ (坐标数据计算, USER -> JOINT)

Usage

通过用户坐标系位置值计算出 Joint坐标系位置值.

Syntax

<变量> = UTOJ([<Robot编号>,] <输入变量>)

Description

参照输入的位置变量的Tool信息及Robot姿势信息，通过用户坐标系位置值计算出 Joint坐标系位置值.
如果未指定Robot姿势信息，则以当前Robot姿势为基准计算Joint坐标系位置值.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <输入变量>中输入值时，务必输入变量格式的值. 输入常数发出Alarm.

Return value

<变量>	Data type: <用户坐标系位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<输入变量>	Data type: <Joint坐标系位置型变量>

Examples

```
MAIN                                          // 主函数声明
   POS RDATA                                  // Joint坐标系位置型变量声明
   UPOS URDATA                                // 用户坐标系位置型变量声明

   URDATA = <0,0,0,30,45,60,ARM = 0,T = 1,U = 3>  // 代入用户坐标值

   RDATA = UTOJ(URDATA)                       // 获取Joint坐标系位置值
   PRINT(0, RDATA)                            // 向监控日志0号保存坐标值
   RDATA = UTOJ(2,URDATA)                     // 获取Robot2的Joint坐标系位置值
   PRINT(1, RDATA)                            // 向监控日志1号保存坐标值
EOP                                           // 主函数结束
```

Related terms

JTOB, BTOJ, JTOU, BTOU, UTOB, TOOL, USER

10.7 BTOU (坐标数据计算, BASE -> USER)

Usage

通过Base坐标系位置值计算用户坐标系位置值.

Syntax

<变量> = BTOU([<Robot编号>,] <输入变量>)

Description

参照位置变量的用户坐标系信息，通过Base坐标系位置值计算出用户坐标系位置值.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <输入变量>中输入值时，务必输入变量格式的值. 输入常数发出Alarm.

Return value

<变量>	Data type: <用户坐标系位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<输入变量>	Data type: <Base坐标系 位置型变量>

Examples

```
MAIN                                          // 主函数声明
   XPOS XRDATA                                // Base坐标系 位置型变量声明
   UPOS URDATA                                // 用户坐标系位置型变量声明

   XRDATA = <10,10,10,30,45,-60,ARM = 0,T = 1>  // 代入Base坐标值
   XRDATA.U = 3
                                              // 代入用户坐标系信息
   URDATA = BTOU(XRDATA)                      // 获得用户坐标系位置值
   PRINT(0, URDATA)                           // 向监控日志0号保存坐标值
   URDATA = BTOU(2,XRDATA)                    // 获得Robot2的用户坐标系位置值
   PRINT(1, URDATA)                           // 向监控日志1号保存坐标值
EOP                                           // 主函数结束
```

Related terms

JTOB, BTOJ, JTOU, UTOJ, UTOB, TOOL, USER

10.8 UTOB (坐标数据计算, USER -> BASE)

Usage

通过用户坐标系位置值计算出Base坐标系位置值.

Syntax

<变量> = UTOB([<Robot编号>,] <输入变量>)

Description

参数输入的用户坐标系信息，通过用户坐标系位置值计算出Base坐标系位置值.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <输入变量>中输入值时，务必输入变量格式的值. 输入常数发出Alarm.

Return value

<变量>	Data type: <Base坐标系 位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<输入变量>	Data type: <用户坐标系位置型变量>

Examples

```
MAIN                                                    // 主函数声明
   XPOS XDATA                                           // Base坐标系 位置型变量声明
   UPOS UDATA                                           // 用户坐标系位置型变量声明

   UDATA = <10,10,10,30,45,60,ARM = 0,T = 1,U = 2>      // 代入用户坐标值

   XDATA = UTOB(UDATA)                                  // 获得Base坐标系位置值
   PRINT(0, XDATA)                                      // 向监控日志0号保存坐标值
   XDATA = UTOB(2, UDATA)                               // 获得Robot2的Base坐标系位置值
   PRINT(1, XDATA)                                      // 向监控日志1号保存坐标值
EOP                                                     // 主函数结束
```

Related terms

JTOB, BTOJ, JTOU, UTOJ, BTOU, TOOL, USER

10.9 RELBASE (Base坐标系基准的相对移动)

Usage

计算以Base坐标系信息为基准移动基准位置中代入的移动量的位置值的函数.

Syntax

<变量> = RELBASE([<Robot编号>,] <基准位置>, <移动量>)

Description

计算从Base坐标系 基准位置向当前Base坐标系 方向相对移动的结果值，返回Base坐标系值的函数.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <基准位置>中输入值时务必输入Base坐标系 位置型变量. 输入常数或其它变量时发出Alarm.
3) <移动量>中输入值时务必输入位置型变量. 输入常数发出Alarm.

Return value

<变量>	Data type: <Base坐标系 位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<基准位置>	Data type: <Base坐标系 位置型变量>
<移动量>	Data type: <所有坐标系位置型变量>

Examples

```
MAIN                                    // 主函数声明
    POS ACUR, DATA                      // Joint坐标系位置型变量声明
    XPOS XDATA1, XDATA2                 // Base坐标系 位置型变量声明
    XPOS RDATA1, RDATA2

    DATA = <100,0,100,0,0,-50>          // 代入移动量

    ACUR = HERE                         // 保存当前位置值
    XDATA1 = JTOB(ACUR)                 // 转换为Base坐标系值
    ACUR = HERE(2)                      // 保存Robot2的当前位置值
    XDATA2 = JTOB(2,ACUR)               // 转换为Robot2的Base坐标系值

    RDATA1 = RELBASE(XDATA1, DATA)      // 计算以Base坐标系基准移动输入的移
                                        //动量的结果
    PRINT(0, RDATA1)                    // 监控日志0中保存结果坐标值
    RDATA2 = RELBASE(2,XDATA2, DATA)    // 计算以Base2坐标系基准移动输入的移
                                        //动量的结果
    PRINT(1, RDATA2)                    // 监控日志1中保存结果坐标值
EOP                                     // 主函数结束
```

Related terms

RELTOOL, RELUSER

10.10 RELTOOL (Tool坐标系基准的相对移动)

Usage

计算以Tool坐标系信息为基准移动基准位置中代入的移动量的位置值的函数

Syntax

<变量> = RELTOOL([<Robot编号>,] <基准位置>, <移动量>)

Description

计算从Base坐标系基准位置向当前Tool坐标系方向相对移动的结果值，返回Base坐标系值的函数.
参照基准位置的Tool编号，进行Tool坐标系相对移动.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <基准位置>中输入值时务必输入Base坐标系 位置型变量. 输入常数或其它变量时发出Alarm.
3) <移动量>中输入值时务必输入位置型变量. 输入常数发出Alarm.

Return value

<变量>	Data type: <Base坐标系 位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<基准位置>	Data type: <Base坐标系 位置型变量>
<移动量>	Data type: <所有坐标系位置型变量>

Examples

```
MAIN                                        // 主函数声明
    POS ACUR, DATA1, DATA2                  // Joint坐标系位置型变量声明
    XPOS XDATA1, XDATA2                     // Base坐标系 位置型变量声明
    XPOS RDATA1, RDATA2

    DATA1 = <100,0,100,0,0,-50>             // 代入移动量
    DATA2 = <-100,0,-100,0,0,-50>

    ACUR = HERE                             // 保存当前位置值
    ACUR.T = 1                              // 指定Tool坐标系1号
    XDATA1 = JTOB(ACUR)                     // 转换为Base坐标系值
    ACUR = HERE(2)                          // 保存Robot2的当前位置值
    ACUR.T = 2                              // 指定Tool坐标系2号
    XDATA2 = JTOB(2,ACUR)                   // 转换为Robot2的Base坐标系值

    RDATA1 = RELTOOL(XDATA1, DATA1)         // 计算以Tool坐标系基准移动输入
                                            //的移动量的结果
    PRINT(0, RDATA1)                        // 监控日志0中保存结果坐标值
    RDATA2 = RELTOOL(2, XDATA2, DATA2)      // 计算以Robot2的Tool坐标系为基
                                            //准移动输入的移动量的结果
    PRINT(1,RDATA2)                         // 监控日志1中保存结果坐标值

    IF ( RDATA1.3 < RDATA2.3 ) THEN         // 计算结果小于Robot2的Z轴位置时
        SVON                                // 伺服 ON
        LMOV RDATA1                         // 从现在位置LMOV到当前位置
    ENDIF                                   // 条件语句结束
EOP                                         // 主函数结束
```

Related terms

RELBASE, RELUSER

10.11 RELUSER (用户坐标系基准的相对移动)

Usage

计算以用户坐标系信息为基准移动基准位置中代入的移动量的位置值的函数.

Syntax

<变量> = RELUSER([<Robot编号>,] <基准位置>, <移动量>)

Description

计算从用户坐标系基准位置向指定用户坐标系方向相对移动的结果值，返回用户坐标系值的函数.
参照基准位置的用户坐标系编号，进行用户坐标系的相对移动.

1) 输入<Robot编号>保存JOB程序时，可以选择非选择的其它Robot.
2) <基准位置>中输入值时务必输入Base坐标系 位置型变量. 输入常数或其它变量时发出Alarm.
3) <移动量>中输入值时务必输入位置型变量. 输入常数发出Alarm.

Return value

<变量>	Data type: <用户坐标系位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<基准位置>	Data type: <用户坐标系位置型变量>
<移动量>	Data type: <所有坐标系位置型变量>

Examples

```
MAIN                                        // 主函数声明
    POS ACUR, DATA1, DATA2                  // Joint坐标系位置型变量声明
    UPOS UDATA1,UDATA2                      // 用户坐标系位置型变量声明
    UPOS RDATA1, RDATA2

    DATA1 = <100,0,100,0,0,-50>             // 代入移动量
    DATA2 = <-100,0,-100,0,0,-50>

    ACUR = HERE                             // 保存当前位置值
    ACUR.T = 0                              // 指定Tool坐标系0号
    ACUR.U = 1                              // 指定用户坐标系1号
    UDATA1 = JTOU(ACUR)                     // 转换为用户坐标系值
    ACUR = HERE(2)                          // 保存Robot2的当前位置值
    ACUR.T = 1                              // 指定Tool坐标系1号
    ACUR.U = 2                              // 指定用户坐标系2号
    UDATA2 = JTOU(2,ACUR)                   // 转换为Robot2的用户坐标系值

    RDATA1 = RELUSER(UDATA1, DATA1)         // 计算以用户坐标系基准移动输入的
                                            //移动量的结果
    PRINT(0, RDATA1)                        // 监控日志0中保存结果坐标值
    RDATA2 = RELUSER(2, UDATA2, DATA2)      // 计算以Robot2用户基准移动输入的
                                            //移动量的结果
    PRINT(1,RDATA2)                         // 监控日志1中保存结果坐标值
EOP                                         // 主函数结束
```

Related terms

RELBASE, RELTOOL

10.12 GBASEDISP (Base坐标系基准移动量检测)

Usage

求输入的两个坐标之间Base坐标系基准移动量的函数.

Syntax

<变量> = GBASEDISP([<Robot编号>,] <位置 1>, <位置 2>)

Description

将输入的<位置1>和 <位置2> 之间的移动量返回为Base坐标系基准值的函数.
<位置2>和 <位置1>应为相同类型的位置型变量. 此外, 代入Base坐标系位置型变量时<位置1>和tool坐标系编号应相同, 代入用户坐标系位置型变量时<位置1>和Tool坐标系编号及用户坐标系编号应相同.

1) 输入<Robot编号>保存JOB程序时, 可以选择非选择的其它Robot.
2) <位置1>可以使用所有位置型变量. 输入常数或其它变量时发出Alarm.
3) <位置2>与 <位置1>的变量类型或坐标系编号不同时发生Alarm, 请注意使用.

Return value

<变量>	Data type: <Base坐标系 位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<位置1>	Data type: <所有坐标系 位置型变量>
<位置2>	Data type: <所有坐标系 位置型变量>

Examples

Example 1

```
MAIN                                    // 主函数声明
    POS AP1, AP2                        // Joint坐标系位置型变量声明
    XPOS DELTA                          // Base坐标系 位置型变量声明
    AP1 = GP0                           // GP0代入 AP1
    AP2 = GP1                           // GP1代入 AP2
    DELTA = GBASEDISP(AP1, AP2)         // 求AP1和AP2之间的Base坐标系移动量
    PRINT(0, DELTA)                     // 监控日志0中保存结果值
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数声明
    XPOS XP1, XP2, DELTA                // Base坐标系 位置型变量声明
    GP0.T = 1                           // 指定GP0的Tool坐标系1号
    XP1 = JTOB(GP0)                     // GP0 值转换为Base坐标系值
    GP1.T = 1                           // 指定GP1的Tool坐标系编号1号
    XP2 = JTOB(GP1)                     // GP1 值转换为Base坐标系值
    DELTA = GBASEDISP(XP1, XP2)         // 求XP1和 XP2之间的Base坐标系移动量
    PRINT(1,DELTA)                      // 监控日志1中保存结果值
    DELTA = GBASEDISP(1,XP1, XP2)       // 求robot 1的XP1和 XP2之间的
                                        //Base坐标系移动量
    PRINT(2,DELTA)                      // 监控日志2中保存结果值
EOP                                     // 主函数结束
```

Example 3

```
MAIN                                    // 主函数声明
    XPOS DELTA                          // Base坐标系 位置型变量声明
    UPOS UP1, UP2                       // 用户坐标系位置型变量声明
    GP0.T = 1                           // 指定GP0的Tool坐标系编号1号
    GP0.U = 1                           // 指定GP0的用户坐标系编号1号
    UP1 = JTOU(GP0)                     // GP0 值转换为用户坐标系值
    GP1.T = 1                           // 指定GP1的Tool坐标系编号1号
    GP1.U = 1                           // 指定GP1的用户坐标系编号1号
    UP2 = JTOU(GP1)                     // GP1 值转换为用户坐标系值
    DELTA = GBASEDISP(UP1,UP2)          // 求UP1和UP2之间的Base坐标系移动量
    PRINT(3,DELTA)                      // 监控日志3中保存结果值
    DELTA = GBASEDISP(1,UP1,UP2)        // 求Robot 1的 UP1和UP2之间的
                                        //Base坐标系移动量
    PRINT(4,DELTA)                      // 监控日志4中保存结果值
EOP                                     // 主函数结束
```

Related terms

GTOOLDISP, GUSERDISP

10.13 GTOOLDISP (Tool坐标系基准移动量检测)

Usage

求输入的两个坐标之间Tool坐标系基准移动量的函数.

Syntax

<变量> = GTOOLDISP([<Robot编号>,] <位置1>, <位置2>)

Description

将输入的<位置1>和 <位置2> 之间的移动量返回为Tool坐标系基准值的函数.
<位置2>和 <位置1>应为相同类型的位置型变量. 此外, 代入Base坐标系位置型变量时<位置1>和tool坐标系编号应相同, 代入用户坐标系位置型变量时<位置1>和Tool坐标系编号及用户坐标系编号应相同.

1) 输入<Robot编号>保存JOB程序时, 可以选择非选择的其它Robot.
2) <位置1>可以使用所有位置型变量. 输入常数或其它变量时发出Alarm.
3) <位置2>与 <位置1>的变量类型或坐标系编号不同时发生Alarm, 请注意使用.

Return value

<变量>	Data type: <Base坐标系 位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<位置1>	Data type: <所有坐标系 位置型变量>
<位置2>	Data type: <所有坐标系 位置型变量>

Examples

Example 1

```
MAIN                                    // 主函数声明
    POS AP1, AP2                        // Joint坐标系位置型变量声明
    XPOS DELTA                          // Base坐标系 位置型变量声明
    AP1 = GP0                           // GP0代入 AP1
    AP2 = GP1                           // GP1代入 AP2
    DELTA = GTOOLDISP(AP1, AP2)         // 求AP1和 AP2之间的Tool坐标系移动量
    PRINT(0, DELTA)                     // 监控日志0中保存结果值
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数声明
    XPOS XP1, XP2, DELTA                // Base坐标系 位置型变量声明
    GP0.T = 1                           // 指定GP0的 Tool坐标系编号1号
    XP1 = JTOB(GP0)                     // GP0 值转换为Base坐标系值
    GP1.T = 1                           // 指定GP1的Tool坐标系编号1号
    XP2 = JTOB(GP1)                     // GP1 值转换为Base坐标系值
    DELTA = GTOOLDISP(XP1, XP2)         // 求XP1和 XP2之间Tool坐标系移动量
    PRINT(1,DELTA)                      // 监控日志1中保存结果值
    DELTA = GTOOLDISP(1,XP1, XP2)       // 求Robot1的XP1和 XP2之间Tool
                                        //坐标系移动量
    PRINT(2,DELTA)                      // 监控日志2中保存结果值
EOP                                     // 主函数结束
```

Example 3

```
MAIN                                    // 主函数声明
    XPOS DELTA                          // Base坐标系 位置型变量声明
    UPOS UP1, UP2                       // 用户坐标系位置型变量声明
    GP0.T = 1                           // 指定GP0的Tool坐标系编号1号
    GP0.U = 1                           // 指定GP0的用户坐标系编号1号
    UP1 = JTOU(GP0)                     // GP0 值转换为用户坐标系值
    GP1.T = 1                           // 指定GP1的Tool坐标系编号1号
    GP1.U = 1                           // 指定GP1的用户坐标系编号1号
    UP2 = JTOU(GP1)                     // GP1 值转换为用户坐标系值
    DELTA = GTOOLDISP(UP1,UP2)          // 求UP1和UP2之间的Tool坐标系移动量
    PRINT(3,DELTA)                      // 监控日志3中保存结果值
    DELTA = GTOOLDISP(1,UP1,UP2)        // 求Robot1的UP1和 UP2之间的
                                        //Tool坐标系移动量
    PRINT(4,DELTA)                      // 监控日志4中保存结果值
EOP                                     // 主函数结束
```

Related terms

GBASEDISP, GUSERDISP

10.14 GUSERDISP (用户坐标系基准移动量检测)

Usage

求输入的两个坐标之间用户坐标系基准移动量的函数.

Syntax

<变量> = GUSERDISP([<Robot编号>,] <位置1>, <位置2>)

Description

将输入的<位置1>和 <位置2> 之间的移动量返回为用户坐标系基准值的函数.
<位置2>和 <位置1>应为相同类型的位置型变量. 此外, 代入Base坐标系位置型变量时<位置1>和tool坐标系编号应相同, 代入用户坐标系位置型变量时<位置1>和Tool坐标系编号及用户坐标系编号应相同.

1) 输入<Robot编号>保存JOB程序时, 可以选择非选择的其它Robot.
2) <位置1>可以使用所有位置型变量. 输入常数或其它变量时发出Alarm.
3) <位置2>与 <位置1>的变量类型或坐标系编号不同时发生Alarm, 请注意使用.

Return value

<变量>	Data type: <用户坐标系位置型变量>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<位置1>	Data type: <所有坐标系 位置型变量>
<位置2>	Data type: <所有坐标系 位置型变量>

Examples

Example 1

```
MAIN                                    // 主函数声明
    POS AP1, AP2                        // Joint坐标系位置型变量声明
    UPOS DELTA                          // 用户坐标系位置型变量声明
    AP1 = GP0                           // GP0代入AP1
    AP2 = GP1                           // GP1代入AP2
    DELTA = GUSERDISP(AP1, AP2)         // 求GP0和 GP1之间用户坐标系移动量
    PRINT(0, DELTA)                     // 监控日志0中保存结果值
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数声明
    XPOS XP1, XP2                       // Base坐标系 位置型变量声明
    UPOS DELTA                          // 用户坐标系位置型变量声明
    GP0.T = 1                           // 指定GP0的 Tool坐标系编号1号
    XP1 = JTOB(GP0)                     // GP0 值转换为Base坐标系值
    GP1.T = 1                           // 指定GP1的Tool坐标系编号1号
    XP2 = JTOB(GP1)                     // GP1 值转换为 Base坐标系值
    DELTA = GUSERDISP(XP1, XP2)         // 求XP1和 XP2之间用户坐标系移动量
    PRINT(1,DELTA)                      // 监控日志1中保存结果值
EOP                                     // 主函数结束
```

Example 3

```
MAIN                                    // 主函数声明
    UPOS UP1, UP2, DELTA                // 用户坐标系位置型变量声明
    GP0.T = 1                           // 指定GP0的 Tool坐标系编号1号
    GP0.U = 1                           // 指定GP0的 用户坐标系编号1号
    UP1 = JTOU(GP0)                     // GP0值转换为用户坐标系值
    GP1.T = 1                           // 指定GP1的 Tool坐标系编号1号
    GP1.U = 1                           // 指定GP1的 用户坐标系编号1号
    UP2 = JTOU(GP1)                     // GP1值转换为用户坐标系值
    DELTA = GUSERDISP(UP1,UP2)          // 求UP1和 UP2之间户坐标系移动量
    PRINT(2,DELTA)                      // 监控日志2中保存结果值
EOP                                     // 主函数结束
```

Related terms

GBASEDISP, GTOOLDISP

10.15 ABS (绝对值)

Usage

返回输入值的绝对值的函数.

Syntax

<变量> = ABS(<输入值>)

Description

返回零点位置到输入值的位移大小.
即, 输入整数型数据时返回正(+) 整数值, 输入实数型数据时返回正 (+) 实数值.

1) <输入值>输入整数型常数和变量或实数型常数和变量.

Return value

<变量>	Data type: <整数或实数型> 返回位移大小.

Arguments

<输入值>	Data type: <整数或实数型>

Examples

Example 1

```
INT AA,BB                   // 声明整数型变量
AA = -5                     // 整数型变量AA初始化为-5
IO = ABS(-5)                // 全局整数型变量IO中保存 5
BB = ABS(AA)                // 整数型变量BB中保存 5
```

Example 2

```
REAL AA, BB                 // 声明实数型变量
AA = -1.5                   // 实数型变量AA初始化为-1.5
FO = ABS(-1.5)              // 全局实数型变量FO中保存 1.5
BB = ABS(AA)                // 实数型变量BB中保存 1.5
```

10.16 DEG (弧度角转换为角度值)

Usage

弧度角 (弧度法)转换为角度值 (六十分法)的函数.

Syntax

<变量> = DEG(<输入值>)

Description

弧度角换算为角度值时使用的函数.
使用以下角度转换公式.

$$degree = (radian) \times \left(\frac{180^{\circ}}{\pi}\right)$$

1) <输入值>输入弧度角.

Return value

<变量>	Data type: <实数型> 结果值的单位为角度 (°).

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 1                  // 整数型变量AA中分配弧度角(1)
F0 = DEG(1)             // 将正常数 (1)变为角度值 (57.29,578)保存
BB = DEG(AA)            // 将整数变量 (AA)变为角度值 (57.29,578)保存
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 3.1416             // 实数型变量AA中分配弧度角 (3.1416)
F0 = DEG(3.1416)        // 将实数常数 (3.1416)变更为角度值 (180)保存
BB = DEG(AA)            // 将实数型变量 (AA)变更为角度值 (180)保存
```

Related terms

RAD

※ 主要角度的六十分法和弧度法

六十分法	0°	30°	45°	60°	90°	180°	270°	360°
弧度法	0	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	π	$\frac{3}{2}\pi$	2π

10.17 RAD（角度值变为弧度值）

Usage

角度值(六十分法)转换为弧度角 (弧度法)的函数.

Syntax

<变量> = RAD(<输入值>)

Description

角度值换算为弧度角时使用的函数.
使用如下弧度角转换公式.

$$radian = (degree) \times \left(\frac{\pi}{180^\circ}\right)$$

1) <输入值>输入角度值 (60分法).

Return value

<变量>	Data type: <实数型> 结果值单位为弧度角 (rad).

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 180                // 整数型变量AA中分配角度(180)
F0 = RAD(180)           // 整数常数 (180)变为弧度角 (3.14,159)保存
BB = RAD(AA)            // 整数变量 (AA)变为弧度角 (3.14,159)保存
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 180.0              // 实数型变量AA中分配角度 (180.0)
F0 = RAD(180.0)         // 实数常数 (180.0)变更为弧度角 (3.14,159)保存
BB = RAD(AA)            // 实数变量 (AA)变为弧度角 (3.14,159)保存
```

Related terms

DEG

※ 度主要角的六十分法和弧度法

六十分法	0°	30°	45°	60°	90°	180°	270°	360°
弧度法	0	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	π	$\frac{3}{2}\pi$	2π

10.18 POW (幂函数)

Usage

求乘方值时使用的函数.

Syntax

<变量> = POW(<底数值>, <指数值>)

Description

求乘方(x^y) 值的函数.
例如，求2的5方时返回$2^5 = 32$ 值.

1) <底数值>输入作为乘方底数的值.
2) <指数值>中输入乘方的指数值.

Return value

<变量> Data type: <实数型>
返回乘方的结果值.
底数值小于0且指数值非整数时返回NaN(Not-A-Number).
底数值等于0且指数值小于0时返回无穷值(Inf).

Arguments

<底数值> Data type: <整数型或实数型>

<指数值> Data type: <整数型或实数型>

Examples

Example 1

```
INT AA, BB                  // 声明整数型变量
REAL CC                     // 声明实数型变量
AA = 2                      // 底数值设置为2
BB = 5                      // 指数值设置为5
F0 = POW(2, 5)              // 整数常数 (2^5)保存到F0
CC = POW(AA, BB)            // 整数变量 (AA^BB) 保存到 CC
```

Example 2

```
REAL AA, BB, CC             // 声明实数型变量
AA = 2.0                    // 底数值设置为2.0
BB = 5.0                    // 底数值设置为5.0
F0 = POW(2.0, 5.0)          // 实数常数(2^5) 保存到 F0
CC = POW(AA, BB)            // 实数变量(AA^BB) 保存到CC
```

Related terms

SQRT

10.19 RND（四舍五入）

Usage

小数部分四舍五入后返回整数值.

Syntax

<变量> = RND(<输入值>)

Description

四舍五入小数点以下值后返回整数型值.
例如，输入12.8时返回 13.
使用以下变换公式.

$$result = (input\ value) + 0.5$$

1) <输入值>中输入四舍五入的对象.

Return value

<变量>	Data type: <整数型> 返回执行四舍五入的结果值.

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA, BB                  // 声明整数型变量
AA = 12                     // 整数型变量赋值12
IO = RND(12)                // 输入整数常数(12)时返回12
BB = RND(AA)                // 输入整数变量(12)时返回 12
```

Example 2

```
REAL AA                     // 声明实数型变量
INT BB                      // 声明整数型变量
AA = 12.5                   // 实数变量赋值12.5
IO = RND(12.5)              // 输入实数常数 (12.5)时返回 13
BB = RND(AA)                // 输入实数变量(AA = 12.5)时返回13
```

Example 3

```
REAL AA                     // 声明实数型变量
AA = RND(49.7/0.125)        // 返回输入公式结果值的四舍五入值
```

10.20 EXP (指数 e^x)

Usage

返回无理数 e^x 的幂函数值. 即为对数的逆函数.

Syntax

<变量> = EXP(<输入值>)

Description

无理数为 e = 2.7182,818,284,590,452,353,602,874,713 ..., 返回 e^x的结果值.
例如, EXP(1) = 2.718,282 时
EXP(2) = 2.718,282 * 2.718,282 = 7.389,056
EXP(3) = 2.718,282 * 2.718,282 * 2.718,282 = 20.08,554.

1) <输入值>中输入无理数e^x 的指数x值.

Return value

<变量>	Data type: <实数型> 返回无理数e^x 的结果值.

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1	INT AA
	REAL BB
	AA = 2
	F0 = EXP(2)
	BB = EXP(AA)
		// 声明整数型变量
		// 声明实数型变量
		// 整数型变量赋值指数值2
		// e^2的结果值保存到全局变量 F0
		// $e^{(AA=2)}$的结果值保存到变量 BB

Example 2	REAL AA, BB
	AA = 2.0
	F0 = EXP(2.0)
	BB = EXP(AA)
		// 声明实数型变量
		// 实数型变量赋值指数值2
		// $e^{2.0}$的结果值保存到全局变量 F0
		// $e^{(AA=2.0)}$的结果值保存到变量 BB

Related terms

POW, LN, LOG

10.21 LN（自然对数）

Usage

返回自然对数$\log_e x$的结果值.

Syntax

<变量> = LN(<输入值>)

Description

求自然对数(Natural Logarithm)值的函数.
例如, LN(5) = 1.704,748.
使用如下转换公式.

$$y = \ln x = \log_e x$$

1) <输入值>中输入自然对数公式的输入值x值.

Return value

<变量>	Data type: <实数型> 返回结果值y.

Arguments

<输入值>	Data type: <整数型 或实数型>

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 15                 // 整数型变量赋值15
F0 = LN(15)             // log_e x 值(2.708) 保存到全局变量 F0
BB = LN(AA)             // log_e x 值(2.708) 保存到BB
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 15.0               // 实数变量中赋指数值15.0
F0 = LN(15.0)           // log_e x 值(2.708) 保存到全局变量 F0
BB = LN(AA)             // log_e x 值(2.708) 保存到实数变量BB
```

Related terms

POW, EXP, LOG

10.22 LOG (常用对数)

Usage

返回常用对数$\log_{10} x$的结果值.

Syntax

<变量> = LOG(<输入值>)

Description

求常用对数 (Common Logarithm)值的函数.
例如, $LOG(10) = 1.0$, $LOG(100) = 2.0$, $LOG(1000) = 3.0$.
使用如下转换公式.

$y = \log_{10} x$

1) <输入值> 输入常用对数公式中的输入值x值.

Return value

<变量>	Data type: <实数型> 返回结果值y.

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 100                // 整数型变量赋值100
F0 = LOG(100)           // log10 x的结果值(2.0) 保存到变量 F0
BB = LOG(AA)            // log10 x的结果值(2.0) 保存到变量BB
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 100.0              // 实数型变量赋指数值100.0
F0 = LOG(100.0)         // log10 x的结果值(2.0) 保存到变量 F0
BB = LOG(AA)            // log10 x的结果值(2.0) 保存到变量 BB
```

Related terms

POW, EXP, LN

10.23 SQRT (平方根)

Usage

返回平方根.

Syntax

<变量> = SQRT(<输入值>)

Description

返回输入值 (x)的平方根(y).
例如, SQRT(1) = 1.00,000, SQRT(2) = 1.41,421, SQRT(3) = 1.73,205, SQRT(4) = 2.00,000.
使用以下转换公式.

$$y = \sqrt{x}$$

1) <输入值> 输入平方根公式中输入值x.

Return value

<变量>	Data type: <实数型> 返回结果值y. 输入值为负数时返回NaN(Not-A-Number).

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 4                  // 整数型变量赋平方根号输入值4
F0 = SQRT(4)            // √4的结果值2保存到全局变量 F0
BB = SQRT(AA)           // √(AA = 4)的结果值保存到变量 BB
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 4.0                // 实数型变量变量赋平方根号输入值 4.0
F0 = SQRT(4.0)          // √4.0的结果值2保存到全局变量F0
BB = SQRT(AA)           // √(AA = 4.0)的结果值2保存到变量BB
```

Related terms

POW

10.24 SIN（正弦函数）

Usage

返回正弦结果值.

Syntax

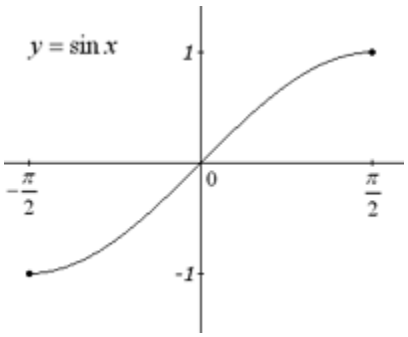
<变量> = SIN(<输入值>)

Description

返回$\sin(x)$ 函数的结果值y.

$y = \sin(x)$

例如，三角函数为特殊角时
$\mathrm{SIN}(\mathrm{RAD}(0)) = 0.0,$
$\mathrm{SIN}(\mathrm{RAD}(30)) = 0.5,$
$\mathrm{SIN}(\mathrm{RAD}(45)) = 0.70{,}711,$
$\mathrm{SIN}(\mathrm{RAD}(60)) = 0.86{,}603,$
$\mathrm{SIN}(\mathrm{RAD}(90)) = 1.$



1) <输入值>输入正弦函数公式的 x值. (x 单位:$radian$)

Return value

<变量>	Data type: <实数型> 返回结果值y. 结果值的范围为 $-1 \le y \le 1$.

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA                      // 声明整数型变量
REAL BB                     // 声明实数型变量
AA = 30                     // 整数型变量赋特殊角 (30)
F0 = SIN(RAD(30))           // sin(30°)的结果值0.5保存到 F0
BB = SIN(RAD(AA))           // 结果值0.5赋给实数型变量BB
```

Example 2

```
REAL AA, BB                 // 声明实数型变量
AA = 30.0                   // 实数型变量赋特殊角(30)
F0 = SIN(RAD(30.0))         // sin(30°)的结果值0.5保存到F0
BB = SIN(RAD(AA))           // 结果值0.5赋给实数型变量BB
```

Related terms

ASIN, COS, ACOS, TAN, ATAN, ATAN2

10.25 ASIN（反正弦函数）

Usage

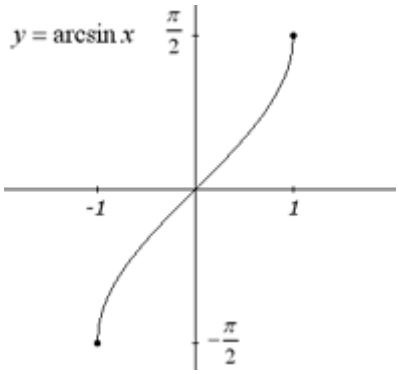
返回反正弦函数的结果值.

Syntax

<变量> = ASIN(<输入值>)

Description

asin(x) 函数为 sin(x)的逆函数， 返回 $-\frac{\pi}{2} \sim \frac{\pi}{2}$ 弧度范围内的asin值. 默认x小于 -1或大于 1时 asin 返回无穷值.



例如,
ASIN$(-1) = -1.5708$,
ASIN$(0) = 0.0$,
ASIN$(1) = 1.5708$.

1) <输入值>输入反正弦函数的输入值 x值. $(-1 \le x \le 1)$

Return value

<变量> Data type: <实数型>
返回结果值y. 结果值的范围为 $-\frac{\pi}{2} \le y \le \frac{\pi}{2}$.

Arguments

<输入值> Data type: <整数型或实数型>
输入值的范围为 $-1 \le x \le 1$.

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 1                  // 整数型变量赋输入值1
F0 = ASIN(1)            // asin(1)的结果值 1.5708保存到 F0
BB = ASIN(AA)           // 结果值赋给实数型变量BB
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 1.0                // 实数型变量赋输入值1.0
F0 = ASIN(1.0)          // asin(1)的结果值1.5708保存到F0
BB = ASIN(AA)           // 结果值赋给实数型变量BB
```

Related terms

SIN, COS, ACOS, TAN, ATAN, ATAN2

10.26 COS（余弦函数）

Usage

返回余弦函数的结果值.

Syntax

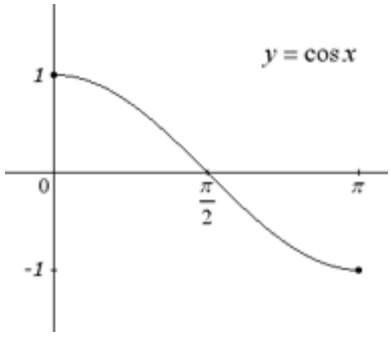
<变量> = COS(<输入值>)

Description

返回$\cos(x)$ 函数的结果值.

$y = \cos(x)$

例如, 三角函数特殊值时
$\mathrm{COS(RAD(0))} = 1.0,$
$\mathrm{COS\big(RAD(30)\big)} = 0.86{,}603,$
$\mathrm{COS\big(RAD(45)\big)} = 0.70{,}711,$
$\mathrm{COS\big(RAD(60)\big)} = 0.5.$
$\mathrm{COS\big(RAD(90)\big)} = 0.0.$



1) <输入值>中输入余弦函数公式中的x值. (x 单位:$radian$)

Return value

<变量>	Data type: <实数型> 返回结果值y. 结果值的范围为 $-1 \le y \le 1$.

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 60                 // 整数型变量赋值特殊角 (60)
F0 = COS(RAD(60))       // cos(60°)的结果值0.5保存到 F0
BB = COS(RAD(AA))       // 结果值0.5赋给实数型变量BB
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 60.0               // 实数型变量赋值特殊角(60)
F0 = COS(RAD(60.0))     // cos(60°)的结果值0.5保存到F0
BB = COS(RAD(AA))       // 结果值0.5赋给实数型变量BB
```

Related terms

SIN, ASIN, ACOS, TAN, ATAN, ATAN2

10.27 ACOS（反余弦函数）

Usage

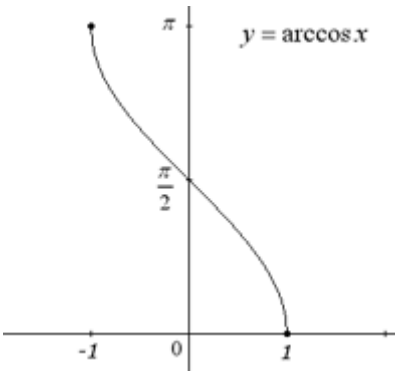
返回反余弦函数结果值.

Syntax

<变量> = ACOS(<输入值>)

Description

acos(x) 函数为 cos(x)函数的逆函数， 返回0 ～ π弧度范围内的 acos值. 默认x小于 -1或大于 1时acos返回无穷值.



例如,
ACOS(-1) = 3.14,159.
ACOS(0) = 1.5708.
ACOS(1) = 0.0.

1) <输入值>输入反反弦函数的输入值 x值. ($-1 \le x \le 1$)

Return value

<变量>	Data type: <实数型> 返回结果值y. 结果值的范围为$0 \le y \le \pi$.

Arguments

<输入值>	Data type: <整数型或实数型> 输入值的范围为$-1 \le x \le 1$.

Examples

Example 1	INT AA
	REAL BB
	AA = 0
	F0 = ACOS(0)
	BB = ACOS(AA)
		// 声明整数型变量
		// 声明实数型变量
		// 整数型变量赋输入值0
		// acos(0)的结果值 1.5708保存到 F0
		// 结果值赋给实数型变量BB

Example 2	REAL AA, BB
	AA = 0.0
	F0 = ACOS(0.0)
	BB = ACOS(AA)
		// 声明实数型变量
		// 实数型变量赋输入值0.0
		// acos(0)的结果值1.5708保存到F0
		// 结果值赋给实数型变量BB

Related terms

SIN, ASIN, COS, TAN, ATAN, ATAN2

10.28 TAN（正切函数）

Usage

返回正切函数的结果值.

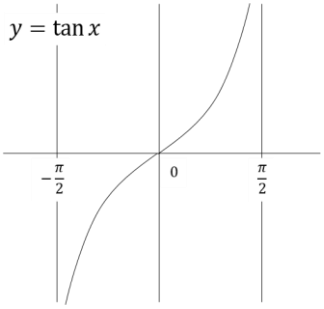
Syntax

<变量> = TAN(<输入值>)

Description

返回tan(x) 函数的结果值.

$y = \tan(x)$



例如，三角函数特殊值时
TAN(RAD(0)) = 0.0，TAN(RAD(30)) = 0.57,735,
TAN(RAD(45)) = 1.0，TAN(RAD(60)) = 1.73,205,
TAN(RAD(90)) = ＊（超出显示范围）.

1） <输入值>中输入正切函数公式中的x值.（x 单位: $radian$）

Return value

<变量>	Data type: <实数型> 返回结果值y.

Arguments

<输入值>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 45                 // 整数型变量赋值特殊角(45)
F0 = TAN(RAD(45))       // tan(45°)的结果值1.0保存到F0
BB = TAN(RAD(AA))       // 结果值1.0赋给实数型变量BB
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 45.0               // 实数型变量赋值特殊角(45)
F0 = TAN(RAD(45.0))     // tan(45°)的结果值1.0保存到F0
BB = TAN(RAD(AA))       // 结果值1.0赋给实数型变量BB
```

Related terms

SIN, ASIN, COS, ACOS, ATAN, ATAN2

1） 如tan(90°)无限发散时数据值大，难以在控制器上定量显示，因此Teaching Pendant上 “*”显示. 请留意这点.

10.29 ATAN（反正切函数）

Usage

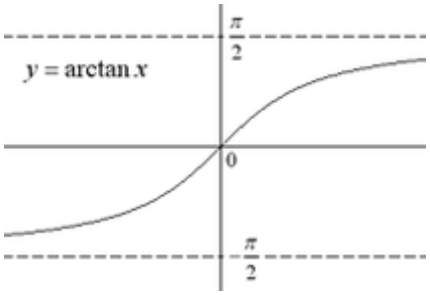
返回反正切函数的结果值.

Syntax

〈变量〉 = ATAN(〈输入值〉)

Description

$\text{atan}(x)$ 函数为 $\tan(x)$ 函数的逆函数，返回结果值.



例如,
$\text{ATAN}(-1) = -0.7854$, $\text{ATAN}(0) = 0.0$,
$\text{ATAN}(1) = 0.7854$.

1) 〈输入值〉输入反正切函数公式中的x值.
(x 单位:$radian$)

Return value

〈变量〉 Data type:〈实数型〉
返回结果值y. 结果值的范围为$-\frac{\pi}{2} \le y \le \frac{\pi}{2}$.

Arguments

〈输入值〉 Data type:〈整数型或实数型〉

Examples

Example 1

```
INT AA                  // 声明整数型变量
REAL BB                 // 声明实数型变量
AA = 1                  // 整数型变量赋值 1
F0 = ATAN(1)            // atan(1) 的结果值 0.7854保存到 F0
BB = ATAN(AA)           // 结果值赋给实数型变量BB
```

Example 2

```
REAL AA, BB             // 声明实数型变量
AA = 1.0                // 数型变量赋值 1.0
F0 = ATAN(1.0)          // atan(1) 的结果值0.7854保存到 F0
BB = ATAN(AA)           // 结果值赋给实数型变量BB
```

Related terms

SIN, ASIN, COS, ACOS, TAN, ATAN2

10.30 ATAN2(第2反正切函数)

Usage

返回第2反正切函数的结果值.

Syntax

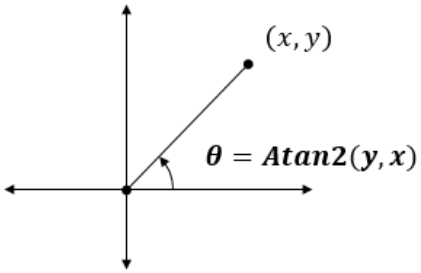
<返回值> = ATAN2(<Y 比例>, <X 比例>)

Description

返回atan2(x) 函数的结果值.

$y = \text{atan2}(x)$

例如,
ATAN2(1.0, 1.0) = 0.7854,
ATAN2(2.0, 1.0) = 1.10,715,
ATAN2(1.0, 2.0) = 0.46,365.



1) <Y 比例>中输入直角三角形高度 (y轴) 值.
2) <X 比例>中输入直角三角形的宽度 (x轴) 值.

Return value

<变量>	Data type: <实数型> 返回结果值y. 结果值的范围为 $-\pi \le y \le \pi$.

Arguments

<Y 比例>	Data type: <整数型或实数型>
<X 比例>	Data type: <整数型或实数型>

Examples

Example 1

```
INT AA, BB                  // 声明整数型变量
REAL CC                     // 声明实数型变量
AA = 1                      // 整数型变量赋值 1
BB = 1                      // 整数型变量赋值 1
F0 = ATAN2(1,1)             // 保存atan2(1,1)的结果值 0.7854
CC = ATAN2(AA,BB)           // 结果值赋给实数型变量 CC
```

Example 2

```
REAL AA, BB,CC              // 声明实数型变量
AA = 1.0                    // 实数型变量赋值1.0
BB = 1.0                    // 实数型变量赋值1.0
F0 = ATAN2(1.0,1.0)         // 保存atan2(1.0,1.0) 的结果值 0.7854
CC = ATAN2(AA,BB)           // 结果值赋给实数型变量CC
```

Related terms

SIN, ASIN, COS, ACOS, TAN, ATAN

10.31 MIN（最小值）

Usage

比较输入的两个值，返回较小值.

Syntax

<变量> = MIN(<输入值1>, <输入值2>)

Description

比较输入值1和输入值2，以实数型类型返回最小值.

1) <输入值1> 输入比较最小值的值1.
2) <输入值2> 输入比较最小值的值2.

Return value

<变量>	Data type: <实数型> 结果值返回两个输入值中的最小值.

Arguments

<输入值1>	Data type: <整数型 或实数型>
<输入值2>	Data type: <整数型 或实数型>

Examples

Example 1

```
INT AA, BB                  // 声明整数型变量
REAL CC                     // 声明实数型变量
AA = 2                      // 整数型比较变量1赋值2
BB = 5                      // 整数型比较变量2赋值5
F0 = MIN(2,5)               // 返回的最小值2.0保存到F0
CC = MIN(AA,BB)             // 最小值2.0保存到实数型变量CC
```

Example 2

```
REAL AA, BB, CC             // 声明实数型变量
AA = 2.0                    // 实数型比较变量1赋值 2.0
BB = 2.5                    // 实数型比较变量2赋值2.5
F0 = MIN(2.0,2.5)           // 返回的最小值2.0保存到F0
CC = MIN(AA,BB)             // 最小值2.0保存到实数型变量CC
```

Related terms

MAX

10.32 MAX (最大值)

Usage

比较输入的两个值，返回最大值.

Syntax

<变量> = MAX(<输入值1>, <输入值2>)

Description

比较输入值1和输入值2，以实数型类型返回最大值.

1) <输入值1> 输入比较最大值的值1.
2) <输入值2> 输入比较最大值的值2.

Return value

<变量>	Data type: <实数型> 结果值返回两个输入值中的最大值.

Arguments

<输入值1>	Data type: <整数型 或实数型>
<输入值2>	Data type: <整数型 或实数型>

Examples

Example 1

```
INT AA, BB                  // 声明整数型变量
REAL CC                     // 声明实数型变量
AA = 2                      // 整数型比较变量1赋值2
BB = 5                      // 整数型比较变量2赋值5
F0 = MAX(2,5)               // 返回的最大值5.0保存到 F0
CC = MAX(AA,BB)             // 最大值 5.0保存到实数型变量CC
```

Example 2

```
REAL AA, BB, CC             // 声明实数型变量
AA = 2.0                    // 实数型比较变量1赋值 2.0
BB = 2.5                    // 实数型比较变量2赋值 2.5
F0 = MAX(2.0,2.5)           // 返回的最大值2.5保存到 F0
CC = MAX(AA,BB)             // 最大值2.5保存到实数型变量CC
```

Related terms

MIN

10.33 RAND (随机数生成)

Usage

生成随机数.

Syntax

〈变量〉 = RAND()

Description

随机数函数生成0到RAND_MAX 之间的随机数. 32bit 位Linux系统中 RAND_MAX值为2,147,483,647. 即, 整数型变量的正最大值

- 1到 10 之间随机数生成方法: RAND()%10+1
- 1到 100 之间随机数生成方法: RAND()%100+1
- 1到 1,000 之间随机数生成方法: RAND()%1000+1

Return value

〈变量〉 Data type: 〈整数型〉
结果值的范围为 0 ~ RAND_MAX.

Examples

Example 1

```
REAL AA, BB                 // 声明实数型变量
WHILE 1                     // 无限执行WHILE ~ ENDWL 块
    AA = RAND()             // 随机值(0 ~ 7FFFFFFF)保存到AA
    BB = AA%100+1           // 生成1 ~ 100 之间随机值保存到BB
    DLAY 100                // 时间延迟 0.1秒
ENDWL
```

Example 2

```
INT AA                      // 声明整数型变量
AA = ((RAND())%100)+1       // 生成1 ~ 100 之间随机值保存到 AA
```

10.34 ASC (返回字符串第一个字母的字符码)

Usage

返回字符串第一个字母的字符码整数值.

Syntax

<变量> = ASC(<字符串>)

Description

返回输入的字符串第一个文字.

字符码(Character Code)表示文字对应的数字(整数).
例) 文字’A’为(65), 文字 ‘B’为整数(66).
请参照ASCII码表.

1) <字符串>输入检测第一个文字的字符串.

Return value

<变量>	Data type: <整数型> 结果值的范围为 0 ~ 127.

Arguments

<字符串>	Data type: <字符串>

Examples

Example 1

```
MAIN                                    // 主函数开始
    IO = ASC(” STATEMENT”)              // 第一个字母 “S”转换为ASCII码 (83)
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数开始
    DEFSTR STR                          // 字符串变量声明
    INT VAL                             // 声明整数型变量
    STR = “STATEMENT”                   // 字符串变量 STR中代入 “STATEMENT”
    VAL = ASC(STR)                      // 第一个字母 “S”转换为ASCII码(83)
EOP                                     // 主函数结束
```

Related terms

CHR

10.35 BINS（转换为2进制字符串）

Usage

将输入的整数值转换为2进制字符串的函数.

Syntax

<变量> = BINS(<输入值>)

Description

将输入的整数值转换为2进制值后返回成32 字节字符串.

1) <输入值>输入变更为2进制字符串的整数值.

Return value

<变量>	Data type: <字符串> 返回输入值的2进制字符串.

Arguments

<输入值>	Data type: <整数型> 可以输入-2^{31}(-2,147,483,648) ~ 2^{31}(2,147,483,647).

Examples

Example 1

```
MAIN                        // 主函数开始
    DEFSTR STR1,STR2        // 字符串变量声明
    INT AA                  // 声明整数型变量
    AA = 16                 // 整数型 变量初始化
    STR1 = BINS(16)         //16的2进制值对应字符串
    STR2 = BINS(AA)         //“00000000000000000000000000010000” 保存
                            //到STR 变量
EOP                         // 主函数结束
```

Example 2

```
MAIN                        // 主函数开始
    DEFSTR STR1,STR2        // 字符串变量声明
    REAL AA                 // 声明实数型变量
    AA = 32.0               // 实数型变量初始化
    STR1 = BINS(32.0)       //32的2进制值对应字符串
    STR2 = BINS(AA)         //“00000000000000000000000000100000”保存
                            //到STR变量
EOP                         // 主函数结束
```

Related terms

FTOS, HTOS, SVAL

10.36 CHR (整数ASCII文字转换)

Usage

返回输入的整数值对应的ASCII文字.

Syntax

<变量> = CHR(<输入值>)

Description

返回输入的整数对应的ASCII字母.

1) <输入值>输入ASCII值内的整数 (0 ~ 127).

Return value

<变量>	Data type: <字符串>

Arguments

<输入值>	Data type: <整数型>

Examples

Example 1

```
MAIN                                    // 主函数开始
    DEFSTR STRA,STRB                    // 字符串变量声明
    INT AA                              // 声明整数型变量
    AA = 66                             // 整数型 变量初始化
    STRA = CHR(65)                      // STRA赋值ASCII码(65)对应的 “A”
    STRB = CHR(AA)                      // STRA赋值ASCII码(66) 对应的“B”
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数开始
    DEFSTR STRA,STRB                    // 字符串变量声明
    REAL AA                             // 声明整数型变量
    AA = 66.0                           // 整数型 变量初始化
    STRA = CHR(65.0)                    // STRA赋值ASCII码(65) 对应的“A”
    STRB = CHR(AA)                      // STRA赋值ASCII码(66) 对应的“B”
EOP                                     // 主函数结束
```

Related terms

ASC

10.37 FLUSH (清除通信输入输出缓冲)

Usage

清除通信输入/输出缓冲.

Syntax

FLUSH <端口编号> <缓冲索引>

Description

清除通信收发信时使用的输入输出缓冲.

<端口编号>的范围为 0 ~ 65,535, 其含义如下.
- 0 : RS-232 串口通信端口
- 10 ~ 12 : TCP I/P Server 端口
  (10 : 端口编号[38201], 11 : 端口编号[38202], 12 : 端口编号[38203])
- ~ 65,535: TCP I/P Client 端口

<缓冲索引>的范围为 1 ~ 2 , 其含义如下.
- 1: 清除输入缓存
- 2: 清除输出缓存
- 3: 清除输入, 输出缓存

Arguments

<端口编号>	Data type: <整数型> <缓冲索引>的范围为0 ~ 65,535.
<缓冲索引>	Data type: <整数型> <缓冲索引>的范围为1 ~ 3.

Examples

Example 1

```
MAIN                            // 开始主函数
    DEFSTR STR                  // 字符串变量声明
    STR = STRIN(0,1000)         // 接收RS-232 串口通信字符串(接收等待1秒)
    FLUSH 1                     // 清除输入缓存
EOP                             // 结束主函数
```

Example 2

```
MAIN                            // 开始主函数
    DEFSTR STR                  // 字符串变量声明
    INT AA                      // 整数型变量声明
    AA = 10                     // 整数型变量初始化
    STR = STRIN(10,1000)        // 接收TCP I/P Server 通信字符串 (接收等待1秒)
    FLUSH AA 1                  // 清除输入缓存
EOP                             // 结束主函数
```

10.38 FTOS (整数/实数值转换为字符串)

Usage

整数或实数值转换为字符串.

Syntax

<变量> = FTOS(<输入值>)

Description

整数或实数数据转换为字符串.

例) FTOS(1234) → “1234”
例) FTOS(-123.45) → “-123.45”

Return value

<变量>	Data type: <字符串>

Arguments

<输入值>	Data type: <整数型 或实数型>

Examples

Example 1

```
MAIN                                    // 主函数开始
    DEFSTR STR1,STR2                    // 字符串变量声明
    INT AA                              // 声明整数型变量
    AA = 12345                          // 整数型 变量初始化
    STR1 = FTOS(1234)                   // 1234转换为字符串 “1234”后保存
    STR2 = FTOS(AA)                     // 12345转换为字符串“12345” 后保存
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数开始
    DEFSTR STR1,STR2                    // 字符串变量声明
    REAL AA                             // 声明实数型变量
    AA = -123.45                        // 实数型变量初始化
    STR1 = FTOS(-12.345)                // -12.345转换为字符串“-12.345” 后保存
    STR2 = FTOS(AA)                     // -123.45转换为字符串“-123.45” 后保存
EOP                                     // 主函数结束
```

Related terms

BINS, HTOS, SVAL

10.39 HTOS (整数转换为16进制字符串)

Usage

整数转换为16进制字符串.

Syntax

<变量> = HTOS(<输入值>)

Description

将输入的整数转换为16进制字符串.

例)HTOS(10) → “A”
例)HTOS(11) → “B”
例)HTOS(12) → “C”
例)HTOS(13) → “D”
例)HTOS(14) → “E”
例)HTOS(15) → “F”

Return value

<变量> Data type: <字符串>.

Arguments

<输入值> Data type: <整数型>

Examples

Example 1

```
MAIN                          // 主函数开始
    DEFSTR STR1,STR2          // 字符串变量声明
    INT AA                    // 声明整数型变量
    AA = 11                   // 整数型 变量初始化
    STR1 = HTOS(10)           // 整数10转换为16进制字符串 (“A”)
    STR2 = HTOS(AA)           // 整数11转换为16进制字符串(“B”)
EOP                           // 主函数结束
```

Example 2

```
MAIN                          // 主函数开始
    DEFSTR STR1,STR2          // 字符串变量声明
    REAL AA                   // 声明实数型变量
    AA = 13.0                 // 实数型变量初始化
    STR1 = HTOS(12.0)         // 整数12.0转换为16进制字符串(“C”)
    STR2 = HTOS(AA)           // 整数13.0转换为16进制字符串(“D”)
EOP                           // 主函数结束
```

Related terms

BINS, FTOS, SVAL

10.40 SLEFT (提取左侧字符串)

Usage

提取输入的字符串左侧字符串.

Syntax

<变量> = SLEFT(<字符串>, <提取个数>)

Description

从输入的字符串从左侧提取输入的整数个数.

1) <字符串>为提取字符串的对象.
2) <提取个数>为从字符串左侧提取字符串的个数.

Return value

<变量>	Data type: <字符串> 返回提取选择的字符串个数的字符串.

Arguments

<字符串>	Data type: <字符串>
<提取个数>	Data type: <整数型>

Examples

Example 1

```
MAIN                                    // 主函数开始
    DEFSTR STR0,STR1,STR2               // 字符串变量声明
    INT AA                              // 声明整数型变量
    STR0 = “TEST”                       // 字符串变量赋值 “TEST”
    AA = 2                              // 整数型 变量初始化
    STR1 = SLEFT(“TEST”, 1)             // STR1 字符串变量中保存 “T”
    STR2 = SLEFT(STR0, AA)              // STR2 字符串变量中保存 “TE”
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数开始
    DEFSTR STR0,STR1,STR2               // 字符串变量声明
    REAL AA                             // 声明实数型变量
    STR0 = “TEST”                       // 字符串变量赋值 “TEST”
    AA = 4.0                            // 实数型变量初始化
    STR1 = SLEFT(“TEST”, 3.0)           // STR1 字符串变量中保存 “TES”
    STR2 = SLEFT(STR0, AA)              // STR2 字符串变量中保存 “TEST”
EOP                                     // 主函数结束
```

Related terms

SLEN, SMID, SPOS, SRIGHT

10.41 SLEN (返回字符串长度)

Usage

返回输入的字符串长度.

Syntax

<变量> = SLEN(<字符串>)

Description

返回输入的字符串长度.

1) <字符串>输入测量长度值的字符串.

Return value

<变量>	Data type: <整数型> 返回输入的字符串长度.

Arguments

<字符串>	Data type: <字符串>

Examples

Example 1

```
MAIN                                    // 主函数开始
    DEFSTR STR1                         // 字符串变量声明
    INT AA                              // 声明整数型变量
    STR1 = “STATEMENT”                  // 字符串变量赋值 “STATEMENT”
    IO = SLEN(“STATEMENT”)              // 全局变量 IO中保存字符串个数
    AA = SLEN(STR1)                     // 内部变量AA中保存字符串个数
EOP                                     // 主函数结束
```

Related terms

SLEFT, SMID, SPOS, SRIGHT

10.42 SMID (提取指定位置字符串)

Usage

从字符串指定位置开始提取输入的值长度字符串.

Syntax

<变量> = SMID(<字符串>, <起始位置>, <提取个数>)

Description

从指定位置开始提取用户输入的整数个数的文字. 字符串的位置从0开始递增.

1) <字符串> 为提取字符串的对象.
2) <起始位置> 指定字符串提取起始位置.
   字符串起始位置为0.
3) <提取个数> 输入从指定位置开始读取的 文字个数.

Return value

<变量>	Data type: <字符串> 从指定位置开始返回设置的文字个数的字符串.

Arguments

<字符串>	Data type: <字符串>
<起始位置>	Data type: <整数型>
<提取个数>	Data type: <整数型>

Examples

```
MAIN                                    // 主函数开始
    DEFSTR STR0,STR1,STR2               // 字符串变量声明
    INT AA, BB                          // 声明整数型变量
    AA = 2                              // 整数型 变量初始化(起始位置)
    BB = 5                              // 整数型 变量初始化(提取个数)
    STR0 = “STATEMENT”                  // 字符串变量赋值 “STATEMENT”
    STR1 = SMID(“STATEMENT”, 2, 3)      // 变量 STR1中保存 “ATE” 字符串
    STR2 = SMID(STR0,AA,BB)             // 变量 STR2中保存 “ATEME” 字符串
EOP                                     // 主函数结束
```

Related terms

SLEFT, SLEN, SPOS, SRIGHT

10.43 SPOS (返回字符串匹配起始位置)

Usage

返回字符串1中匹配字符串2的起始位置.

Syntax

<变量> = SPOS(<字符串 1>, <字符串 2>)

Description

返回字符串1中匹配字符串2的起始位置.

1) <字符串 1> 输入比较对象字符串.
2) <字符串 2> 输入对象字符串中检索的字符串.

Return value

<变量>	Data type: <整数型> 比较输入的字符串1和2，返回匹配起始位置. 没有一致的字符串时返回-1 值.

Arguments

<字符串 1>	Data type: <字符串>
<字符串 2>	Data type: <字符串>

Examples

```
MAIN                                    // 主函数开始
    DEFSTR STR1,STR2                    // 字符串变量声明
    INT AA                              // 声明整数型变量
    STR1 = “TEST LINE”                  // 字符串变量初始化(整体字符串)
    STR2 = “LINE”                       // 字符串变量初始化(检索字符串)
    IO = SPOS(“TEST LINE”,“LINE”)       // 匹配字符串位置值5保存到全局变量
    AA = SPOS(STR1,STR2)                // 匹配字符串位置值5保存到局部变量
EOP                                     // 主函数结束
```

Related terms

SLEFT, SLEN, SMID, SRIGHT

10.44 SRIGHT (提取右侧字符串)

Usage

提取输入的字符串的右侧字符串.

Syntax

<变量> = SRIGHT(<字符串>, <提取个数>)

Description

从输入的字符串右侧开始提取输入的整数数量.

1) <字符串>为提取字符串的对象.
2) <提取个数>为从字符串右侧提取字符串的个数.

Return value

<变量>	Data type: <字符串> 返回提取选择的字符串个数的字符串.

Arguments

<字符串>	Data type: <字符串>
<提取个数>	Data type: <整数型>

Examples

Example 1

```
MAIN                                    // 主函数开始
    DEFSTR STR0,STR1,STR2               // 字符串变量声明
    INT AA                              // 声明整数型变量
    STR0 = “TEST”                       // 字符串变量赋值 “TEST”
    AA = 2                              // 整数型 变量初始化
    STR1 = SRIGHT(“TEST”, 1)            // STR1 字符串变量中保存 “T”
    STR2 = SRIGHT(STR0, AA)             // STR2 字符串变量中保存 “ST”
EOP                                     // 主函数结束
```

Example 2

```
MAIN                                    // 主函数开始
    DEFSTR STR0,STR1,STR2               // 字符串变量声明
    REAL AA                             // 声明实数型变量
    STR0 = “TEST”                       // 字符串变量赋值 “TEST”
    AA = 4.0                            // 实数型变量初始化
    STR1 = SRIGHT(“TEST”, 3.0)          // STR1 字符串变量中保存 “EST”
    STR2 = SRIGHT(STR0, AA)             // STR2 字符串变量中保存 “TEST”
EOP                                     // 主函数结束
```

Related terms

SLEFT, SLEN, SMID, SPOS

10.45 STRIN（接收通信字符串）

Usage

读取通过串口及Ethernet通信输入的字符串.

Syntax

<变量> = STRIN(<端口编号>, <等待时间>)

Description

输入的等待时间 (单位: ms)期间等待通信输入. 设置的等待时间期间未接收数据，则系统区域检查等待状态并执行下一步命令.

通过STRIN 函数输入的数据为判断字符串末端，使用CR+LF(默认设置) 或CR, LF 文字，称为换行符. 换行符必须出现在字符串的末端, 如果没有换行符则无法判断输入结束.

串口及Ethernet通信设置可以在Public参数通信设置区域设置.

1) <端口编号>指定接收字符串的物理端口.
   10：端口号38201
   11：端口号38202
   12：端口号38203
2) <等待时间>设置收信等待时间.

Return value

<变量>	Data type: <字符串> 返回收信等待时间期间接收的字符串.

Arguments

<端口编号>	Data type: <整数型> 仅可以输入RS-232(0) 或 Ethernet(10, 11, 12)
<等待时间>	Data type: <整数型>

Examples

```
MAIN                                    // 主函数开始
    DEFSTR RECVPK                       // 字符串变量声明
    INT LEN                             // 声明整数型变量
    RECVPK = STRIN(0,1000)              // RS-232 端口字符串收信等待1秒
    LEN = SLEN(RECVPK)                  // 保存接收的字符串长度
EOP                                     // 主函数结束
```

Related terms

STROUT

1) 为了从外部通过Ethernet与控制器收发字符串，除了IP 地址值以外还需要TCP 端口编号. 这种端口编号与使用STRIN, STROUT 命令时输入的<端口编号>不同，具有固定的值38201,38202,38203.

10.46 STROUT（发送通信字符串）

Usage

通过串口及Ethernet通信发送字符串.

Syntax

<变量> = STROUT(<端口编号>, <字符串>)

Description

通过串口通信输出输入的字符串，返回未发送的文字数. 例如，传送“ABCDEFGH”的文字时，成功返回0，没有全部发送则返回8. 此外，传送失败时返回赋值.

通过STROUT 函数输出的数据为判断字符串末端，使用CR+LF(默认设置) 或CR, LF 文字，称为换行符.

串口及Ethernet通信设置可以在Public参数通信设置区域设置.

1) <端口编号>指定发送字符串的物理端口.
   10：端口号38201
   11：端口号38202
   12：端口号38203
2) <字符串>输入发送的字符串.

Return value

<变量>	Data type: <整数型> 正常完车鞯传送时: 返回0 传送未完成时: 返回未传送的文字个数 发生传送错误时: 返回负数

Arguments

<端口编号>	Data type: <整数型> 仅可以输入RS-232(0) 或Ethernet(10, 11, 12).
<字符串>	Data type: <字符串>

Examples

```
MAIN                              // 主函数开始
    DEFSTR SENDPK                 // 字符串变量声明
    INT LEN                       // 声明整数型变量
    SENDPK = “SUCCESS”            // 传送字符串初始化
    IO = STROUT(10, “TEST”)       // 通过Ethernet端口传送“TEST”字符串
    LEN = STROUT(10,SENDPK)       // 通过Ethernet端口传送“SUCCESS”字符串
EOP                               // 主函数结束
```

Related terms

STRIN

1) 为了从外部通过Ethernet与控制器收发字符串，除了IP 地址值以外还需要TCP 端口编号. 这种端口编号与使用STRIN, STROUT 命令时输入的<端口编号>不同，具有固定的值38201,38202,38203.

10.47 SVAL (字符串转换为数字)

Usage

字符串转换为数字.

Syntax

<变量> = SVAL(<字符串>)

Description

将输入的字符串转换为整数或实数.

1) <字符串> 输入转换为数字字符串.

Return value

<变量>	Data type: <整数型>

Arguments

<字符串>	Data type: <字符串>

Examples

```
MAIN                          // 主函数开始
    DEFSTR STR1,STR2          // 字符串变量声明
    INT IVAL1                 // 声明整数型变量
    REAL FVAL                 // 声明实数型变量
    STR1 = “1234”             // 赋值转换为整数的字符串 “1234”
    STR2 = “12.34”            // 赋值转换为实数的字符串 “12.34”
    I0 = SVAL(“1234”)         // 整数型全局变量 I0中保存整数型常数1234
    IVAL1 = SVAL(STR1)        // 整数型内部变量IVAL中保存整数型常数1234
    F0 = SVAL(“12.34”)        // 实数型全局变量 F0中保存实数型常数12.34
    FVAL = SVAL(STR1)         // 实数型内部变量FVAL中保存实数型常数12.34
EOP                           // 主函数结束
```

Related terms

BINS, FTOS, HTOS

11. 系统

11.1 SETERR (发生SYSTEM EMG Alarm)

Usage

产生用户自定义的System emergency Alarm.

Syntax

SETERR(<用户Alarm编号>[, <用户Alarm信息>])

Description

JOB 程序执行过程中由用户发出System emergency的命令, Alarm信息窗口输出用户Alarm编号和信息.
此命令主要用户System模式的主线程.

1) <用户Alarm编号>在不超出整数范围的情况下用户自定义.
2) <用户Alarm信息>输出功能为方便识别用户自定义Alarm而提供.

Arguments

<用户Alarm编号>	Data type: <整数型> 用户自定义的Alarm编号
<用户Alarm信息>	Data type: <字符串> 用户自定义的Alarm输出信息

Examples

```
Example 1
IF (IN0 == 1) THEN                  // 0号输入信号为1时
    SETERR(1)                       // 通过用户Alarm编号1发出 System emergency
ENDIF                               // 条件语句结束

Example 2
IF (IN0 == 1) THEN                  // 0号输入信号为1时
    SETERR(1, “USER ERR 1”)         // 与显示信息发出System emergency
ENDIF                               // 条件语句结束
```

Related terms

RESET

11.2 RESET (解除Alarm)

Usage

解除所有发生的Alarm.

Syntax

RESET[(<Robot编号>)]

Description

清除当前发生的所有Alarm.
此命令主要用于System模式的主线程..

1) 输入<Robot编号>选择解除Alarm的Robot. 未输入时清除所有Robot的Alarm.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.

Examples

```
WHILE 1                                 // 开始重复语句
    IF(IN0 == 1) THEN                   // 0号输入信号为1时
        RESET                           // 清除所有Alarm
    ELSEIF (IN1 == 1) THEN              // 1号输入信号为1时
        RESET(1)                        // 解除Robot 1的Alarm
    ELSEIF (IN2 == 1) THEN              // 2号输入信号为1时
        RESET(2)                        // 解除Robot 2的Alarm
    ELSEIF (IN3 == 1) THEN              // 3号输入信号为1时
        RESET(3)                        // 解除Robot 3的Alarm
    ENDIF                               // 条件语句结束
ENDWL                                   // 重复语句结束
```

Related terms

SETERR

11.3 RERROR (返回最后Alarm编码)

Usage

返回最后 (最近)发生的Alarm编码.

Syntax

<变量> = RERROR(<Robot编号>)

Description

返回发生的Alarm中最后 (最近)发生的Alarm编码.
此命令主要用于System模式的主线程..

1) 输入<Robot编号>选择Robot. 输入0 时返回所有Robot的Alarm中最后 (最近)发生的Alarm编码.

Return value

<变量>	Data type: <整数型>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 0 ~ 3. (0: All, 1 ~ 3: 选择Robot)

Examples

```
INT A_ST, ECODE                         // 声明整数型变量

WHILE 1                                 // 执行重复语句
    A_ST = RSTATE(1,1)                  // 确认Alarm状态
    IF( A_ST == 1) THEN                 // 发生Alarm时
        ECODE = RERROR(0)               // 返回最后Alarm Code
        PRINT(0,ECODE)                  // 监控日志0号中保存最后Alarm编码
    ENDIF                               // 条件语句结束
ENDWL                                   // 重复语句结束
```

Related terms

RERRCNT, RERRCODE, RERRTEXT, READ_DERR

11.4 RERRCNT (返回发生的Alarm个数)

Usage

返回Alarm发生的总个数.

Syntax

<变量> = RERRCNT(<Robot编号>)

Description

返回发生的Alarm总个数.
此命令主要用于System模式的主线程..

1) 输入<Robot编号>选择Robot. 输入 0 时计算整体Robot发生的所有Alarm并返回其个数.

Return value

<变量>	Data type: <整数型>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 0 ~ 3. (0: All, 1 ~ 3: 选择Robot)

Examples

```
INT A_ST, ECNT                          // 声明整数型变量

WHILE 1                                 // 执行重复语句
    A_ST = RSTATE(1,1)                  // 确认Alarm状态
    IF( A_ST == 1) THEN                 // 发生Alarm的状态时
        ECNT = RERRCNT(0)               // 返回整体Robot发生的Alarm总数
        PRINT(0,ECNT)                   // 监控日志0号中保存Alarm总数
    ENDIF                               // 条件语句结束
ENDWL                                   // 重复语句结束
```

Related terms

RERROR, RERRCODE, RERRTEXT, READ_DERR

11.5 RERRCODE (返回索引编号的Alarm编码)

Usage

返回输入的索引编号对应的Alarm Code.

Syntax

<变量> = RERRCODE(<Robot编号>, <索引编号>)

Description

发生的所有Alarm中返回输入索引编号的Alarm Code.
此命令主要用于System模式的主线程..

1) 输入<Robot编号>选择Robot. 输入0 表示所有Robot.
2) <索引编号>表示Alarm发生顺序. 因此,所选Robot上发生的Alarm中返回符合顺序的Alarm Code.

Return value

<变量>	Data type: <整数型>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 0 ~ 3. (0: All, 1 ~ 3: 选择Robot)
<索引编号>	Data type: <整数型> 索引编号不可以超出Alarm的总个数.

Examples

```
INT A_ST, AA                              // 声明整数型变量
INT ECNT, ECODE
WHILE 1                                   // 执行重复语句
    A_ST = RSTATE(1,1)                    // 确认Alarm状态
    IF( A_ST == 1) THEN                   // 发生Alarm的状态时
        ECNT = RERRCNT(0)                 // 返回整体Robot中发生的Alarm总数
        PRINT(0,ECNT)                     // 监控日志0号中保存Alarm总数

        FOR AA = 1 TO ECNT                // 确认输入Alarm个数的Alarm Code
            ECODE = RERRCODE(0,AA)        // 返回指定索引的Alarm Code值
            PRINT(1+AA, ECODE)            // 监控日志中保存Alarm Code
        NEXT                              // 重复语句结束
    ENDIF                                 // 条件语句结束
ENDWL                                     // 重复语句结束
```

Related terms

RERROR, RERRCNT, RERRTEXT, READ_DERR

11.6 RERRTEXT (返回Alarm内容)

Usage

返回输入的索引编号对应的Alarm内容.

Syntax

<变量> = RERRTEXT(<Robot编号>, <索引编号>)

Description

发生的多个Alarm中返回输入的索引编号的Alarm内容.
此命令主要用于System模式的主线程..

1) 输入<Robot编号>选择Robot. 输入0 表示所有Robot.
2) <索引编号>表示Alarm发生顺序. 因此, 返回选择的Robot发生的Alarm中符合顺序的Alarm Code内容.

Return value

<变量>	Data type: <字符串>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 0 ~ 3. (0: All, 1 ~ 3: 选择Robot)
<索引编号>	Data type: <整数型> 索引编号不可以超过Alarm总数.

Examples

```
INT A_ST, AA                                // 声明整数型变量
INT ECNT, ECODE
DEFSTR E_STR                                // 字符串变量声明
WHILE 1                                     // 执行重复语句
    A_ST = RSTATE(1,1)                      // 确认Alarm状态
    IF( A_ST == 1) THEN                     // 发生Alarm的状态时
        ECNT = RERRCNT(0)                   // 返回发生的Alarm总数
        PRINT(0,ECNT)                       // 监控日志0号中保存Alarm总数
        FOR AA = 1 TO ECNT                  // 确认Alarm个数的Alarm Code
            ECODE = RERRCODE(0,AA)          // 返回指定索引的Alarm Code值
            E_STR = RERRTEXT(0,AA)          // 返回指定索引的Alarm内容
            PRINT(1+AA, ECODE, E_STR)       // 监控日志中保存Alarm Code和内容
        NEXT                                // 重复语句结束
    ENDIF                                   // 条件语句结束
ENDWL                                       // 重复语句结束
```

Related terms

RERROR, RERRCNT, RERRCODE, READ_DERR

11.7 READ_DERR (返回驱动Alarm Code)

Usage

返回指定轴发生的驱动Alarm Code.

Syntax

<变量> = READ_DERR(<Robot编号>, <轴编号>)

Description

返回指定轴编号指向的驱动Alarm Code.
此命令主要用于System模式的主线程..

1) 输入<Robot编号>选择Robot.
2) <轴编号>表示Robot关节连接顺序.

Return value

<变量>	Data type: <字符串>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> 轴编号不可以超过轴最大个数或小于最小个数.

Examples

```
INT ECODE, AA                          // 声明整数型变量

FOR AA = 1 TO 6                        // 1 ~ 6 轴为止执行重复语句
    ECODE = READ_DERR(1,AA)            // 返回Robot 1的各轴驱动Alarm
    PRINT(AA, ECODE)                   // 保存到监控缓冲区
NEXT                                   // 重复语句结束
```

Related terms

RERROR, RERRCNT, RERRCODE, RERRTEXT

11.8 RSTATE (获取Robot状态信息)

Usage

返回Robot状态的命令.

Syntax

<变量> = RSTATE(<Robot编号>, <状态索引>)

Description

获取指定Robot指定的索引信息.

1) <Robot编号>表示读取状态的Robot编号.
2) <状态索引>为预先定义的Robot特定状态信息.

Return value

<变量> Data type: <整数型>

Arguments

<Robot编号> Data type: <整数型>
<Robot编号>的范围为 1 ~ 3.

<状态索引> Data type: <整数型>
<状态索引>的范围为 0 ~ 16.

※ 下表中未说明的索引编号目前未使用.

<table>
<tr><th>INDEX</th><th>STATE NAME</th><th>内容</th></tr>
<tr><td>0</td><td>ALL</td><td>返回所有状态信息</td></tr>
<tr><td>1</td><td>ALARM</td><td>返回Alarm状态</td></tr>
<tr><td>2</td><td>MC STATUS (READY)</td><td>返回MC 状态 (0: OFF, 1: ON)</td></tr>
<tr><td>3</td><td>ORIGIN</td><td>返回原点完成状态
(编码器类型为ABS 类型时始终ON)</td></tr>
<tr><td>4</td><td>INPOSITION</td><td>完成目标位置到达</td></tr>
<tr><td>5</td><td>Reserved</td><td>预约</td></tr>
<tr><td>6</td><td>SERVO STATE</td><td>返回SERVO MOTOR ON(1)/OFF(0) 状态</td></tr>
<tr><td>7</td><td>Reserved</td><td>预约</td></tr>
<tr><td>8</td><td>MODE MISMATCH</td><td>Return T/P mode switch information and status of upper equipment (INDEX) mode
<table>
<tr><th>值</th><th>T/P MODE</th><th>INDEX MODE</th></tr>
<tr><td rowspan="2">0</td><td>SYSTEM</td><td>SYSTEM</td></tr>
<tr><td>ROBOT</td><td>ROBOT</td></tr>
<tr><td rowspan="2">1</td><td>SYSTEM</td><td>ROBOT</td></tr>
<tr><td>ROBOT</td><td>SYSTEM</td></tr>
</table>
※ Supported from IF F / W V1.2B or higher.</td></tr>
<tr><td>9</td><td>SYSTEM MODE</td><td>返回System(1)/Robot(0) 状态</td></tr>
<tr><td>10</td><td>Reserved</td><td>预约</td></tr>
</table>

11	EMG	发生紧急停止状况时ON(1)
12 ~ 14	Reserved	预约
15	ROBOT MOVING	通知相应Robot的Motion执行状态 (0: 未移动, 1: 移动中)
16	FAN STATE	通知Fan Alarm状态 (0: 无Alarm, 1:发生Alarm)

Examples

Example 1

- 确认所有状态，通过I/O单独输出

```
INT TEMP                          // 声明整数型变量
TEMP = RSTATE(1,0)                // 读取Robot 1号整体状态信息
OUT10 = (TEMP>>6)&OH01            // 输出伺服状态
OUT9 = (TEMP>>9)& OH01            // 输出系统状态
OUT8 = TEMP&OH01                  // 输出Alarm状态
```

Example 2

- 确认各别状态通过I/O输出

```
OUT10 = RSTATE(2,6)               // 输出Robot2号伺服状态
OUT9 = RSTATE(2,9)                // 输出Robot2号系统状态
OUT8 = RSTATE(2,1)                // 输出Robot2号Alarm状态
```

11.9 SWLIMIT (读取S/W LIMIT 参数值)

Usage

读取限制Robot各轴移动范围的参数值.

Syntax

<变量> = SWLIMIT(<Robot编号>, <轴编号>, <最大/最小>)

Description

Robot参数中读取BODY-RANG 设置的 S/W Limit 值.

1) 输入<Robot编号>选择读取参数值的Robot太.
2) 输入<轴编号>选择多个轴中读取值的轴编号.
3) <最大/最小>选择移动范围最小值或最大值.

Return value

<变量>	Data type: <实数型>

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ DOF.
<最大/最小>	Data type: <整数型> 读取最小值时选择0，读取最大值时选择1.

Examples

```
REAL LIMIT_MAX, LIMIT_MIN                // 声明实数型变量
LIMIT_MIN = SWLIMIT(1,1,0)               // 读取1号Robot1号轴移动范围最小值
LIMIT_MAX = SWLIMIT(1,1,1)               // 读取1号Robot1号轴移动范围最大值
```

11.10 READ_RPM (读取RPM 值)

Usage

读取Robot各轴当前RPM 值.

Syntax

<变量> = READ_RPM(<Robot编号>, <轴编号>)

Description

读取与输入的Robot 轴编号对应的当前RPM 值.

1) 输入<Robot编号>选择Robot.
2) 输入<轴编号>选择读取当前RPM的轴.

Return value

<变量>	Data type: <实数型> 结果值单位为RPM.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // 声明整数型变量
REAL RPM[6]                            // 声明实数型变量
DEFSTR STR                             // 字符串变量声明
WHILE 1                                // WHILE语句开始
    STR = “RPM:”                       // 字符串变量初始化
    FOR AA = 0 TO 5                    // 开始0到 5的 FOR语句
        RPM[AA] = READ_RPM(1,AA+1)     // 读取各轴RPM保存到数组
        STR = STR + FTOS(RPM[AA])      // 保存到字符串变量
    NEXT                               // FOR 语句结束
    R_VAL = STROUT(10, STR)            // 通过Ethernet端口传送字符串数
                                       //据

    DLAY 100                           // 100ms 等待
ENDWL                                  // WHILE 语句结束
```

Related terms

READ_LOAD, READ_TRQMAX, READ_VEL, READ_TRQ, READ_MTEMP, READ_CTEMP

11.11 READ_LOAD (读取负荷率)

Usage

读取Robot各轴当前负荷率.

Syntax

<变量> = READ_LOAD(<Robot编号>, <轴编号>)

Description

读取输入的Robot轴编号对应的当前负荷值.

1) 输入<Robot编号>选择Robot..
2) 输入<轴编号>选择读取当前负荷率的轴.

Return value

<变量>	Data type: <实数型> 结果值单位为%.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ DOF.

Examples

```
INT AA, R_VAL                              // 声明整数型变量
REAL LOAD[6]                               // 声明实数型变量
DEFSTR STR                                 // 字符串变量声明
WHILE 1                                    // WHILE 语句开始
    STR = “LOAD:”                          // 字符串变量初始化
    FOR AA = 0 TO 5                        // 开始0到 5的 FOR语句
        LOAD [AA] = READ_LOAD(1,AA+1)      // 读取各轴负载值保存到数组
        STR = STR + FTOS(LOAD [AA])        // 保存到字符串变量
    NEXT                                   // FOR 语句结束
    R_VAL = STROUT(10, STR)                // 通过Ethernet端口传送字符串数
                                           //据
    DLAY 100                               // 100ms 等待
ENDWL                                      // WHILE 语句结束
```

Related terms

READ_RPM, READ_TRQMAX, READ_VEL, READ_TRQ, READ_MTEMP, READ_CTEMP

11.12 READ_TRQMAX (读取最大扭矩值)

Usage

读取Robot的各轴最大扭矩值.

Syntax

<变量> = READ_TRQMAX(<Robot编号>, <轴编号>)

Description

读取输入的Robot和轴编号对应的最大扭矩值.

1) 输入<Robot编号>选择Robot..
2) 输入<轴编号>选择读取最大扭矩值的轴.

Return value

<变量>	Data type: <实数型> 结果值单位为 %.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ DOF.

Examples

```
INT AA, R_VAL                                   // 声明整数型变量
REAL T_MAX[6]                                   // 声明实数型变量
DEFSTR STR                                      // 字符串变量声明
WHILE 1                                         // WHILE语句开始
    STR = “TRQ_MAX:”                            // 字符串变量初始化
    FOR AA = 0 TO 5                             // 开始0到 5的 FOR语句
        T_MAX[AA] = READ_TRQMAX(1,AA+1)         // 读取各轴最大扭矩值保存到数组
        STR = STR + FTOS(T_MAX [AA])            // 保存到字符串变量
    NEXT                                        // FOR 语句结束
    R_VAL = STROUT(10, STR)                     // 通过Ethernet端口传送字符
                                                //串数据
    DLAY 100                                    // 100ms 等待
ENDWL                                           // WHILE 语句结束
```

Related terms

READ_RPM, READ_LOAD, READ_VEL, READ_TRQ, READ_MTEMP, READ_CTEMP

11.13 READ_VEL (读取速度)

Usage

读取Robot各轴当前速度值.

Syntax

<变量> = READ_VEL(<Robot编号>, <轴编号>)

Description

读取输入Robht和轴编号对应的当前速度值.

1) 输入<Robot编号>选择Robot..
2) 输入<轴编号>选择读取当前速度值的轴.

Return value

<变量>	Data type: <实数型> 旋转轴时的结果值单位为 degree/s, 直线移动轴时的单位为mm/s.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // 声明整数型变量
REAL V_VAL[6]                          // 声明实数型变量
DEFSTR STR                             // 字符串变量声明
WHILE 1                                // WHILE 语句开始
    STR = “VEL:”                       // 字符串变量初始化
    FOR AA = 0 TO 5                    // 开始0到 5的 FOR语句
        V_VAL[AA] = READ_VEL(1,AA+1)   // 读取各轴速度值保存到数组
        STR = STR + FTOS(V_VAL[AA])    // 保存到字符串变量
    NEXT                               // FOR 语句结束
    R_VAL = STROUT(10, STR)            // 通过Ethernet端口传送字符串数据
    DLAY 100                           // 100ms 等待
ENDWL                                  // WHILE 语句结束
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_TRQ, READ_MTEMP, READ_CTEMP

11.14 READ_TRQ (读取扭矩值)

Usage

读取Robot各轴当前扭矩值.

Syntax

<变量> = READ_TRQ(<Robot编号>, <轴编号>)

Description

读取输入的Robot和轴编号对应的当前扭矩值.

1) 输入<Robot编号>选择Robot..
2) 输入<轴编号>选择读取当前扭矩值的轴.

Return value

<变量>	Data type: <实数型> 结果值单位为 %.

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ DOF.

Examples

```
INT AA, R_VAL                                   // 声明整数型变量
REAL TRQ_VAL[6]                                 // 声明实数型变量
DEFSTR STR                                      // 字符串变量声明
WHILE 1                                         // WHILE 语句开始
    STR = "TRQ:"                                // 字符串变量初始化
    FOR AA = 0 TO 5                             // 开始0到5的FOR语句
        TRQ_VAL[AA] = READ_TRQ(1,AA+1)          // 读取各轴当前扭矩值保存到数组
        STR = STR + FTOS(TRQ_VAL[AA])           // 保存到字符串变量
    NEXT                                        // FOR 语句结束
    R_VAL = STROUT(10, STR)                     // 通过Ethernet端口传送字符串
                                                //数据
    DLAY 100                                    // 100ms 等待
ENDWL                                           // WHILE 语句结束
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_VEL, READ_MTEMP, READ_CTEMP

11.15 READ_MTEMP (读取电机编码器温度值)

Usage

读取Robot各轴当前电机编码器温度值.

Syntax

<变量> = READ_MTEMP(<Robot编号>, <轴编号>)

Description

读取输入的Robot和轴编号对应的当前电机编码器温度值.

1) 输入<Robot编号>选择Robot..
2) 输入<轴编号>选择读取当前电机编码器温度值的轴.

Return value

<变量>	Data type: <实数型> 结果值的单位为摄氏度(℃).

Arguments

<Robot编号>	Data type: <整数型> <Robot编号>的范围为 1 ~ 3.
<轴编号>	Data type: <整数型> <轴编号>的范围为 1 ~ DOF.

Examples

```
INT AA, R_VAL                              // 声明整数型变量
REAL TEMP[6]                               // 声明实数型变量
DEFSTR STR                                 // 字符串变量声明
WHILE 1                                    // WHILE 语句开始
    STR = " ENCODER TEMP:"                 // 字符串变量初始化
    FOR AA = 0 TO 5                        // 开始0到5的FOR语句
        TEMP[AA] = READ_MTEMP(1,AA+1)      // 读取各轴当前电机编码器温
                                           //度值保存到数组
        STR = STR + FTOS(TEMP[AA])         // 保存到字符串变量
    NEXT                                   // FOR 语句结束
    R_VAL = STROUT(10, STR)                // 通过Ethernet端口传送字符
                                           //串数据
    DLAY 100                               // 100ms 等待
ENDWL                                      // WHILE 语句结束
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_VEL, READ_TRQ, READ_CTEMP

11.16 READ_CTEMP (读取控制器内部温度值)

Usage

读取控制器内部温度值.

Syntax

<变量> = READ_CTEMP()

Description

读取当前控制器内部温度值.

Return value

<变量>	Data type: <实数型> 结果值的单位为摄氏度 (℃).

Examples

```
INT R_VAL                              // 声明整数型变量
REAL C_TEMP                            // 声明实数型变量
DEFSTR STR                             // 字符串变量声明
WHILE 1                                // WHILE 语句开始
    STR = " CONTROLLER TEMP:"          // 字符串变量初始化
    C_TEMP = READ_CTEMP ()             // 读取当前控制器温度值保存到数组
    STR = FTOS(C_TEMP)                 // 保存到字符串变量
    R_VAL = STROUT(10, STR)            // 通过Ethernet端口传送字符串数据
    DLAY 100                           // 100ms 等待
ENDWL                                  // WHILE 语句结束
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_VEL, READ_TRQ, READ_MTEMP

11.17 PRINT (监控日志)

Usage

记录监控用日志.

Syntax

PRINT(<索引>, <日志内容1> [{, <日志内容2>, … <日志内容N>}])

Description

执行JOB 程序的过程中记录可确认的日志.

记录日志时可以在日志内容代入所有变量及常数. 但, 目前不允许<日志内容>中代入函数保存其返回值的使用方式.

- 可使用例子: PRINT(0, 12, "STRING DATA", F[12], IO, AA) ← (○)
- 不可使用例子: PRINT(0,FTOS(R_VAL)) ← (×)

使用PRINT 留下的记录仅可以通过Pendant RUN 执行画面的 DISP 窗口 9.PRINT确认. 移动到用户指定的索引编号确认数据, 输入的日志内容分类由 "," 符号进行. 对于位置型数据使用 "<", ">" 符号分类, 符号内输出各轴位置值.

- DISP 画面9.PRINT 输出例子

```
000:[0:0]<-11.860,8.636,18.761,57.015,12.979,-100.037>
001:[0:0]1,DATA,12.4000000,45
002:
003:
```

使用PRINT 函数时在用户输入的日志内容前追加显示执行JOB程序的模式及Thread 编号.

- 画面输出信息例子: [<执行模式>:<Thread 编号>] <日志内容1>, …, <日志内容N>

执行模式和 Thread 编号的输出含义请参考下表.

分类	说明
<执行模式>	System (1), Robot (0)
<Thread 编号>	0 ~ 3

1) 指定<索引>选择保存的位置.
2) <日志内容>中记录用户需要保存的数据.

Arguments

<索引> Data type: <整数型>
<索引>的范围为 0 ~ 999.

<日志内容> Data type: 所有变量及常数

Examples

Example 1

```
MAIN                                    // 主函数开始
    POS ACUR                            // 位置型变量声明
    ACUR = HERE                         // 变量中保存当前Robot位置
    PRINT(0,ACUR)                       // 监控日志0号索引中保存位置值
EOP                                     // 主函数结束
```

- T/P 输出例子

```
000:[0:0]<-11.860,8.636,18.761,57.015,12.979,-100.037>
```

Example 2

```
MAIN                                    // 主函数开始
    INT I_VAL                           // 声明整数型变量
    REAL R_VAL                          // 声明实数型变量

    I_VAL = 1                           // 整数型 变量初始化
    R_VAL = 12.4                        // 实数型变量初始化
    PRINT(1,I_VAL, “DATA”, R_VAL,45)    // 监控日志1号索引中保存多值
EOP                                     // 主函数结束
```

- T/P 输出例子

```
001:[0:0]1,DATA,12.4000000,45
```

Related terms

WLOG

11.18 WLOG (用户日志)

Usage

以文件的形式保存包含时间信息的任意用户日志.

Syntax

WLOG(<日志内容1>, [{, <日志内容2>, … <日志内容N>}])

Description

执行JOB 程序的过程中记录任意的日志. 以文件的形式保存包含时间信息的日志，可以使用Uni-host程序或USB备份功能下载.

※ 对于下载功能的详细说明及使用方法，请参照Uni-host说明书(N2-HM-C□□) 或操作及应用说明书(N2-OM-C□□).

记录日志时日志内容中可以代入所有变量及常数. 但, 目前不允许<日志内容>中代入函数保存其返回值的使用方式.

- 可使用例子: WLOG(12, “STRING DATA”, F[12], IO, AA) ← (○)
- 不可使用例子: WLOG(FTOS(R_VAL)) ← (×)

使用WLOG 函数留下的记录可以通过Robot 模式9.INFO-> 3.LOG -> 3.ULOG 画面确认. 但，使用小型T/P时只能确认最近100条记录. 输入的日志内容分类由 “,” 符号进行. 输入的日志内容分类由 “,” 符号进行. 对于位置型数据使用 “<”, “>” 符号分类, 符号内输出各轴位置值.

- ULOG 画面输出例子

```
00:[180901 15:00:06.085]
01:[0:0]WLOG DATA1,12.4000000,45
02:[180901 15:15:12.072]
03:[0:0]WLOG DATA2,9.548000,0
```

使用WLOG 函数时首先记录日志的时间信息. 其后下一行记录用户输入的日志内容，日志内容前追加JOB 程序执行模式及Thread 编号记录.

- 画面输出信息例子: [<记录时间信息>]
  [<执行模式>:<Thread 编号>] <日志内容1>, …, <日志内容N>

执行模式和Thread 编号的输出含义如下表.

分类	说明
<执行模式>	System (1), Robot (0)
<Thread 编号>	0 ~ 3

1) <日志内容>中记录用户需要保存的数据.

Arguments

<日志内容>	Data type: 所有变量及常数

Examples

Example 1

```
MAIN                                    // 主函数开始
    POS ACUR                            // 位置型变量声明
    ACUR = HERE                         // 当前Robot位置保存到变量
    WLOG(“CUR POS”, ACUR)               // 用户日志中保存位置值
EOP                                     // 主函数结束
```

- 记录例子

```
01:[180901 15:30:05.085]
02:[0:0]CUR POS,<-11.860,8.636,18.761,57.015,12.979,-100.037>
```

Example 2

```
MAIN                                    // 主函数开始
    INT I_VAL                           // 声明整数型变量
    REAL R_VAL                          // 声明实数型变量

    I_VAL = 100                         // 整数型 变量初始化
    R_VAL = 12.4                        // 实数型变量初始化
    WLOG(“WLOG DATA”, R_VAL, I_VAL)     // 用户日志中保存多值
EOP                                     // 主函数结束
```

- 记录日志

```
01:[180901 15:30:06.072]
02:[0:0]WLOG DATA,12.4000000,100
```

Related terms

PRINT

11.19 CLOG (通信用日志) ※ 目前不支持.

11.20 WGPNT (保存全局Point变量文件)

Usage

将全局Point变量保存为文件.

Syntax

[<变量> =] WGPNT(<同步>)

Description

JOB 程序执行中修改全局Point变量时, 使用WGPNT函数保存到磁盘.

1) <同步>中输入值选择是否等待到同步磁盘.

Return value

<变量>	Data type: <整数型> 结果值返回 0(写入正常) 或 -1(写入失败)值.

Arguments

<同步>	Data type: <整数型> 只能输入0(不等待磁盘同步) 或 1(同步磁盘同步).

Examples

```
MAIN                                        // 主函数开始
    POS ACUR                                // 位置型变量声明
    ACUR = <100,100,100,100,100,100>        // 位置型变量初始化
    GP0 = ACUR                              // GP0中代入位置型变量
    WGPNT(1)                                // 全局Point变量保存到文件
EOP                                         // 主函数结束
```

Related terms

WGINT, WGFLT

1) 程序执行过程中变更全局变量而未执行保存命令时, 无法保证电源切断或重启时变更数据仍存在.
2) 保存文件的过程中突然断电时有可能发生数据丢失.

11.21 WGINT (保存全局整数变量文件)

Usage

全局整数变量保存在文件中.

Syntax

[<变量> =] WGINT(<同步>)

Description

JOB 程序执行过程中修改全局整数变量时，为了保存到磁盘使用WGINT 函数.

1) <同步>中输入值选择是否等待磁盘同步.

Return value

<变量>	Data type: <整数型> 结果值返回0(写入正常) 或 -1(写入错误)值.

Arguments

<同步>	Data type: <整数型> 只可以输入0(不等待磁盘同步) 或 1(等待磁盘同步).

Examples

```
MAIN                                // 主函数开始
    INT I_VAL                       // 声明整数型变量
    I_VAL = 50 +25                  // 整数型 变量初始化
    I100 = 100                      // I100中代入值
    I101 = I_VAL                    // I101中代入值
    WGINT(1)                        // 全局整数变量保存到文件中
EOP                                 // 主函数结束
```

Related terms

WGPNT, WGFLT

1) 程序执行过程中变更全局变量而未执行保存命令时， 无法保证电源切断或重启时变更数据仍存在.
2) 保存文件的过程中突然断电时有可能发生数据丢失.

11.22 WGFLT (保存全局实数变量文件)

Usage

将全局Point变量保存为文件.

Syntax

[<变量> =] WGFLT(<同步>)

Description

程序执行过程中修改全局实数变量时，为了保存到磁盘使用WGFLT 函数.

1) <同步>中输入值选择是否等待磁盘同步.

Return value

<变量> Data type: <整数型>
结果值返回0(写入正常) 或 -1(写入错误)值.

Arguments

<同步> Data type: <整数型>
只可以输入0(不等待磁盘同步) 或 1(等待磁盘同步).

Examples

```
MAIN                                    // 主函数开始
    REAL R_VAL                          // 声明实数型变量
    R_VAL = RAD(90)                     // 实数型变量初始化
    F100 = 3.14                         // F100中代入值
    F101 = R_VAL                        // F101中代入值
    WGFLT(1)                            // 全局实数变量保存到文件中
EOP                                     // 主函数结束
```

Related terms

WGPNT, WGINT

1) 程序执行过程中变更全局变量而未执行保存命令时， 无法保证电源切断或重启时变更数据仍存在.
2) 保存文件的过程中突然断电时有可能发生数据丢失.

11.23 TACT_START（设置时间测量起始位置）

Usage

设置为测量工程时间的测量起始位置.

Syntax

TACT_START()

Description

JOB 程序执行过程中执行需要测量移动命令或工程整体的执行时间时， 为设置测量起始位置使用TACT_START 函数.

Examples

```
MAIN                                  // 主函数开始
     INT VAL                          // 声明整数型变量
     TACT_START()                     // 设置时间测量起始位置
     JMOV P0                          // JMOV 到P0
     LMOV P1                          // LMOV到P1
     VAL = TACT_STOP()                // 设置时间测量结束位置
     PRINT(0,VAL)                     // 监控日志0号保存索引测量时间
EOP                                   // 主函数结束
```

Related terms

TACT_START, TACT_STOP

1） 使用TACT_STOP 函数不设置时间测量结束位置时，无法使用时间测量功能.

11.24 TACT_STOP (设置时间测量结束位置)

Usage

设置工程时间测量的测量结束位置.

Syntax

[<变量> =] TACT_STOP()

Description

JOB程序执行过程中执行需要测量移动命令或工程整体的执行时间时， 为设置测量结束位置使用TACT_STOP 函数.

Return value

<变量>	Data type: <整数型> 结果值返回测量时间. 结果值的单位为ms.

Examples

```
MAIN                              // 主函数开始
    INT VAL                       // 声明整数型变量
    TACT_START()                  // 设置时间测量起始位置
    JMOV P0                       // JMOV到P0
    LMOV P1                       // LMOV到P1
    VAL = TACT_STOP()             // 设置时间测量结束位置
    PRINT(0,VAL)                  // 监控日志0号索引中保存测量时间
EOP                               // 主函数结束
```

Related terms

TACT_START, TACT_STOP

1) 使用TACT_START 函数不设置时间测量起始位置时，无法使用时间测量功能

12. System模式专用命令

12.1 PSEL (选择程序)

Usage

匹配指定 Robot Thread 编号中输入的 JOB 文件.

Syntax

[<变量> =] PSEL(<Thread 编号>, “<JOB 文件名>”)

Description

读取输入的JOB 文件确认文件中是否存在错误，向指定Robot Thread匹配JOB 文件.
此命令只能在控制器System模式使用，无法在Robot模式使用.

1) <Thread 编号>输入值选择要执行的JOB 程序.
2) <JOB 文件名>中输入除文件结束符号 “.” 和扩展名 “JOB”的文件名.

Return value

<变量> Data type: <整数型>
结果值返回0(选择成功) 或负值 (选择失败).

- 选择失败原因

值	原因
-1	非小型 T/P的System 模式或, Robot模式下使用PSEL 命令时
-2	输入数据错误
-3	输入的JOB文件不存在时
-4	输入的JOB 文件名为 “BGTD” 或“BGTF”时
-5	选择的Thread正在执行状态时
-6	JOB 文件加载失败时

Arguments

<Thread 编号> Data type: <整数型>
Thread 编号的范围为 1 ~ 4(TH1 ~ TH4).

Examples

```
INT R_VAL, STATE                          // 声明整数型变量

R_VAL = PSEL(1,"MOVE1")                   // Robot 1号 Thread选择
                                          //MOVE1.JOB 文件
IF (R_VAL<0) THEN                         // 确认负值返回值时
    GOTO END_PROGRAM                      // 程序选择失败, 移动到结束跳转点
ENDIF                                     // 条件语句结束

WHILE 1                                   // 执行重复语句
    STATE = PSTATE(1)                     // 读取程序执行状态
    IF ((STATE == 2)&&(IN0 == 1))THEN     // 状态值为2(TH_STOP), 且0
                                          //号输入为1时
        R_VAL = PSEL(1,"MOVE2")           // Robot 1号 Thread选择
                                          //MOVE2.JOB 文件
        IF (R_VAL<0) THEN                 // 确认负值返回值时
            GOTO END_PROGRAM              // 选择程序失败, 移动到结束跳转点
        ENDIF                             // 条件语句结束
    ELSEIF ((STATE == 2)&&(IN1 == 1))THEN // 状态值为2(TH_STOP), 且1
                                          //号输入为1时
        R_VAL = PSEL(1,"MOVE1")           // Robot 1号 Thread选择
                                          //MOVE1.JOB 文件
        IF (R_VAL<0) THEN                 // 确认负值返回值时
            GOTO END_PROGRAM              // 选择程序失败, 移动到结束跳转点
        ENDIF                             // 条件语句结束
    ELSEIF ((STATE == 2)&&(IN2 == 1)) THEN // 状态值为2(TH_STOP), 且2
                                          //号输入为1时
        PSTART(1)                         // 开始执行程序
    ELSEIF ((STATE == 3)&&(IN3 == 1)) THEN // 状态值为3(TH_RUN), 且3号
                                          //输入为1时
        PSTOP(1)                          // 停止程序
    ELSEIF ((STATE == 2)&&(IN4 == 1)) THEN // 状态值为2(TH_STOP), 且4
                                          //号输入为1时
        PRESTART(1)                       // 重新开始程序
    ELSEIF ((STATE == 2)&&(IN5 == 1)) THEN // 状态值为2(TH_STOP), 且5
                                          //号输入为1时
        PEXIT(1)                          // 程序解除
    ELSEIF STATE == 0 THEN                // 状态值为0(TH_ERROR)时
        GOTO END_PROGRAM                  // 移动到结束跳转点
    ENDIF                                 // 条件语句结束
ENDWL                                     // 重复语句结束
LABL END_PROGRAM                          // 程序结束跳转点声明
```

Related terms

PSTART, PSTOP, PSTATE, PRESTART, PEXIT

1) 此命令只可以在控制器System模式下使用, Robot模式下使用时发出 “Cannot use PSEL at foretask.” Alarm.
2) 当前 Thread 编号只能选择TH1(1).

12.2 PSTART (开始程序)

Usage

执行指定 Robot Thread 编号中选择的 JOB 程序.

Syntax

[<变量> =] PSTART(<Thread 编号>)

Description

从第一行开始执行指定Robot Thread 编号加载的程序.

1) <Thread 编号>中输入值执行JOB 程序.

Return value

<变量> Data type: <整数型>
结果值返回0(执行成功) 或负值 (执行失败).

- 执行失败原因

值	原因
-1	非小型 T/P的System 模式或, Robot模式下使用PSEL 命令时
-2	输入数据错误
-3	选择的Thread无法执行或正在执行状态时

Arguments

<Thread 编号> Data type: <整数型>
Thread 编号的范围为 1 ~ 4(TH1 ~ TH4).

Examples

```
INT R_VAL                              // 声明整数型变量
PSEL(1, “MOVE”)                        // Robot 1号 Thread选择MOVE.JOB 文件
R_VAL = PSTART(1)                      // 开始执行1号Thread匹配的程序
PRINT(0, “RETURN VALUE: ”,R_VAL)       // 监控日志0号中保存返回值
WHILE 1                                // 开始重复语句
ENDWL                                  // 重复语句结束
```

Related terms

PSEL, PSTOP, PSTATE, PRESTART, PEXIT

1) 此命令只可以在控制器System模式下使用, Robot模式下使用时发出 “Cannot use PSEL at foretask.” Alarm.
2) 当前 Thread 编号只能选择TH1(1).

12.3 PSTOP (停止程序执行)

Usage

停止执行指定 Robot Thread 编号中输入的 JOB 文件.

Syntax

[<变量> =] PSTOP(<Thread 编号>)

Description

停止执行Robot Thread 编号上执行的JOB程序.

1) <Thread 编号>中输入值，停止执行中的JOB程序.

Return value

<变量> Data type: <整数型>
结果值返回0(成功停止) 或负值 (停止失败).

- 停止失败原因

值	原因
-1	非小型 T/P的System 模式或, Robot模式下使用PSEL 命令时
-2	输入数据错误

Arguments

<Thread 编号> Data type: <整数型>
Thread 编号的范围为 1 ~ 4(TH1 ~ TH4).

Examples

```
PSEL(1, “MOVE”)                  // Robot 1号 Thread选择MOVE.JOB 文件
PSTART(1)                        // 开始执行Robot 1号Thread匹配的程序
WHILE 1                          // 开始重复语句
    IF INO == 1 THEN             // 0号输入值为1时
        PSTOP(1)                 // 停止Robot 1号 Thread匹配的程序
    ENDIF                        // 条件语句结束
ENDWL                            // 重复语句结束
```

Related terms

PSEL, PSTART, PSTATE, PRESTART, PEXIT

1) 此命令只可以在控制器System模式下使用, Robot模式下使用时发出 “Cannot use PSEL at foretask.” Alarm.
2) 当前 Thread 编号只能选择TH1(1).

12.4 PSTATE (确认程序状态)

Usage

读取指定 Robot Thread 编号的状态.

Syntax

<变量> = PSTATE(<Thread 编号>)

Description

读取指定Robot Thread 编号的当前状态.

1) <Thread 编号>中输入值读取状态.

Return value

<变量> Data type: <整数型>
返回Robot Thread的当前状态信息.

- 0 (TH_ERROR): Thread 编号错误
- 1 (TH_IDLE): 未分配Thread, 可以使用
- 2 (TH_STOP): Thread 停止
- 3 (TH_RUN): Thread 执行中

状态确认失败时返回负值.

- 失败原因

值	原因
-1	非小型 T/P的System 模式或, Robot模式下使用PSEL 命令时
-2	输入数据错误

Arguments

<Thread 编号> Data type: <整数型>
Thread 编号的范围为 1 ~ 4(TH1 ~ TH4).

Examples

```
INT STATE                              // 声明整数型变量
PSEL(1, “MOVE”)                        // Robot 1号 Thread选择MOVE.JOB 文件
PSTART(1)                              // 开始执行Robot 1号Thread匹配的程序
WHILE 1                                // 开始重复语句
    STATE = PSTATE(1)                  // 确认Robot 1号Thread的状态
    IF STATE < 0 THEN                  // 状态确认失败时
        PRINT(0,”ERROR:”, STATE)       // 监控日志0号中保存失败原因
        BREAK                          // 强制结束重复语句
    ENDIF                              // 条件语句结束
ENDWL                                  // 重复语句结束
```

Related terms

PSEL, PSTART, PSTOP, PRESTART, PEXIT

1) 此命令只可以在控制器System模式下使用, Robot模式下使用时发出 “Cannot use PSEL at foretask.” Alarm.
2) 当前 Thread 编号只能选择TH1(1).

12.5 PRESTART (重新开始程序)

Usage

重新执行指定 Robot Thread 编号中停止的 JOB 程序.

Syntax

[<变量> =] PRESTART(<Thread 编号>)

Description

加载到指定Robot Thread 编号的程序中从停止行开始执行.

1) <Thread 编号>输入值重新执行JOB 程序.

Return value

<变量> Data type: <整数型>
结果值返回0(执行成功) 或负值 (执行失败).

- 执行失败原因

值	原因
-1	非小型 T/P的System 模式或, Robot模式下使用PSEL 命令时
-2	输入数据错误

Arguments

<Thread 编号> Data type: <整数型>
Thread 编号的范围为 1 ~ 4(TH1 ~ TH4).

Examples

```
INT STATE                                   // 声明整数型变量
PSEL(1, "MOVE")                             // Robot 1号 Thread选择
                                            //MOVE.JOB 文件
PSTART(1)                                   // 开始执行Robot 1号Thread
                                            //匹配的程序
WHILE 1                                     // 开始重复语句
    STATE = PSTATE(1)                       // 确认Robot 1号Thread的状态
    IF ((STATE == 3)&&(IN0 == 1)) THEN      // 状态值为3(TH_RUN), 且0号
                                            //输入为 1时
        PSTOP(1)                            // 停止程序
    ELSEIF ((STATE == 2)&&(IN1 == 1)) THEN  // 状态值为2(TH_STOP), 且1
                                            //号输入为1时
        PRESTART(1)                         // 重新开始程序
    ELSEIF ((STATE == 2)&&(IN2 == 1)) THEN  // 状态值为2(TH_STOP), 且2
                                            //号输入为1时
        PEXIT (1)                           // 解除程序
    ENDIF                                   // 条件语句结束
ENDWL                                       // 重复语句结束
```

Related terms

PSEL, PSTART, PSTOP, PSTATE, PEXIT

1) 此命令只可以在控制器System模式下使用, Robot模式下使用时发出 “Cannot use PSEL at foretask.” Alarm.
2) 当前 Thread 编号只能选择TH1(1).

12.6 PEXIT(解除程序)

Usage

解除指定 Robot Thread 编号 JOB.

Syntax

[<变量> =] PEXIT(<Thread 编号>)

Description

解除指定Robot Thread 编号中加载的程序.

1) <Thread 编号>中输入值解除JOB程序.

Return value

<变量> Data type: <整数型>
结果值返回0(执行成功) 或负值 (执行失败).

- 执行失败原因

值	原因
-1	非小型 T/P的System 模式或, Robot模式下使用PSEL 命令时
-2	输入数据错误

Arguments

<Thread 编号> Data type: <整数型>
Thread 编号的范围为 1 ~ 4(TH1 ~ TH4).

Examples

```
INT STATE                                  // 声明整数型变量
PSEL(1, "MOVE")                            // Robot 1号 Thread选择MOVE.JOB 文件
PSTART(1)                                  // 开始执行Robot 1号Thread匹配的程序
WHILE 1                                    // 开始重复语句
     STATE = PSTATE(1)                     // 确认Robot 1号Thread的状态
     IF ((STATE == 2)&&(IN1 == 1)) THEN    // 状态值为2(TH_STOP), 且1号输入为1时
          PSTOP(1)                         // 程序结束
          PEXIT(1)                         // 解除程序
     ENDIF                                 // 条件语句结束
ENDWL                                      // 重复语句结束
```

Related terms

PSEL, PSTART, PSTOP, PSTATE, PEXIT

1) 此命令只可以在控制器System模式下使用, Robot模式下使用时发出 “Cannot use PSEL at foretask.” Alarm.
2) 当前 Thread 编号只能选择TH1(1).

13. 对外沟通

13.1 NET_OPEN (TCP I/P Client 套接字创建及接入请求)

Usage

创建TCP I/P Client 套接字，请求访问Server的指令.

Syntax

<整数型变量> = NET_OPEN(<IP >, <端口编号>, <等待时间>)

Description

创建TCP I/P Client 套接字，请求访问输入的IP, Port 编号的TCP I/P Server.
同时最多可连接3个Port，无法追加创建.
请求访问Server 时等待输入的等待时间 ， 正常连接时返回相应套接字的File descriptor，连接失败时返回-1.

1) <IP >中输入 Server的 IP .
2) <端口编号>中输入 TCP 端口编号.
3) <等待时间>输入 msec 单位的时间，访问服务器后等待输入的等待时间的应答.

Return value

<变量>	Data type: <整数型> 正常连接时返回相应套接字的File descriptor，连接失败时返回-1.

Arguments

<IP编号>	Data type: <字符串>
<端口编号>	Data type: <整数型> 端口编号的范围为1 ~ 65,535.
<等待时间>	Data type: <整数型> 等待时间的单位为msec, 范围为1 ~ 10,000.

Examples

```
MAIN                                    // 声明主函数
    INT FD_NUM                          // 返回值整数型变量声明
    DEFSTR IP                           // IP 字符串变量声明
    INT PORT                            // 端口编号 整数型变量声明
    INT WAIT_T                          // 等待时间 整数型变量声明

    IP = “192.168.1.196”                // IP 定义
    PORT = 10000                        // 端口编号定义
    WAIT_T = 1000                       // 等待时间定义 (1秒)

    FD_NUM = NET_OPEN(IP,PORT, WAIT_T)  // TCP I/P 访问请求
EOP                                     // 结束主函数
```

Related terms

NET_CLOSE, NET_STATUS, NET_KEEPALIVE, NET_DELIMITER, NET_FD

13.2 NET_CLOSE (TCP I/P 通信结束)

Usage

结束控制器的TCP I/P 通信，清除客户端套接字的指令.

Syntax

[<整数型变量> =] NET_CLOSE(<文件描述编号>)

Description

结束连接的控制器TCP I/P 通信.

1) <文件描述编号>中输入NET_FD 或 NET_OPEN 指令返回的文件描述符 (File descriptor)编号.

Return value

<变量>	Data type: <整数型> 结果值返回0(成功结束) 或 -1(结束失败)值.

Arguments

<文件描述编号>	Data type: <整数型> 文件描述编号的范围为-1 ~ 1,024.

Examples

```
MAIN                                        // 声明主函数
    INT FD_NUM                              // 文件描述编号 整数型变量声明
    DEFSTR IP                               // IP字符串变量声明
    INT PORT                                // 端口编号 整数型变量声明
    INT WAIT_T                              // 等待时间 整数型变量声明

    IP = “192.168.1.196”                    // IP 定义
    PORT = 10000                            // 端口编号定义
    WAIT_T = 1000                           // 等待时间定义 (1秒)

    FD_NUM = NET_OPEN(IP,PORT, WAIT_T)      // TCP I/P 访问请求
    NET_CLOSE(FD_NUM)                       // TCP I/P 结束请求
EOP                                         // 结束主函数
```

Related terms

NET_OPEN, NET_STATUS, NET_KEEPALIVE, NET_DELIMITER, NET_FD

1) 文件描述编号表示系统分配的套接字固有编号.
2) TCP I/P Server可以通过NET_FD 指令返回, Client 可以通过NET_OPEN 命令返回.

13.3 NET_STATUS (确认TCP I/P 通信状态)

Usage

确认控制器的TCP I/P 通信状态的指令.

Syntax

<整数型变量> = NET_STATUS(<文件描述编号>)

Description

确认并返回控制器的TCP I/P 通信状态.

1) <文件描述编号>中输入NET_FD 或 NET_OPEN 指令返回的文件描述符 (File descriptor)编号.

Return value

<变量>	Data type: <整数型> 结果值返回0(成功结束) 或 -1(错误或链接失败)值.

Arguments

<文件描述编号>	Data type: <整数型> 文件描述编号的范围为 -1 ~ 1,024.

Examples

```
MAIN                                        // 声明主函数
    INT FD_NUM                              // 文件描述编号整数型变量声明
    DEFSTR IP                               // IP字符串变量声明
    INT PORT                                // 端口编号 整数型变量声明
    INT WAIT_T                              // 等待时间 整数型变量声明

    IP = "192.168.1.196"                    // IP 定义
    PORT = 10000                            // 端口编号定义
    WAIT_T = 1000                           // 等待时间定义 (1秒)

    FD_NUM = NET_OPEN(IP,PORT, WAIT_T)      // TCP I/P 访问请求
    RET_VEL = NET_STATUS(FD_NUM)            // TCP I/P 状态信息请求
EOP                                         // 结束主函数
```

Related terms

NET_OPEN, NET_CLOSE, NET_KEEPALIVE, NET_DELIMITER, NET_FD

1) 文件描述编号表示系统分配的套接字固有编号.
2) TCP I/P Server可以通过NET_FD 指令返回, Client 可以通过NET_OPEN 命令返回.

13.4 NET_KEEPALIVE (TCP I/P Client 套接字通信状态检查用选项设置)

Usage

TCP I/P 通信过程中周期性检查通信状态，感知LAN 断线或非正常会话结束的指令.

Syntax

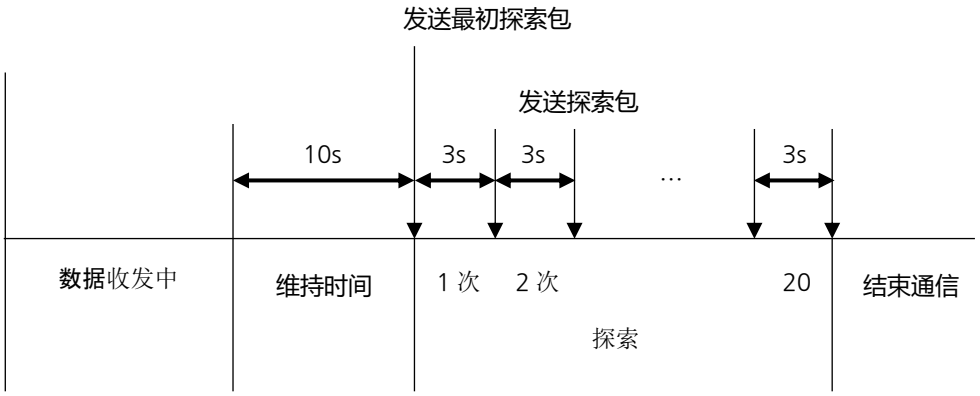
<整数型变量> = NET_KEEPALIVE(<文件描述编号>, <维持时间>, <探索最大次数>, <探索周期>)

Description

最后数据包到达后经过<维持时间>后传送探索包，等待<探索周期>. 无应答时每隔<探索周期> 发送探索包, 如果<探索最大次数> 期间无应答则结束通信.
默认值分别为10秒, 20次, 3秒，可以通过NET_KEEPALIVE 指令更改设定值.

1) <文件描述编号>中输入NET_FD 或 NET_OPEN 指令返回的文件描述符 (File descriptor)编号.
2) <维持时间>表示从发送最后数据包开始的时间到开始发送最初探索包的时间间隔.
3) <探索最大次数>表示如无探索包相应应答时重复发送探索包的最大次数.
4) <探索周期>表示发送探索包的间隔.

EX) 输入<维持时间: 10秒>, <探索最大次数: 20次>, <探索周期: 3秒> 时，最后数据包到达后10秒内没有发送数据则发送探索包，仍没有应答时以3秒为周期发送探索包. 传送20次探索包后，超出最大次数则结束通信.



Return value

<变量>	Data type: <整数型> 结果值返回0(执行成功) 或 -1(执行失败).

Arguments

<文件描述编号>	Data type: <整数型> 文件描述编号的范围为 -1 ~ 1,024.
<维持时间>	Data type: <整数型> 维持时间的单位为秒(sec), 范围为1 ~ 10,000.
<探索最大次数>	Data type: <整数型> 重复次数单位为整数, 范围为1 ~ 10,000.
<探索周期>	Data type: <整数型> 重复周期的单位为秒 (sec), 范围为1 ~ 10,000.

Examples

```
MAIN                                                    // 声明主函数
    INT RET_V                                           // 返回值整数型变量声明
    INT FD                                              // 文件描述编号整数变量声明
    DEFSTR IP                                           // IP字符串变量声明
    INT PORT, WAIT_T                                    // 整数型变量声明
    INT T_IDLE, T_CNT, T_INTV

    IP = "192.168.1.196"                                // IP 定义
    PORT = 10000                                        // 端口编号定义
    WAIT_T = 1000                                       // 等待时间定义 (1秒)

    FD = NET_OPEN(IP,PORT, WAIT_T)                      // TCP I/P 访问请求

    T_IDLE = 10                                         // 维持时间定义为 10秒
    T_CNT = 20                                          // 探索最大次数定义为20次
    T_INTV = 3                                          // 探索周期定义为3秒

    RET_V = NET_KEEPALIVE(FD,T_IDLE,T_CNT,T_INTV)       // 执行函数
EOP                                                     // 结束主函数
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_DELIMITER, NET_FD

1) 文件描述编号表示系统分配的套接字固有编号.
2) TCP I/P Server可以通过NET_FD 指令返回, Client 可以通过NET_OPEN 命令返回.

13.5 NET_DELIMITER (TCP I/P 结束字符串设定函数)

Usage

TCP I/P 通信的输入/输出结束字符串设定指令.

Syntax

<整数型变量> = NET_DELIMITER(<文件描述编号>, <结束字符串类型>)

Description

初始结束字符串设置为CR(0x0A) + LF(0x0D) ，可以更改设定值.

1) <文件描述编号>中输入NET_FD 或 NET_OPEN 指令返回的文件描述符 (File descriptor)编号.
2) <结束字符串类型>的范围为0 ~ 2，其含义如下.
   0: CR + LF
   1: LF
   2: CR

Return value

<变量>	Data type: <整数型> 结果值范围0(执行成功) 或 -1(执行失败).

Arguments

<文件描述编号>	Data type: <整数型> 文件描述编号的范围为 -1 ~ 1024.
<结束字符串类型>	Data type: <整数型> 结束字符串类型范围为0 ~ 2.

Examples

```
MAIN                                    // 声明主函数
    INT FD                              // 文件描述编号整数变量声明
    DEFSTR IP                           // IP字符串变量声明
    INT PORT, WAIT_T,                   // 整数型变量声明
    INT RET_V, DELI

    IP = "192.168.1.196"                // IP 定义
    PORT = 10000                        // 端口编号定义
    WAIT_T = 1000                       // 等待时间定义 (1秒)

    FD = NET_OPEN(IP,PORT, WAIT_T)      // TCP I/P 访问请求

    DELI = 2                            // 结束字符串设置为2(CR)

    RET_V = NET_DELIMITER(FD)           // 执行函数
EOP                                     // 结束主函数
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_KEEPALIVE, NET_FD

1) 文件描述编号表示系统分配的套接字固有编号.
2) TCP I/P Server可以通过NET_FD 指令返回, Client 可以通过NET_OPEN 命令返回.

13.6 NET_FD (返回TCP I/P Server FD)

Usage

返回访问控制器 TCP I/P Server的Client Socket文件描述符(File descriptor) 编号.

Syntax

<整数型变量> = NET_FD(<索引>)

Description

返回与<索引>对应的TCP I/P Server Port接入的 Client Socket的文件描述编号.
成功连接时返回相应Client Socket的文件描述编号, 未连接时返回-1.

1) <索引>的范围为10 ~ 12, 其含义如下.
   10: Port 号 38201
   11: Port 号 38202
   12: Port 号 38203

Arguments

<索引>	Data type: <常数, 整数型变量> <索引>的范围为10 ~ 12.

Examples

```
MAIN                                    // 声明主函数
    INT FD_NUM                          // 返回值整数型变量声明
    INT INDEX                           // Index 值整数型变量声明

    INDEX = 10                          // Index 值设置

    FD_NUM = NET_FD(FD_NUM)             // 执行函数
EOP                                     // 结束主函数
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_KEEPALIVE, NET_DELIMITER

13.7 NET_STRIN (接收通信字符串)

Usage

通过以太网通信读取输入的字符串.

Syntax

<字符串变量> = NET_STRIN(<文件描述编号>, <等待时间>)

Description

等待输入的等待时间(单位: ms)的通信输入. 设定的等待时间期间未接收数据，则检查系统区域的等待状态执行下一个步骤的指令.

通过NET_STRIN 函数输入的数据使用CR+LF(默认设置) 或 CR, LF 文字判断字符串的结束，称为换行符. 换行符必须出现在字符串的结尾，无换行符时无法判断输入的结尾.

1) <文件描述编号>中输入NET_FD 或 NET_OPEN 指令返回的文件描述符 (File descriptor)编号.
2) <等待时间>设置接收等待时间.

Return value

<变量>	Data type: <字符串> 返回接收等待时间期间接收的字符串.

Arguments

<文件描述编号>	Data type: <整数型> 文件描述编号的范围为 -1 ~ 1,024.
<等待时间>	Data type: <整数型>

Examples

```
MAIN                                    // 开始主函数
    DEFSTR RECVPK                       // 字符串变量声明
    INT FD_NUM                          // 整数型变量声明
    RECVPK = NET_STRIN(FD_NUM,1000)     // 字符串接收等待1秒
EOP                                     // 结束主函数
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_STROUT, NET_KEEPALIVE, NET_FD

13.8 NET_STROUT (发送通信字符串)

Usage

通过串口及以太网通信发送字符串.

Syntax

<变量> = NET_STROUT(<文件描述编号>, <字符串>)

Description

通过串口通信将输入的字符串输出, 返回未传送的字符数. 例如, 传送“ABCDEFGH”成功时返回0, 全部未传送时返回8, 传送失败时返回负数.

通过NET_STROUT 函数输出的数据自动追加CR+LF(默认设置) 或 CR, LF字符传送, 从而判断字符串的结尾, 这种符号称为换行符.

1) <文件描述编号>中输入NET_FD 或 NET_OPEN 指令返回的文件描述符 (File descriptor)编号.
2) <字符串>输入传送的字符串.

Return value

<变量>	Data type: <整数型> 正常完成传送时: 返回0 未完成传送时: 返回未传送的字符数 传送发生错误时: 返回负数

Arguments

<文件描述编号>	Data type: <整数型> 文件描述编号的范围为 -1 ~ 1,024.
<字符串>	Data type: <字符串>

Examples

```
MAIN                                    // 开始主函数
     INT FD_NUM                         // 整数型变量声明
     DEFSTR SENDPK                      // 字符串变量声明
     INT LEN                            // 整数型变量声明
     SENDPK = “SUCCESS”                 // 传送字符串初始化
     IO = NET_STROUT(FD_NUM, “TEST”)    // 传送“TEST” 字符串
     LEN = NET_STROUT(FD_NUM,SENDPK)    // 传送“SUCCESS”字符串
EOP                                     // 结束主函数
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_STRIN, NET_KEEPALIVE, NET_FD

第5章 修改历史

版本	日期	修改内容
1	2019.01.21	初始版本
2	2019.06.19	添加新命令 - 外部通讯（专用于TCP/IP 客户端的命令）
3	2020.02.24	1. 添加新命令 - CNET 输入输出 2. 修复命令语法 - 特定于TCP/IP Clinet的命令
4	2020.05.07	正确的命令语法 -添加RSTATE命令状态索引定义 -OUT和COUT指令规范更改

A. 参考文献

此章节指定使用本产品的所有Robot系统服务、修理或安装中需要的所有文献.
所有文献ID的第一个词表示控制器名称， 第二个词表示相应文献的省略名. 最后，标记语言和相应文献版本.

语言按照以下规则表示.

- 韩语: K
- 英语: E
- 中文: C
- 越南语: V

文献 ID	说明
N2-IM-C□□	**安装及使用说明书 (Installation and handling manual)** 控制器结构及安装，外部设备之间接口方法的说明.
N2-OM-C□□	**操作及运行说明书 (Operation manual)** 控制器及Teach Pendant的使用方法及参数设置， JOB 程序编辑， 附加功能等说明.
N2-PM-C□□	**开发说明书 (Programming manual)** Robostar Robot程序RRL(Robostar robot language) 制作方法及提供的命令相关说明.
N2-HM-C□□	**Unihost说明书 (Unihost manual)** Robostar Online PC 程序Unihost相关说明.
N2-AM-C□□	**Alarm及维护保修说明书 (Alarm and maintenance manual)** 相应控制器为基础的Robot系统中发生的问题相关信息及解决方法、处理步骤相关说明.

B. 危险阶段及符号

符号	阶段	含义
	DANGER	如不遵守指南可能发生事故、重伤或致命伤害及/或严重产品损坏.
	WARNING	如不遵守指南可能发生事故，因此有可能发生严重受伤、致命伤或产品损坏.
	CAUTION	如不遵守指南有可能发生事故，导致产品损坏.
	ELECTRICAL SHOCK	触电或触电符号表示存在重伤或死亡的电气性危险.
	NOTE	告知重要事实及条件.
	PROHIBITION	为正常使用产品禁止的事项.

C. 命令列表（字母顺序）

No.	命令	说明	Ref.
1	ABS	返回输入值绝对值的函数.	10.15
2	ACC	设置加速时间的百分比 (%).	7.3
3	ACOS	返回逆余弦函数的结果值.	10.27
4	AMOV	从当前位置连接经由点 1，2 的同时画圆弧移动.	6.6
5	ASC	返回字符串第一个字母的字符码整数值.	10.34
6	ASIN	返回反正弦函数的结果值.	10.25
7	ATAN	返回反正切函数的结果值.	10.29
8	ATAN2	返回第 2 反正切函数的结果值.	10.30
9	BINS	将输入的整数值转换为 2 进制字符串的函数.	10.35
10	BREAK	重复语句内退出重复语句时使用.	5.9
11	BTOJ	通过 Base 坐标系位置值计算出 Joint 坐标系位置值.	10.4
12	BTOU	通过 Base 坐标系位置值计算用户坐标系位置值.	10.7
13	CBIN	从现场总线读取指定字节编号的值.	8.7
14	CBOUT	向指定现场总线卡的字节编号中输出指定值.	8.8
15	CDIN	以整数类型读取现场总线卡中指定的双单词编号的值.	8.9
16	CDOUT	以整数类型向指定现场总线卡的双单词编号输出指定值.	8.10
17	CFIN	以实数类型读取现场总线卡中指定的双单词编号的值.	8.13
18	CFOUT	以实数类型向指定现场总线卡的双单词编号输出指定值.	8.14
19	CHR	返回输入的整数值对应的 ASCII 文字.	10.36
20	CIN	从现场总线读取指定比特编号的值.	8.5
21	CLOG	**※ 目前不支持.**	11.19
22	CMOV	从当前位置连接经由点 1，2 的同时画圆移动.	6.5
23	Comment	对程序执行没有任何作用.	3.2
24	CONTINUE	重复语句内不执行后续语句，立即执行重复语句后面部分及利用条件式分段时使用.	5.8
25	COS	返回余弦函数的结果值.	10.26
26	COUT	向指定现场总线卡的比特编号输出指定值.	8.6
27	CTHREAD	同时执行函数或程序.	9.1
28	CWIN	读取现场总线卡中指定的单词编号的值.	8.11
29	CWOUT	向指定现场总线卡的单词编号输出指定值.	8.12
30	DEC	设置减速时间的百分比(%).	7.4
31	DEFSTR	声明程序内部使用的字符串变量.	2.3
32	DEG	弧度角（弧度法)转换为角度值（六十分法)的函数.	10.16
33	DLAY	指定时间期间暂停程序处理.	7.6

34	ETHREAD	强制停止 Thread.	9.2
35	EXIT	停止 JOB 程序执行.	5.12
36	EXP	返回幂函数的结果值. 即为对数的逆函数.	10.20
37	F	系统中预声明的实数型全局变量，在整体程序通用的变量.	2.9
38	FLUSH	清除串口通信输入，输出缓冲.	10.37
39	FOR … TO … [BY] … NEXT	直到条件变为 false(0)重复执行程序块.	5.4
40	FOS	轴线段到达目标点之前变更轨道至下一个目标点.	7.5
41	FTOS	整数或实数值转换为字符串.	10.38
42	FUNC … END	表示函数开始及结束.	5.2
43	GAIN	转换 Gain 组.	7.11
44	GBASEDISP	求两个坐标之间 Base 坐标系基准移动量的函数.	10.12
45	GP	系统中预声明的 Joint 坐标系位置型全局变量，在整体程序通用的变量.	2.10
46	GTOOLDISP	求输入的两个坐标之间 Tool 坐标系基准移动量的函数..	10.13
47	GUSERDISP	求输入的两个坐标之间用户坐标系基准移动量的函数.	10.14
48	HERE	系统中预声明的变量，当前位置返回为 Joint 角度值.	2.13
49	HERE_REF	系统中预声明的变量，当前参照位置返回为 Joint 角度值.	2.14
50	HTOS	整数转换为 16 进制字符串.	10.39
51	I	系统中预声明的整数型全局变量，在整体程序通用的变量.	2.8
52	IF …THEN … ENDIF	根据条件表达式的运算结果执行程序块.	5.6
53	IN	以 1 比特为单位读取指定比特编号的值.	8.1
54	INCLUDE	将特定程序文件引入到当前位置，当成一个程序使用.	3.1
55	INT	声明程序内部使用的整数型变量.	2.1
56	IPE	设置到达目标位置为止的允许时间最大值. 目	7.12
57	JCALL	调用 JOB 文件立即执行.	5.10
58	JMOV	从当前位置移动到目标点.	6.3
59	JTOB	从 Joint 坐标系位置值计算出 Base 坐标系位置值.	10.3
60	JTOU	通过 Joint 坐标系位置值计算出用户坐标系位置值.	10.5
61	LABL … GOTO	指定跳转位置，移动到相应转移位置的命令.	5.7
62	LIMT	限制各轴的移动范围.	7.9
63	LMOV	从当前位置到目标点进行直线插值移动.	6.4
64	LN	返回自然对数的结果值.	10.21
65	LOG	返回常用对数的结果值.	10.22
66	MAIN … EOP	表示主程序的开始及结束.	5.1
67	MAX	比较输入的两个值，返回最大值.	10.32
68	MIN	比较输入的两个值，返回较小值.	10.31

69	MVR	系统中预声明的变量，返回整体移动距离中占的移动距离比例.	2.12
70	NET_CLOSE	结束控制器的 TCP I/P 通信，清除客户端套接字的指令.	13.2
71	NET_DELEMETER	TCP I/P 通信的输入/输出结束字符串设定指令	13.5
72	NET_FD	返回访问控制器 TCP I/P Server 的 Client Socket 文件描述符(File descriptor) 编号.	13.6
73	NET_KEEPALIVE	TCP I/P 通信过程中周期性检查通信状态，感知 LAN 断线或非正常会话结束的指令.	13.4
74	NET_OPEN	创建 TCP I/P Client 套接字，请求访问 Server 的指令.	13.1
75	NET_STATUS	确认控制器的 TCP I/P 通信状态的指令.	13.3
76	NET_STRIN	通过以太网通信读取输入的字符串.	13.7
77	NET_STROUT	通过以太网通信发送字符串.	13.8
78	OFFS	目标点在指定值上加减后移动.	7.7
79	OFFSCLR	解除设置的偏差.	7.8
80	Operators	执行特定数学或逻辑操作的符号.	4
81	OUT	输出指定的比特编号中的值.	8.2
82	P	系统中预声明的 Joint 坐标系位置型变量作为位置值保存到文件中，作为全局变量使用.	2.7
83	PEXIT	解除指定 Robot Thread 编号 JOB.	12.6
84	PIN	读取端口为单位的指定端口编号的值.	8.3
85	POS	声明程序内部使用的 Joint 坐标系位置型变量.	2.4
86	POUT	向指定端口编号输出指定值.	8.4
87	POW	求乘方值时使用的函数.	10.18
88	PRESTART	重新执行指定 Robot Thread 编号中停止的 JOB 程序.	12.5
89	PRINT	记录监控用日志.	11.17
90	PSEL	匹配指定 Robot Thread 编号中输入的 JOB 文件.	12.1
91	PSTART	执行指定 Robot Thread 编号中选择的 JOB 程序.	12.2
92	PSTATE	读取指定 Robot Thread 编号的状态.	12.4
93	PSTOP	停止执行指定 Robot Thread 编号中输入的 JOB 文件.	12.3
94	RAD	角度值(六十分法)转换为弧度角 (弧度法)的函数.	10.17
95	RAND	生成随机数.	10.33
96	READ_CTEMP	读取控制器内部温度值.	11.16
97	READ_DERR	返回指定轴发生的驱动 Alarm Code.	11.7
98	READ_LOAD	读取 Robot 各轴当前负荷率.	11.11
99	READ_MTEMP	读取 Robot 各轴当前电机编码器温度值.	11.15
100	READ_RPM	读取 Robot 各轴当前 RPM 值.	11.10
101	READ_TRQ	读取 Robot 各轴当前扭矩值.	11.14
102	READ_TRQMAX	读取 Robot 的各轴最大扭矩值.	11.12
103	READ_VEL	读取 Robot 各轴当前速度值.	11.13

104	REAL	声明程序内部可以使用的实数型变量.	2.2
105	RELBASE	计算以 Base 坐标系信息为基准移动基准位置中代入的移动量的位置值的函数.	10.9
106	RELTOOL	计算以 Tool 坐标系信息为基准移动基准位置中代入的移动量的位置值的函数	10.10
107	RELUSER	计算以用户坐标系信息为基准移动基准位置中代入的移动量的位置值的函数.	10.11
108	RERRCNT	返回 Alarm 发生的总个数.	11.4
109	RERRCODE	返回输入的索引编号对应的 Alarm Code.	11.5
110	RERROR	返回最后发生的 Alarm 编码.	11.3
111	RERRTEXT	返回输入的索引编号对应的 Alarm 内容.	11.6
112	RESET	解除所有发生的 Alarm.	11.2
113	RET	结束函数的执行返回值，恢复到调用函数的位置.	5.3
114	RND	小数部分四舍五入后返回整数值.	10.19
115	RSTATE	返回 Robot 状态的命令.	11.8
116	SETERR	产生用户自定义的 System emergency Alarm.	11.1
117	SIN	返回正弦结果值.	10.24
118	SLEFT	提取输入的字符串左侧字符串.	10.40
119	SLEN	返回输入的字符串长度.	10.41
120	SMID	从字符串指定位置开始提取输入的值长度字符串.	10.42
121	SPOS	返回字符串 1 中匹配字符串 2 的起始位置.	10.43
122	SQRT	返回平方根.	10.23
123	SRIGHT	提取输入的字符串的右侧字符串.	10.44
124	STOP	停止 Robot 动作.	5.11
125	STRIN	读取通过串口及 Ethernet 通信输入的字符串.	10.45
126	STROUT	通过串口及 Ethernet 通信发送字符串.	10.46
127	SVAL	字符串转换为数字.	10.47
128	SVOF	关闭伺服电机电源.	6.2
129	SVON	开启电机电源.	6.1
130	SWLIMIT	读取限制 Robot 各轴移动范围的参数值.	11.9
131	TACT_START	设置为测量工程时间的测量起始位置.	11.23
132	TACT_STOP	设置工程时间测量的测量结束位置.	11.24
133	TAN	返回正切函数的结果值.	10.28
134	TH_RESUME	重新执行停止的 Thread.	9.5
135	TH_STATE	确认 Thread 的当前状态.	9.3
136	TH_SUSPEND	停止执行中的 Thread.	9.4
137	TMR	系统中预声明的 Timer 变量， 指定计数器值或读取当前计数值时使用的变量.	2.11
138	TOOL	选择 Robot 上使用的 Tool 和相应坐标系.	10.1

139	TRQ	设置扭矩限制值冲突时发生 Alarm 的命令.	7.10
140	UPOS	声明程序内部使用的用户坐标系位置型变量.	2.6
141	USER	选择用户坐标系.	10.2
142	UTOB	通过用户坐标系位置值计算出 Base 坐标系位置值.	10.8
143	UTOJ	通过用户坐标系位置值计算出 Joint 坐标系位置值.	10.6
144	VEL	设置轴移动速度的千分比 (%).	7.2
145	WGFLT	将全局 Point 变量保存为文件.	11.22
146	WGINT	全局整数变量保存在文件中.	11.21
147	WGPNT	将全局 Point 变量保存为文件.	11.20
148	WHILE … ENDWL	条件表达式为真(true)时重复执行程序块.	5.5
149	WITH … ENDWT	Robot 动作的同时处理程序块内执行语句.	7.1
150	WLOG	以文件的形式保存包含时间信息的任意用户日志.	11.18
151	XBIN	参考指定的设备及字节编号，从 CNET 接收缓冲中读取值.	8.17
152	XBOUT	参考指定的设备及字节编号，向 CNET 设备发信缓冲发送值.	8.18
153	XDIN	参考指定的设备及 Double Word 编号，从 CNET 接收缓冲中读取值.	8.21
154	XDOUT	参考指定的设备及 Double Word 编号，向 CNET 设备发信缓冲发送值.	8.22
155	XFIN	参考指定的设备及 Double Word 编号，从 CNET 接收缓冲中读取实数型值.	8.23
156	XFOUT	参考指定的设备及 Double Word 编号，向 CNET 设备发信缓冲输出实数型值.	8.24
157	XIN	参考指定的设备及比特编号，从 CNET 接收缓冲读取值.	8.15
158	XOUT	参考指定的设备及比特编号，向 CNET 设备发信缓冲输出值.	8.16
159	XPOS	声明程序内部使用的 Base 坐标系位置型变量.	2.5
160	XWIN	参考指定的设备及 Word 编号，从 CNET 接收缓冲读取值.	8.19
161	XWOUT	参考指定设备及 Word 编号，向 CNET 设备发信缓冲输出值.	8.20

N2 Series Controller

Programming Manual
Fourth edition, May 07, 2020

ROBOSTAR CO., LTD.
ROBOT R&D CENTER