MITSUBISHI

三菱可编程控制器

MELSEC Q 系列　　MELSEC L 系列

MELSEC-Q/L结构体编程手册

结构化文本篇

● 安全注意事项 ●

（使用之前请务必阅读）

在使用MELSEC-Q系列、MELSEC-L系列可编程控制器之前，应仔细阅读各产品附带的手册及附带手册中所介绍的关联手册，同时在充分注意安全的前提下正确地操作。

请妥善保管产品附带手册，放置于操作人员易于取阅的地方，并应将本手册交给最终用户。

A - 1

● 关于产品的应用 ●

(1) 在使用三菱可编程控制器时，应该符合以下条件:即使在可编程控制器设备出现问题或故障时也不会导致重大事故，并且应在设备外部系统地配备能应付任何问题或故障的备用设备及失效保险功能。

(2) 三菱可编程控制器是以一般工业用途等为对象设计和制造的通用产品。因此，三菱可编程控制器不应用于以下设备・系统等特殊用途。如果用于以下特殊用途，对于三菱可编程控制器的质量、性能、安全等所有相关责任(包括但不限于债务未履行责任、瑕疵担保责任、质量保证责任、违法行为责任、制造物责任)，三菱将不负责。

・面向各电力公司的核电站以及其它发电厂等对公众有较大影响的用途。
・用于各铁路公司或公用设施目的等有特殊质量保证体系要求的用途。
・航空航天、医疗、铁路、焚烧・燃料装置、载人移动设备、载人运输装置、娱乐设备、安全设备等预计对人身财产有较大影响的用途。

然而，对于上述应用，如果在限定于具体用途，无需特殊质量(超出一般规格的质量等)要求的条件下，经过三菱的判断也可以使用三菱可编程控制器，详细情况请与当地三菱代表机构协商。

A - 2

修订记录

*本手册号在封底的左下角。

印刷日期	*手册编号	修改内容
2010年04月	SH(NA)-080907CHN-A	第一版

日文手册原稿: SH-080363-I

A - 3

前言

在此感谢贵方购买了三菱电机的FA软件MELSOFT系列的产品。
在使用之前应熟读本书，在充分了解MELSEC系列可编程控制器的功能• 性能的基础上正确地使用本产品。
此外，应将本手册交给最终用户。

目录

安全注意事项 A - 1
关于产品的应用 A - 2
修订记录 A - 3
前言 A - 4
目录 A - 4
关于手册 A - 12
手册阅读方法 A - 13
本手册中使用的总称 • 略称 A - 14

1 概要 1 - 1～1 - 4

1.1 关于ST语言 1 - 1
1.2 MELSEC-Q/L系列中的ST程序的特点 1 - 3
1.3 ST程序创建步骤 1 - 4

2 系统配置 2- 1～2- 4

2.1 系统配置 2 - 1
2.1.1 适用CPU 2 - 1
2.1.2 ST程序用编程工具 2 - 1
2.1.3 ST程序规格 2 - 1

3 ST程序的字符及数值的处理 3 - 1～3 - 16

3.1 可用字符 3 - 1
3.2 数据的处理 3 - 3
3.2.1 数据类型 3 - 3
3.2.2 关于ANY类型 3 - 4
3.2.3 数组及结构体 3 - 5
3.3 数据的表示方法 3 - 8
3.3.1 常数 3 - 8
3.3.2 标签 3 - 11
3.3.3 软元件 3 - 14

4 ST程序的表达式 4 - 1～4 - 32

4.1 代入语句 4 - 1
4.2 运算符 4 - 2
4.2.1 运算符列表 4 - 2
4.2.2 运算符的使用示例 4 - 4

4.3 控制语句 ····· 4 - 6
4.3.1 控制语句列表 ····· 4 - 6
4.3.2 条件语句 ····· 4 - 7
4.3.3 循环语句 ····· 4 - 15
4.3.4 其它控制语句 ····· 4 - 20
4.3.5 使用控制语句时的注意事项 ····· 4 - 22
4.4 功能块的调用 ····· 4 - 29
4.5 注释 ····· 4 - 32

5 MELSEC函数 5 - 1～5 - 110

函数的阅读方法 ····· 5 - 1
5.1 输出 ····· 5 - 4
5.1.1 软元件的输出 ····· OUT_M ····· 5 - 4
5.1.2 低速型 ····· TIMER_M ····· 5 - 4
5.1.3 高速型 ····· TIMER_H_M ····· 5 - 5
5.1.4 计数器 ····· COUNTER_M ····· 5 - 5
5.1.5 软元件的设置 ····· SET_M ····· 5 - 6
5.1.6 软元件的复位 ····· RST_M ····· 5 - 6
5.1.7 直接输出的脉冲化 ····· DELTA_M ····· 5 - 7
5.2 移动 ····· 5 - 8
5.2.1 软元件的1位移动 ····· SFT_M ····· 5 - 8
5.3 结束 ····· 5 - 9
5.3.1 停止 ····· STOP_M ····· 5 - 9
5.4 比较运算 ····· 5 - 10
5.4.1 块数据比较(=) ····· BKCMP_EQ_M ····· 5 - 10
5.4.2 块数据比较(<>) ····· BKCMP_NE_M ····· 5 - 10
5.4.3 块数据比较(>) ····· BKCMP_GT_M ····· 5 - 11
5.4.4 块数据比较(<=) ····· BKCMP_LE_M ····· 5 - 11
5.4.5 块数据比较(<) ····· BKCMP_LT_M ····· 5 - 12
5.4.6 块数据比较(>=) ····· BKCMP_GE_M ····· 5 - 12
5.5 算术运算 ····· 5 - 13
5.5.1 BCD4位的加法(2软元件) ····· BPLUS_M ····· 5 - 13
5.5.2 BCD4位的加法(3软元件) ····· BPLUS_3_M ····· 5 - 13
5.5.3 BCD4位的减法(2软元件) ····· BMINUS_M ····· 5 - 14
5.5.4 BCD4位的减法(3软元件) ····· BMINUS_3_M ····· 5 - 14
5.5.5 BCD8位的加法(2软元件) ····· DBPLUS_M ····· 5 - 15
5.5.6 BCD8位的加法(3软元件) ····· DBPLUS_3_M ····· 5 - 15
5.5.7 BCD8位的减法(2软元件) ····· DBMINUS_M ····· 5 - 16
5.5.8 BCD8位的减法(3软元件) ····· DBMINUS_3_M ····· 5 - 16
5.5.9 BCD4位的乘法 ····· BMULTI_M ····· 5 - 17
5.5.10 BCD4位的除法 ····· BDIVID_M ····· 5 - 17
5.5.11 BCD8位的乘法 ····· DBMULTI_M ····· 5 - 18
5.5.12 BCD8位的除法 ····· DBDIVID_M ····· 5 - 18
5.5.13 字符串数据合并(2软元件) ····· STRING_PLUS_M ····· 5 - 19
5.5.14 字符串数据合并(3软元件) ····· STRING_PLUS_3_M ····· 5 - 19
5.5.15 BIN块加法 ····· BKPLUS_M ····· 5 - 20
5.5.16 BIN块减法 ····· BKMINUS_M ····· 5 - 20
5.5.17 递增 ····· INC_M ····· 5 - 21

5.5.18 递减 ····· DEC_M ····· 5 - 21
5.5.19 32位BIN递增 ····· DINC_M ····· 5 - 22
5.5.20 32位BIN递减 ····· DDEC_M ····· 5 - 22
5.6 数据转换 ····· 5 - 23
5.6.1 BIN→BCD转换 ····· BCD_M ····· 5 - 23
5.6.2 32位BIN→BCD转换 ····· DBCD_M ····· 5 - 23
5.6.3 BCD→BIN转换 ····· BIN_M ····· 5 - 24
5.6.4 32位BCD→BIN转换 ····· DBIN_M ····· 5 - 24
5.6.5 浮动小数点→BIN转换 ····· INT_E_MD ····· 5 - 25
5.6.6 32位浮动小数点→BIN转换 ····· DINT_E_MD ····· 5 - 25
5.6.7 BIN→浮动小数点转换 ····· FLT_M ····· 5 - 26
5.6.8 32位BIN→浮动小数点转换 ····· DFLT_M ····· 5 - 26
5.6.9 16位BIN→32位BIN转换 ····· DBL_M ····· 5 - 27
5.6.10 32位BIN→16位BIN转换 ····· WORD_M ····· 5 - 27
5.6.11 BIN→格雷码转换 ····· GRY_M ····· 5 - 28
5.6.12 32位BIN→格雷码转换 ····· DGRY_M ····· 5 - 28
5.6.13 格雷码→BIN转换 ····· GBIN_M ····· 5 - 29
5.6.14 32位格雷码→BIN转换 ····· DGBIN_M ····· 5 - 29
5.6.15 16位BIN的2的补数 ····· NEG_M ····· 5 - 30
5.6.16 32位BIN的2的补数 ····· DNEG_M ····· 5 - 30
5.6.17 浮动小数点的2的补数 ····· ENEG_M ····· 5 - 31
5.6.18 块转换BIN→BCD转换 ····· BKBCD_M ····· 5 - 31
5.6.19 块转换BCD→BIN转换 ····· BKBIN_M ····· 5 - 32
5.7 数据传送 ····· 5 - 33
5.7.1 16位数据否定传送 ····· CML_M ····· 5 - 33
5.7.2 32位数据否定传送 ····· DCML_M ····· 5 - 33
5.7.3 块传送 ····· BMOV_M ····· 5 - 34
5.7.4 同一数据块传送 ····· FMOV_M ····· 5 - 34
5.7.5 16位数据替换 ····· XCH_M ····· 5 - 35
5.7.6 32位数据替换 ····· DXCH_M ····· 5 - 35
5.7.7 块数据替换• ····· BXCH_M ····· 5 - 36
5.7.8 高低字节替换 ····· SWAP_MD ····· 5 - 36
5.8 程序执行控制 ····· 5 - 37
5.8.1 中断禁止 ····· DI_M ····· 5 - 37
5.8.2 中断允许 ····· EI_M ····· 5 - 37
5.9 I/O刷新 ····· 38
5.9.1 I/O刷新 ····· RFS_M ····· 5 - 38
5.10 逻辑运算指令 ····· 5 - 39
5.10.1 逻辑积(2软元件) ····· WAND_M ····· 5 - 39
5.10.2 逻辑积(3软元件) ····· WAND_3_M ····· 5 - 39
5.10.3 32位数据逻辑积(2软元件) ····· DAND_M ····· 5 - 40
5.10.4 32位数据逻辑积(3软元件) ····· DAND_3_M ····· 5 - 40
5.10.5 块数据逻辑积 ····· BKAND_M ····· 5 - 41
5.10.6 逻辑和(2软元件) ····· WOR_M ····· 5 - 41
5.10.7 逻辑和(3软元件) ····· WOR_3_M ····· 5 - 42
5.10.8 32位数据逻辑和(2软元件) ····· DOR_M ····· 5 - 42
5.10.9 32位数据逻辑和(3软元件) ····· DOR_3_M ····· 5 - 43
5.10.10 块数据逻辑和 ····· BKOR_M ····· 5 - 43

5.10.11 排他逻辑和(2软元件) ····· WXOR_M ····· 5 - 44
5.10.12 排他逻辑和(3软元件) ····· WXOR_3_M ····· 5 - 44
5.10.13 32位数据排他逻辑和(2软元件) ····· DXOR_M ····· 5 - 45
5.10.14 32位数据排他逻辑和(3软元件) ····· DXOR_3_M ····· 5 - 45
5.10.15 块数据排他逻辑和 ····· BKXOR_M ····· 5 - 46
5.10.16 否定排他逻辑和(2软元件) ····· WXNR_M ····· 5 - 46
5.10.17 否定排他逻辑和(3软元件) ····· WXNR_3_M ····· 5 - 47
5.10.18 32位数据否定排他逻辑和(2软元件) ····· DXNR_M ····· 5 - 47
5.10.19 32位数据否定排他逻辑和(3软元件) ····· DXNR_3_M ····· 5 - 48
5.10.20 块数据否定排他逻辑和 ····· BKXNR_M ····· 5 - 48
5.11 旋转 ····· 5 - 49
5.11.1 右旋转(不包含进位标志) ····· ROR_M ····· 5 - 49
5.11.2 右旋转(包含进位标志) ····· RCR_M ····· 5 - 49
5.11.3 左旋转(不包含进位标志) ····· ROL_M ····· 5 - 50
5.11.4 左旋转(包含进位标志) ····· RCL_M ····· 5 - 50
5.11.5 32位数据右旋转(不包含进位标志) ····· DROR_M ····· 5 - 51
5.11.6 32位数据右旋转(包含进位标志) ····· DRCR_M ····· 5 - 51
5.11.7 32位数据左旋转(不包含进位标志) ····· DROL_M ····· 5 - 52
5.11.8 32位数据左旋转(包含进位标志) ····· DRCL_M ····· 5 - 52
5.12 移动 ····· 5 - 53
5.12.1 n位右移 ····· SFR_M ····· 5 - 53
5.12.2 n位左移 ····· SFL_M ····· 5 - 53
5.12.3 n位数据1位右移 ····· BSFR_M ····· 5 - 54
5.12.4 n位数据1位左移 ····· BSFL_M ····· 5 - 54
5.12.5 1字右移 ····· DSFR_M ····· 5 - 55
5.12.6 1字左移 ····· DSFL_M ····· 5 - 55
5.13 位处理 ····· 5 - 56
5.13.1 字软元件的位设置 ····· BSET_M ····· 5 - 56
5.13.2 字软元件的位复位 ····· BRST_M ····· 5 - 56
5.13.3 字软元件的位测试 ····· TEST_MD ····· 5 - 57
5.13.4 32位数据的位测试 ····· DTEST_MD ····· 5 - 57
5.13.5 位软元件批量复位 ····· BKRST_M ····· 5 - 58
5.14 数据处理 ····· 5 - 59
5.14.1 数据查找 ····· SER_M ····· 5 - 59
5.14.2 32位数据查找 ····· DSER_M ····· 5 - 59
5.14.3 位校验 ····· SUM_M ····· 5 - 60
5.14.4 32位数据位校验 ····· DSUM_M ····· 5 - 60
5.14.5 编译 ····· DECO_M ····· 5 - 61
5.14.6 编码 ····· ENCO_M ····· 5 - 61
5.14.7 7段编译 ····· SEG_M ····· 5 - 62
5.14.8 16位数据的4位分离 ····· DIS_M ····· 5 - 62
5.14.9 16位数据的4位合并 ····· UNI_M ····· 5 - 63
5.14.10 任意数据的位分离 ····· NDIS_M ····· 5 - 63
5.14.11 任意数据的位合并 ····· NUNI_M ····· 5 - 64
5.14.12 字节单位数据分离 ····· WTOB_MD ····· 5 - 64
5.14.13 字节单位数据合并 ····· BTOW_MD ····· 5 - 65
5.14.14 数据最大值查找 ····· MAX_M ····· 5 - 65
5.14.15 32位数据最大值查找 ····· DMAX_M ····· 5 - 66
5.14.16 数据最小值查找 ····· MIN_M ····· 5 - 66

5.14.17 32位数据最小值查找 ····· DMIN_M ····· 5 - 67
5.14.18 数据排序 ····· SORT_M ····· 5 - 67
5.14.19 32位数据排序 ····· DSORT_M ····· 5 - 68
5.14.20 合计值计算 ····· WSUM_M ····· 5 - 68
5.14.21 32位数据合计值计算 ····· DWSUM_M ····· 5 - 69
5.15 结构化 ····· 5 - 70
5.15.1 刷新 ····· COM_M ····· 5 - 70
5.16 缓冲存储器访问 ····· 5 - 71
5.16.1 智能功能模块1字数据读取 ····· FROM_M ····· 5 - 71
5.16.2 智能功能模块2字数据读取 ····· DFRO_M ····· 5 - 71
5.16.3 智能功能模块1字数据写入 ····· TO_M ····· 5 - 72
5.16.4 智能功能模块2字数据写入 ····· DTO_M ····· 5 - 72
5.17 字符串处理 ····· 5 - 73
5.17.1 BIN→10进制ASCII转换 ····· BINDA_S_MD ····· 5 - 73
5.17.2 32位BIN→10进制ASCII转换 ····· DBINDA_S_MD ····· 5 - 73
5.17.3 BIN→16进制ASCII转换 ····· BINHA_S_MD ····· 5 - 74
5.17.4 32位BIN→16进制ASCII转换 ····· DBINHA_S_MD ····· 5 - 74
5.17.5 BCD4位→10进制ASCII转换 ····· BCDDA_S_MD ····· 5 - 75
5.17.6 BCD8位→10进制ASCII转换 ····· DBCDDA_S_MD ····· 5 - 75
5.17.7 10进制ASCII→BIN转换 ····· DABIN_S_MD ····· 5 - 76
5.17.8 10进制ASCII→32位BIN转换 ····· DDABIN_S_MD ····· 5 - 76
5.17.9 16进制ASCII→BIN转换 ····· HABIN_S_MD ····· 5 - 77
5.17.10 16进制ASCII→32位BIN转换 ····· DHABIN_S_MD ····· 5 - 77
5.17.11 10进制ASCII→BCD4位转换 ····· DABCD_S_MD ····· 5 - 78
5.17.12 10进制ASCII→BCD8位转换 ····· DDABCD_S_MD ····· 5 - 78
5.17.13 软元件的注释数据读取 ····· COMRD_S_MD ····· 5 - 79
5.17.14 字符串的长度检测 ····· LEN_S_MD ····· 5 - 79
5.17.15 BIN→字符串转换 ····· STR_S_MD ····· 5 - 80
5.17.16 32位BIN→字符串转换 ····· DSTR_S_MD ····· 5 - 80
5.17.17 字符串→BIN转换 ····· VAL_S_MD ····· 5 - 81
5.17.18 字符串→32位BIN转换 ····· DVAL_S_MD ····· 5 - 81
5.17.19 浮动小数点→字符串转换 ····· ESTR_M ····· 5 - 82
5.17.20 字符串→浮动小数点转换 ····· EVAL_M ····· 5 - 82
5.17.21 BIN→ASCII转换 ····· ASC_S_MD ····· 5 - 83
5.17.22 ASCII→BIN转换 ····· HEX_S_MD ····· 5 - 83
5.17.23 从字符串右侧截取 ····· RIGHT_M ····· 5 - 84
5.17.24 从字符串左侧截取 ····· LEFT_M ····· 5 - 84
5.17.25 字符串中的任意截取 ····· MIDR_M ····· 5 - 85
5.17.26 字符串中的任意替换 ····· MIDW_M ····· 5 - 85
5.17.27 字符串查找 ····· INSTR_M ····· 5 - 86
5.17.28 浮动小数点→BCD分解 ····· EMOD_M ····· 5 - 86
5.17.29 BCD格式数据→浮动小数点 ····· EREXP_M ····· 5 - 87
5.18 特殊函数 ····· 5 - 88
5.18.1 浮动小数点SIN运算 ····· SIN_E_MD ····· 5 - 88
5.18.2 浮动小数点COS运算 ····· COS_E_MD ····· 5 - 88
5.18.3 浮动小数点TAN运算 ····· TAN_E_MD ····· 5 - 89
5.18.4 浮动小数点SIN^{-1}运算 ····· ASIN_E_MD ····· 5 - 89
5.18.5 浮动小数点COS^{-1}运算 ····· ACOS_E_MD ····· 5 - 90
5.18.6 浮动小数点TAN^{-1}运算 ····· ATAN_E_MD ····· 5 - 90

5.18.7 浮动小数点角度→弧度 ····· RAD_E_MD ····· 5 - 91
5.18.8 浮动小数点弧度→角度转换 ····· DEG_E_MD ····· 5 - 91
5.18.9 浮动小数点平方根 ····· SQR_E_MD ····· 5 - 92
5.18.10 浮动小数点指数运算 ····· EXP_E_MD ····· 5 - 92
5.18.11 浮动小数点自然对数运算 ····· LOG_E_MD ····· 5 - 93
5.18.12 随机数发生 ····· RND_M ····· 5 - 93
5.18.13 系列变更 ····· SRND_M ····· 5 - 94
5.18.14 BCD4位平方根 ····· BSQR_MD ····· 5 - 94
5.18.15 BCD8位平方根 ····· BDSQR_MD ····· 5 - 95
5.18.16 BCD型SIN运算 ····· BSIN_MD ····· 5 - 95
5.18.17 BCD型COS运算 ····· BCOS_MD ····· 5 - 96
5.18.18 BCD型TAN运算 ····· BTAN_MD ····· 5 - 96
5.18.19 BCD型SIN^{-1}运算 ····· BASIN_MD ····· 5 - 97
5.18.20 BCD型COS^{-1}运算 ····· BACOS_MD ····· 5 - 97
5.18.21 BCD型TAN^{-1}运算 ····· BATAN_MD ····· 5 - 98
5.19 数据控制 ····· 5 - 99
5.19.1 上下限极限控制 ····· LIMIT_MD ····· 5 - 99
5.19.2 32位数据上下限极限控制 ····· DLIMIT_MD ····· 5 - 99
5.19.3 死区控制 ····· BAND_MD ····· 5 - 100
5.19.4 32位数据死区控制 ····· DBAND_MD ····· 5 - 100
5.19.5 位域控制 ····· ZONE_MD ····· 5 - 101
5.19.6 32位数据位域控制 ····· DZONE_MD ····· 5 - 101
5.19.7 文件寄存器的块No.切换 ····· RSET_MD ····· 5 - 102
5.19.8 文件寄存器用文件的设置 ····· QDRSET_M ····· 5 - 102
5.19.9 注释用文件的设置 ····· QCDSET_M ····· 5 - 103
5.20 时钟 ····· 5 - 104
5.20.1 时钟数据的读取 ····· DATERD_MD ····· 5 - 104
5.20.2 时钟数据的写入 ····· DATEWR_MD ····· 5 - 104
5.20.3 时钟数据的加法 ····· DATEPLUS_M ····· 5 - 105
5.20.4 时钟数据的减法 ····· DATEMINUS_M ····· 5 - 105
5.20.5 时钟数据格式转换(时、分、秒→秒) ····· SECOND_M ····· 5 - 106
5.20.6 时钟数据格式转换(秒→时、分、秒) ····· HOUR_M ····· 5 - 106
5.21 程序控制 ····· 5 - 107
5.21.1 程序待机 ····· PSTOP_M ····· 5 - 107
5.21.2 程序输出OFF待机 ····· POFF_M ····· 5 - 107
5.21.3 程序扫描执行登录 ····· PSCAN_M ····· 5 - 108
5.21.4 程序低速执行登录 ····· PLOW_M ····· 5 - 108
5.22 其它 ····· 5 - 109
5.22.1 WDT复位 ····· WDT_M ····· 5 - 109

6 IEC函数 6 - 1～6 - 78

函数的阅读方法 ····· 6 - 1
6.1 类型转换功能 ····· 6 - 3
6.1.1 布尔型(BOOL)→双精度整数型(DINT)转换 ····· BOOL_TO_DINT(_E) ····· 6 - 3
6.1.2 布尔型(BOOL)→整数型(INT)转换 ····· BOOL_TO_INT(_E) ····· 6 - 4
6.1.3 布尔型(BOOL)→字符串型(STRING)转换 ····· BOOL_TO_STR(_E) ····· 6 - 5
6.1.4 双精度整数型(DINT)→布尔型(BOOL)转换 ····· DINT_TO_BOOL(_E) ····· 6 - 6
6.1.5 双精度整数型(DINT)→整数型(INT)转换 ····· DINT_TO_INT(_E) ····· 6 - 7

6.1.6 双精度整数型(DINT)→实数型(REAL)转换 ····· DINT_TO_REAL(_E) ····· 6 - 8
6.1.7 双精度整数型(DINT)→字符串型(STRING)转换 ····· DINT_TO_STR(_E) ····· 6 - 9
6.1.8 整数型(INT)→布尔型(BOOL)转换 ····· INT_TO_BOOL(_E) ····· 6 - 10
6.1.9 整数型(INT)→双精度整数型(DINT)转换 ····· INT_TO_DINT(_E) ····· 6 - 11
6.1.10 整数型(INT)→实数型(REAL)转换 ····· INT_TO_REAL(_E) ····· 6 - 12
6.1.11 整数型(INT)→字符串型(STRING)转换 ····· INT_TO_STR(_E) ····· 6 - 13
6.1.12 实数型(REAL)→双精度整数型(DINT)转换 ····· REAL_TO_DINT(_E) ····· 6 - 14
6.1.13 实数型(REAL)→整数型(INT)转换 ····· REAL_TO_INT(_E) ····· 6 - 15
6.1.14 实数型(REAL)→字符串型(STRING)转换 ····· REAL_TO_STR(_E) ····· 6 - 16
6.1.15 字符串型(STRING)→布尔型(BOOL)转换 ····· STR_TO_BOOL(_E) ····· 6 - 17
6.1.16 字符串型(STRING)→双精度整数型(DINT)转换 ····· STR_TO_DINT(_E) ····· 6 - 18
6.1.17 字符串型(STRING)→整数型(INT)转换 ····· STR_TO_INT(_E) ····· 6 - 19
6.1.18 字符串型(STRING)→实数型(REAL)转换 ····· STR_TO_REAL(_E) ····· 6 - 20
6.2 数值功能(一般函数) ····· 6 - 21
6.2.1 绝对值 ····· ABS(_E) ····· 6 - 21
6.2.2 平方根 ····· SQRT(_E) ····· 6 - 22
6.3 数值功能(对数函数) ····· 6 - 23
6.3.1 自然对数 ····· LN(_E) ····· 6 - 23
6.3.2 自然指数 ····· EXP(_E) ····· 6 - 24
6.4 数值功能(三角函数) ····· 6 - 25
6.4.1 浮动小数点SIN运算 ····· SIN(_E) ····· 6 - 25
6.4.2 浮动小数点COS运算 ····· COS(_E) ····· 6 - 26
6.4.3 浮动小数点TAN运算 ····· TAN(_E) ····· 6 - 27
6.4.4 浮动小数点SIN^{-1}运算 ····· ASIN(_E) ····· 6 - 28
6.4.5 浮动小数点COS^{-1}运算 ····· ACOS(_E) ····· 6 - 29
6.4.6 浮动小数点TAN^{-1}运算 ····· ATAN(_E) ····· 6 - 30
6.5 算术运算功能 ····· 6 - 31
6.5.1 加法 ····· ADD_E ····· 6 - 31
6.5.2 乘法 ····· MUL_E ····· 6 - 32
6.5.3 减法 ····· SUB_E ····· 6 - 33
6.5.4 除法 ····· DIV_E ····· 6 - 34
6.5.5 剩余 ····· MOD(_E) ····· 6 - 35
6.5.6 指数 ····· EXPT(_E) ····· 6 - 36
6.5.7 代入 ····· MOVE(_E) ····· 6 - 38
6.6 位移功能 ····· 6 - 39
6.6.1 位左移 ····· SHL(_E) ····· 6 - 39
6.6.2 位右移 ····· SHR(_E) ····· 6 - 40
6.6.3 右旋转 ····· ROR(_E) ····· 6 - 41
6.6.4 左旋转 ····· ROL(_E) ····· 6 - 42
6.7 位型布尔功能 ····· 6 - 43
6.7.1 逻辑积 ····· AND_E ····· 6 - 43
6.7.2 逻辑和 ····· OR_E ····· 6 - 44
6.7.3 排他逻辑和 ····· XOR_E ····· 6 - 45
6.7.4 逻辑否定 ····· NOT(_E) ····· 6 - 46
6.8 选择功能 ····· 6 - 47
6.8.1 二进制的选择 ····· SEL(_E) ····· 6 - 47
6.8.2 最大值 ····· MAX(_E) ····· 6 - 49
6.8.3 最小值 ····· MIN(_E) ····· 6 - 51

6.8.4 限制器 ····· LIMIT(_E) ····· 6 - 53
6.8.5 多路调制器 ····· MUX(_E) ····· 6 - 55
6.9 比较功能 ····· 6 - 57
6.9.1 大于号(>) ····· GT_E ····· 6 - 57
6.9.2 大于等于号(>=) ····· GE_E ····· 6 - 59
6.9.3 等号(=) ····· EQ_E ····· 6 - 61
6.9.4 小于等于号(<=) ····· LE_E ····· 6 - 63
6.9.5 小于号(<) ····· LT_E ····· 6 - 65
6.9.6 不等号(<>) ····· NE_E ····· 6 - 67
6.10 字符串功能 ····· 6 - 69
6.10.1 字符串长度截取 ····· LEN(_E) ····· 6 - 69
6.10.2 从字符串的开始位置截取 ····· LEFT(_E) ····· 6 - 70
6.10.3 从字符串的终端截取 ····· RIGHT(_E) ····· 6 - 71
6.10.4 从字符串的指定位置截取 ····· MID(_E) ····· 6 - 72
6.10.5 字符串的连接 ····· CONCAT(_E) ····· 6 - 73
6.10.6 至指定位置的字符串插入 ····· INSERT(_E) ····· 6 - 74
6.10.7 从字符串的指定位置删除 ····· DELETE(_E) ····· 6 - 75
6.10.8 从字符串的指定位置替换 ····· REPLACE(_E) ····· 6 - 76
6.10.9 从字符串的指定位置查找 ····· FIND(_E) ····· 6 - 77

7 出错列表 7 - 1～7 - 12

附录 附- 1～附- 4

附录1 标签・FB名中不能使用的字符串 ····· 附 - 1
附录2 GX Developer与GX Works2中ST指令对应表 ····· 附 - 3

索引 索引- 1～索引- 8

关于手册

与本产品有关的手册如下所示。
请根据需要参考本表订购。

关联手册

手册名称	手册编号
GX Developer Version8操作手册(入门篇) 对GX Developer的系统构成、安装方法、启动方法的有关内容进行说明。 (另售)	SH-080740CHN
GX Developer Version8操作手册 对GX Developer的程序创建方法、打印方法、监视方法、调试方法等有关内容进行说明。 (另售)	SH-080311C
GX Developer Version8操作手册(功能块篇) 对GX Developer的功能块的创建，打印方法等有关内容进行说明。 (另售)	SH-080639CHN
GX Developer Version8操作手册(结构化文本篇) 对GX Developer的结构化文本(ST)程序的创建方法、打印方法等有关内容进行说明。 (另售)	SH-080666CHN
结构化文本(ST)编程指南 是以初次进行结构化文本(ST)程序创建的人员为对象的参考手册。通过样本程序对基本的操作方法以及功能进行说明。 (另售)	SH-080665CHN
MELSEC-Q/L编程手册(公共指令篇) 顺控指令、基本指令以及应用指令的使用方法等有关内容进行说明。 (另售)	SH-080814CHN

备注

各操作手册与软件包一道被刻录在同一个CD-ROM中。
备有用于另售的印刷品，希望单独购买手册的情况下，请通过上述表格中的手册编号购买。

A - 12

该手册. . .

本手册用于使用GX Developer进行结构化文本(以下略称为ST)编程。适用于具有可编程控制器・梯形图程序的相关知识及编程经验的用户，以及具有C语言相关知识及编程经验的用户。
“第1章 概要”记载了ST语言的概要、ST编程的特点、ST程序的创建步骤。
“第2章 系统配置”记载了适用CPU、ST程序规格等。
“第3章 ST程序的字符及数值的处理”记载了ST程序中使用的数据类型及表示方法。
“第4章 ST程序的公式”记载了ST程序中使用的运算符、控制语句等的公式。
“第5章 MELSEC函数”、“第6章 IEC函数”记载了ST程序中使用的函数的自变量・返回值・表示示例。

操作手册. . .

“GX Developer Version8 操作手册(ST篇)”详细说明了用于ST编程的所有菜单及菜单选项。对于操作的详细信息请根据需要进行参照。
需要除ST编程以外的操作信息的情况下，请参阅“GX Developer Version8 操作手册”，或者“GX Developer Version8 操作手册(入门篇)”。

初次使用ST语言的情况下. . .

请参阅“初次使用ST”。其中记载了ST语言的概要、用于通过GX Developer创建ST程序并写入到可编程控制器CPU模块中的步骤等必要信息。

具有ST语言方面的知识，希望立即进行编程的情况下. . .

请跳跃到“第5章 MELSEC函数”。其中记载了在ST程序在6使用函数的必要事项。希望了解ST程序中使用的数据的情况下，请参阅“第3章 ST程序的字符及数值的处理”。其中记载了ST程序中使用的数据类型及表示方法。希望在ST程序中使用控制语句的情况下，请参阅“第4章 ST程序表达式”。其中记载了ST程序中使用的控制语句格式及表示示例。

以下对本手册中使用的符号及内容进行说明。

符号	内容	示例
要点	记载作为该项目相关知识应预先了解的内容。	
备注	记载作为该项目相关知识预先了解可带来方便的内容。	备注
[ ]	菜单栏的菜单名	[工程]

A - 13

本手册中使用的总称・略称

在本手册中，将GX Developer软件包、可编程控制器CPU模块等以如下所示的总称・略称表示。在需要标明相关型号的情况下，将记载模块型号。

总称/略称	内容/对象模块
GX Developer	产品型号SWnD5C-GPPW、SWnD5C-GPPW-A、SWnD5C-GPPW-V、SWnD5C-GPPW-VA的总称产品名。(n=版本8以后)
GX Works2	产品型号SWnDNC-GXW2的总称产品名。 (n=版本)
ST	结构化文本的略称。
FB	功能块的略称。
基本型QCPU	功能版本B以后的Q00JCPU、Q00CPU、Q01CPU的总称。
通用型QCPU	Q00UJCPU、Q00UCPU、Q01UCPU、Q02UCPU、Q03UDCPU、Q03UDECPU、Q04UDHCPU、Q04UDEHCPU、Q06UDHCPU、Q06UDEHCPU、Q10UDHCPU、Q10UDEHCPU、Q13UDHCPU、Q13UDEHCPU、Q20UDHCPU、Q20UDEHCPU、Q26UDHCPU、Q26UDEHCPU的总称。
高性能型QCPU	Q02(H)CPU、Q06CPU、Q12HCPU、Q25HCPU的总称。
过程CPU	Q02PHCPU、Q06PHCPU、Q12PHCPU、Q25PHCPU的总称。
冗余CPU	Q12PRHCPU、Q25PRHCPU的总称。
QCPU(Q模式)	Q00J、Q00UJ、Q00、Q00U、Q01、Q01U、Q02(H)、Q02PH、Q02U、Q03UD、Q03UDE、Q04UDH、Q04UDEH、Q06H、Q06PH、Q06UDH、Q06UDEH、Q10UDH、Q10UDEH、Q12H、Q12PH、Q12PRH、Q13UDH、Q13UDEH、Q20UDH、Q20UDEH、Q25H、Q25PH、Q25PRH、Q26UDH、Q26UDEHCPU的总称。
LCPU	L02CPU、L26CPU-BT的总称。

A - 14

1 概要

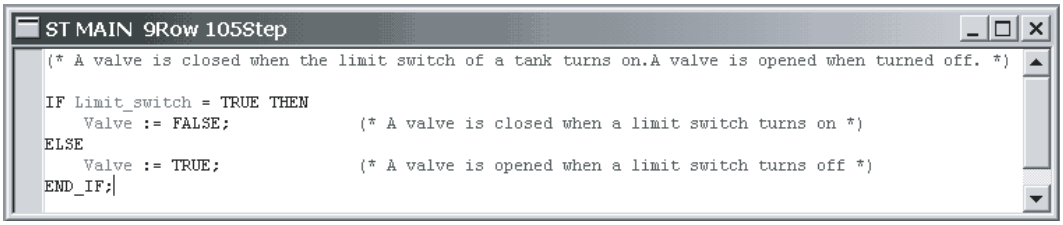
1.1 关于ST语言

ST语言是指，对于打开・控制中的逻辑表示方式所规定的国际标准IEC61131-3中定义的语言。
ST语言中支持运算符・控制语句・函数，可以按以下方式进行记述。

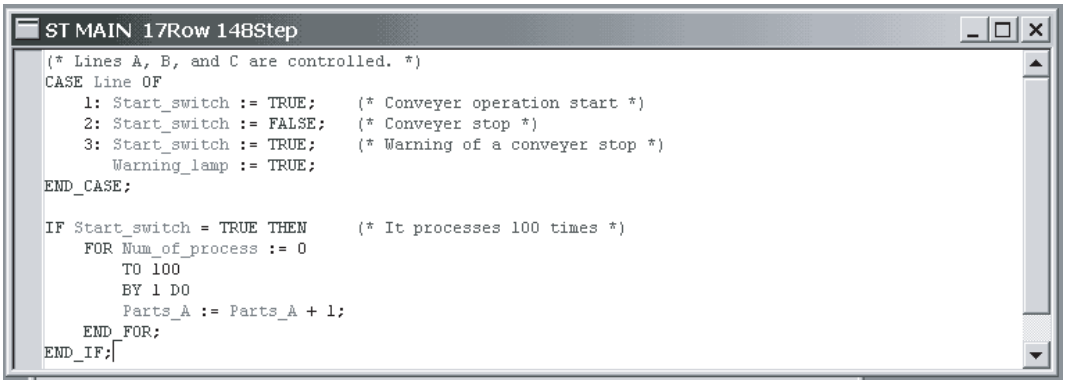
- 通过条件语句进行选择分支，通过循环语句进行重复等的控制语句。
- 使用运算符号(*、/、+、-、<、>、=等)的表达式。
- 用户定义的功能块(FB)的调用。
- 函数的调用(MELSEC函数・IEC函数)。
- 包含汉字等的全角字符的注释记述。

ST语言的主要特点如下所示。

(1) 通过文本方式的自由记述
ST语言是以半角英文数字的文本格式进行记述。在注释及字符串中，也可以使用汉字等的全角字符。



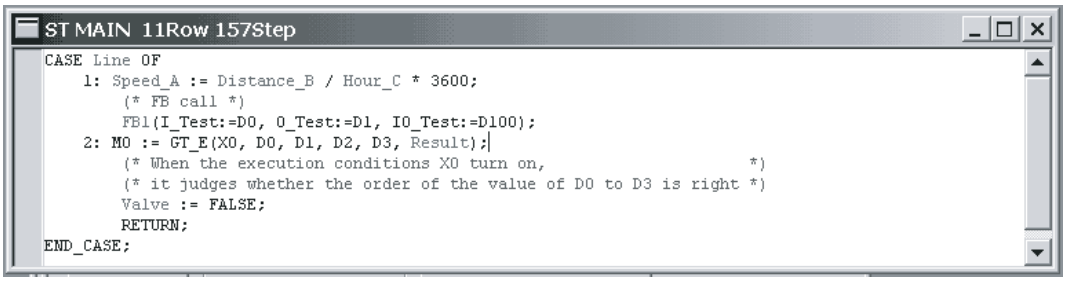
(2) 可以进行与C语言等高级语言相同的编程
ST语言可以与C语言等高级语言一样，通过条件语句进行选择分支，通过循环语句进行重复等的语句对控制进行记述。因此，可以简洁容易地进行程序编写。



1

(3) 可以容易地记述运算处理

ST语言可以对列表及梯形图中难以记述的运算处理简洁而容易地进行记述，因此程序的可读性优良，适用于进行复杂的算术运算・比较运算等的领域。



1.2 MELSEC-Q/L系列中的ST程序的特点

ST程序是通过ST语言进行记述的程序。
在进行ST编程时通过使用GX Developer，可以在优异的操作环境下进行高效编程。
MELSEC-Q/L系列中的ST程序的主要特点如下所示。

(1) 通过部件化可以实现设计高效化
在ST语言中将经常使用的处理作为功能块(FB)进行部件化而预先定义，可以在各程序的必要部分进行调用。由此可使程序开发高效化，同时减少了程序错误，提高了程序质量。
详细内容请参阅关联手册中记载的“GX Developer Version8操作手册(功能块篇)”。

(2) 可以从可编程控制器中读取以恢复ST程序
在MELSEC-Q/L系列的ST程序中，创建的程序被写入到可编程控制器中执行，也可以从可编程控制器中读取后进行恢复，以ST语言格式进行编辑。

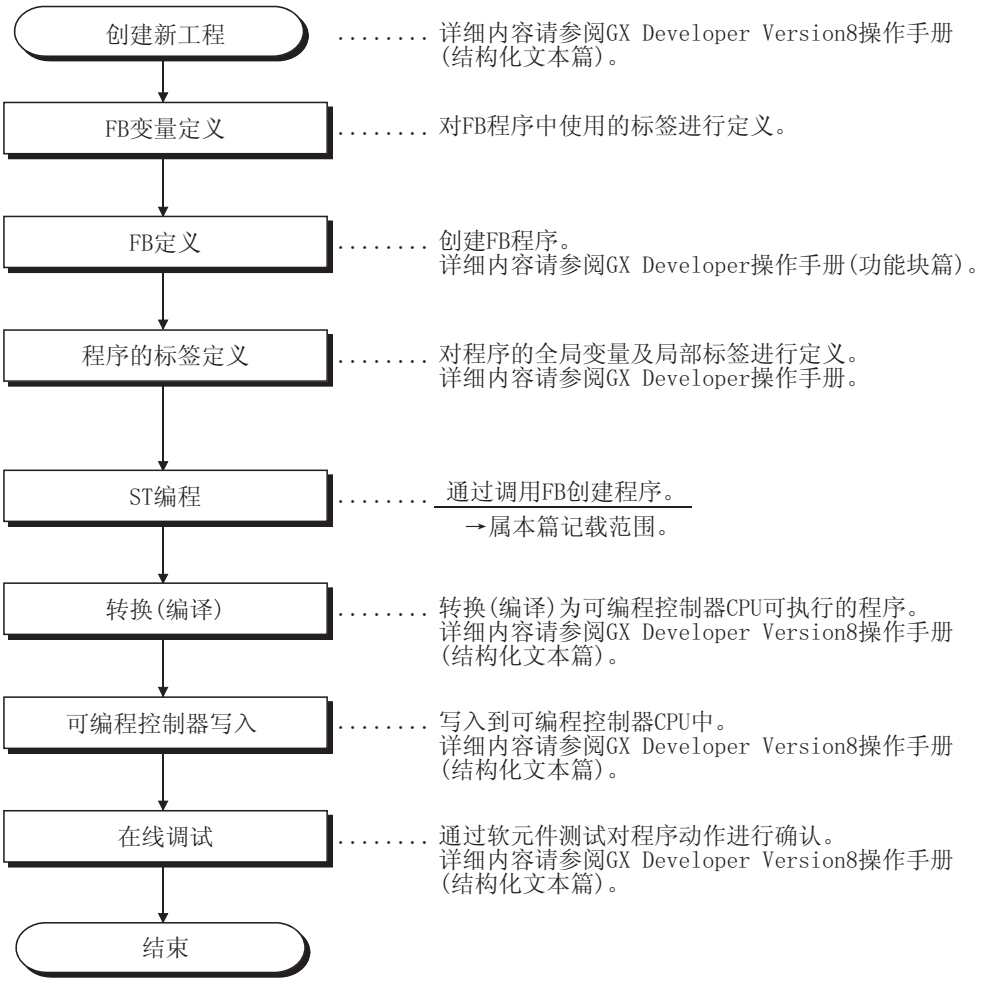
(3) 可以在系统运行过程中进行程序变更(RUN中写入)
可以在不停运系统的状况下对执行中的程序进行部分变更。

(4) 与其它语言程序的联用
MELSEC-Q/L系列也支持除ST语言以外的其它语言，因此可以使用适用于处理的语言进行高效率的程序创建。
各程序可以以文件为单位进行执行条件设置，可以对1个CPU写入多个程序文件。
由于支持多个语言，可以以最佳控制对应于广泛的用途。

(5) 丰富的函数群
在MELSEC-Q/L系列的ST程序中，配备有MELSEC-Q/L系列用的各种公共指令对应的MELSEC函数及IEC61131-3中定义的IEC函数。

1.3 ST程序创建步骤

将ST编程的大致步骤以流程图进行表示。
以下示例是通过功能块进行部件创建，然后通过ST语言创建主程序的示例。



2 系统配置

2.1 系统配置

本节介绍使用ST程序时的系统配置有关内容。

2.1.1 适用CPU

下述CPU模块可支持ST程序。

基本型QCPU	高性能型QCPU	通用型QCPU	过程CPU	冗余CPU	LCPU
Q00JCPU Q00CPU Q01CPU	Q02CPU Q02HCPU Q06HCPU Q12HCPU Q25HCPU	Q00UJCPU Q00UCPU Q01UCPU Q02UCPU Q03UDCPU Q03UDECPU Q04UDHCPU Q04UDEHCPU Q06UDHCPU Q06UDEHCPU Q10UDHCPU Q10UDEHCPU Q13UDHCPU Q13UDEHCPU Q20UDHCPU Q20UDEHCPU Q26UDHCPU Q26UDEHCPU	Q02PHCPU Q06PHCPU Q12PHCPU Q25PHCPU	Q12PRHCPU Q25PRHCPU	L02CPU L26CPU-BT

2.1.2 ST程序用编程工具

ST程序的创建・编辑・监视通过下述编程工具进行。

软件包名	运行环境
GX Developer Version8.00A以后	请参阅“GX Developer Version8操作手册(入门篇)”。

2.1.3 ST程序规格

以下介绍ST的规格以及可以使用的软元件。

(1) 程序容量
每1个程序的文件容量为839680(半角)。

要点

● 对文件内的字符数进行计数时应注意以下事项。
- 回车(CR)、换行(LF)符作为2个字符处理。
- 全角字符作为2个字符处理。
- 半角空格作为1个字符处理。
- TAB制表符作为1个字符处理。

(2) 可用软元件

ST程序中可用软元件名如下所示。软元件点数可通过参数设置进行变更。
软元件的表示方法的详细内容请参阅“3.3.3 软元件”。

分类	类别	软元件	表示
内部用户软元件	位	输入	X
		输出	Y
		内部继电器	M
		锁存继电器	L
		报警器	F
		链接继电器	B
		链接特殊继电器	SB
	字	数据寄存器	D
		链接寄存器	W
		链接特殊寄存器	SW
内部系统软元件	位	特殊继电器	SM
	字	特殊寄存器	SD
链接直接软元件	位	链接输入	Jn\X
		链接输出	Jn\Y
		链接继电器	Jn\B
		链接特殊继电器	Jn\SB
	字	链接寄存器	Jn\W
		链接特殊寄存器	Jn\SW
模块访问软元件	字	智能功能模块软元件	Un\G
变址寄存器	字	变址寄存器	Z*1
文件寄存器	字	文件寄存器	R
			ZR
常数	位/ 字/ 双字	10进制常数	K
		16进制常数	H
	实数	实数常数	E
	字符串	字符串常数	“ABC”等
其它	位	SFC块软元件	BL
	位	SFC转移软元件	BL\TR
	位	SFC步进继电器	BL\S
	位	直接输入	DX
	位	直接输出	DY

*1: Z0、Z1禁止使用。通用型QCPU/LCPU的情况下，Z16～Z19禁止使用。

2 - 2

(3) 只能在ST程序中使用的软元件

在ST程序中,将定时器・计数器的触点・线圈・当前值作为个别软元件表示、使用。
定时器・计数器的触点・线圈・当前值的软元件表示及类别如下所示。

分类	类别	软元件	软元件表示
内部用户软元件	位	定时器触点	TS
		定时器线圈	TC
		累计定时器触点	STS
		累计定时器线圈	STC
		计数器触点	CS
		计数器线圈	CC
	字	定时器当前值	TN/T
		累计定时器当前值	STN/ST
		计数器当前值	CN/C

使用示例

(1) [ST程序]
```
M0:=TS0;
```
[相应列表程序]
```
LD    T0
OUT   M0
```

(2) [ST程序]
```
COUNTER_M(X0, CC20, 10);
```
[相应列表程序]
```
LD    X0
OUT   C20    K10
```

要点

关于可用指令的详细内容，请参阅下述手册。

・MELSEC-Q/L 编程手册(公共指令篇)

备忘录

2 - 4

3 ST程序的字符及数值的处理

3.1 可用字符

ST语言是以文本格式进行记述的编程语言。
可以按与通过一般文本编辑器进行文件编辑相同的方式进行记述，但语法及可用字符以及符号是预先进行了定义的。

(1) 用字符

ST程序中的可用字符如下所示。

字符类别	使用位置				字符示例
	程序文	注释	字符串	标签*1	
英文数字	○	○	○	○	ABC、IF、D0
半角符号 + - * / = < > [ ] ( ) . , _ : ; $ # ″ ’ { }	○	○	△*2	×	(D0 * D1)
半角空格	○	○	○	×	
全角空格	×	○	○	×	
换行符	○	○	×	×	
TAB制表符	○	○	×	×	

○：可以使用　×：不能使用　△：部分不能使用

*1: 关于标签中不能使用的字符，请参阅“附录1 标签・FB名中不能使用的字符串”。

*2: 字符串中不能使用半角双引号(")。
如果使用将会发生转换出错。

(2) 字符的类型

ST程序中使用的字符可按以下进行分类。

分类项目		内容	示例
标签名		用户任意定义的字符串。 包括功能块名・数组名・结构体名等。	Switch_A
常数		直接写入到程序中的值 (整数・实数・字符串等)	123,"abc"
注释		程序中不作为控制处理对象的注释文。	(* ON *)
保留字	数据类型名	表示数据类型的单词。	BOOL,DWORD
	控制语句	作为控制语句使用的在语法上被定义了含义的单词。	IF,CASE,WHILE,RETURN
	软元件名	MELSEC中的可编程控制器用数据名	X,Y,M,ZR
	函数名	定义的MELSEC函数・IEC函数名	OUT_M REAL_TO_STR_E
运算符		用于公式及代入语句的被定义了含义的字符代码。	+ - < > =
分组符号		用于标明程序结构的被定义了含义的字符代码。	; ( )
其它符号		用于数组整齐的代码	半角空格 换行符、TAB

3.2 数据的处理

在ST程序中使用的数据类型是被进行了定义的。
在3.2节、3.3节中，介绍了ST程序中的数据类型及其表示方法。

3.2.1 数据类型

ST程序中可用的数据类型如下所示。

数据类型	内容	范围	梯形图中的类型	C语言中的类型
BOOL	布尔型	TRUE・FALSE, 1・0*1	位	bool
INT	整数型	-32768～32767	字	signed short
DINT	双精度整数型	-2147483648～ 2147483647	双字	signed long
REAL	实数型	-3.402823^{+38}～ -1.175495^{-38}, 0.0, $+1.175495^{-38}$～ $+3.402823^{+38}$	实数	float
STRING	字符串型	最多可以定义50 个字符。	字符串	char
ARRAY	数组数据类型	根据指定要素的 数据类型。	数组	char[ ]等
STRUCT	结构化数据类型	根据指定要素的 数据类型。	结构体	struct

*1: K、H指定的K0・K1・H0・H1不能作为BOOL型处理。

点

● 运算结果超出了数据类型的范围时的注意事项
运算结果超出了数据类型的范围时，无法得出正确的结果。

3.2.2 关于ANY类型

在函数的自变量・返回值等允许多个数据类型的情况下，使用ANY类型。ANY类型是可处理任意数据类型的数据类型，其类型如下表所示。
例如，函数自变量被定义为ANY_NUM的情况下，可以从字型・双字型・实数型指定任意的数据类型作为自变量。

[记述示例]

```
REAL EXPT(REAL In1,ANY_NUM In2);        (*函数EXPT的函数定义*)
                    └▶可以指定单字型・双字型・实数型
```

・指定为单字型软元件的情况下

```
RealLabel := EXPT(E1.0, D0);
```

・指定为双字型标签的情况下

```
RealLabel := EXPT(E1.0, DWLabel);
```

・指定为实数的情况下

```
RealLabel := EXPT(E1.0, E1.0);
```

与ANY类型的类型相对应的数据类型・软元件型如下所示。

ANY类型名	数据类型	BOOL	INT	DINT	REAL	STRING
	梯形图中的类型	位	字	双字	实数	字符串
ANY		○	○	○	○	○
ANY_SIMPLE		○	○	○	○	○
ANY_BIT		○	△	□	－	－
ANY_NUM		－	○	○	○	－
ANY_REAL		－	－	－	○	－
ANY_INT		－	○	○	－	－
ANY16		－	○	－	－	－
ANY32		－	－	○	－	－

○: 可作为对应类型进行指定
－: 不能指定
△: 软元件・常数・位数指定可以使用/标签不能使用
□: 常数・位数指定可以使用

3.2.3 数组及结构体

ST程序中，可以将数组及结构体作为数据使用。
数组及结构体是指，通过在使用之前将各要素定义为局部标签或者全局标签，可以在程序中将其作为一个块进行处理的数据。

(1) 数组

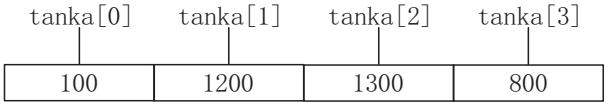
数组是指，将相同类型的多个数据进行组合定义的数据类型。
对于ST程序中的数组，通过在数组类型中定义的变量(标签)名的后面用[ ]指定要素编号，可以对各要素分别进行参照。
数组要素的指定编号从0开始计数。

[格式]

数组名[数组要素的指定编号]

[示意图]

将要素数为4个的字型数组的数组名命名为tanka的情况下
数组要素的指定编号变为0、1、2、3。



字型数组的情况下，在各数组要素中输入字数据。

[记述示例]

```
tanka[0] := 100;        (*在数组第1要素中代入10*)

(*利用软元件D1在数组第2要素中代入1200*)
D1 := 1;
tanka[D1] := 1200;
```

可以通过数据类型INT指定数组要素的指定编号。

```
(*在数组第3要素中代入tanka[0]+tanka[1]*)
tanka[2] := tanka[0] + tanka[1];

pen1 := 3;
tanka[pen1] := 800;
```

可以使用标签作为数组要素的指定编号。

要点

● 使用数组要素的指定编号时的注意事项
要素数为n个的数组的情况下，数组要素的指定编号为0～n-1，因此如果指定为n以上则转换时将发生出错。
例）要素数为4个的数组的情况下
tanka[4] := 100; ←发生出错。

● 数组要素的指定编号中使用数组时的注意事项
数组要素的指定编号中可以使用数组。最多可以使用5个嵌套。使用17个以上使用时将发生转换出错。
例）嵌套为5个的情况下
tanka[tanka[tanka[tanka[tanka[D1]]]]] := 100;

● 设置数组要素的指定编号时的注意事项
有可能损坏其它软元件的信息，因此应注意数组要素编号中指定的值不要超出数组要素数。

● 设置数组的要素数时的注意事项
通过全局(局部)变量设置画面进行登录。可登录的要素数为256个。

(2) 结构体

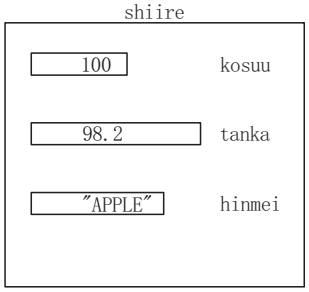
结构体是将任意类型的数据进行组合定义的数据类型。
在结构体型中定义的变量(标签)名的后面，通过将要素名用点号(.)分开记述，可以对各要素进行分别参照。
要素名也被称为成员变量。

[格式]

结构体名.结构体要素名

[示意图]

结构体名 shiire，
结构体要素:
字型1个 结构体要素名 kosuu
实数型1个 结构体要素名 tanka
字符串型1个 结构体要素名 hinmei
的情况下



[记述示例]

```
(*在结构体的要素kosuu中代入100*)
shiire.kosuu := 100;

(*在结构体的要素tanka中代入98.2*)
shiire.tanka := E98.2;

(*在结构体的要素hinmei中代入"APPLE"*)
shiire.hinmei := "APPLE";
```

要点

- 使用结构体的成员变量时的注意事项
  通过结构体变量设置画面可登录的成员数为128个。

3.3 数据的表示方法

ST程序中可用的数据有常数・标签・软元件。

项目	内容	表示示例
常数	直接写入到程序中的数值及字符串数据。 程序执行过程中不变化。	123, "ABC"
标签	用户对类型及名称进行定义的数据。	Switch_A
软元件	QCPU(Q模式)/LCPU中使用的软元件。 通过软元件名・软元件编号识别。	X0,Y0,D100, J1\X0

3.3.1 常数

在ST程序中各常数按以下方式表示。

数据类型	进制数	表示方法	示例
BOOL		TRUE・FALSE 1・0	M0 := TRUE;
	2进制数	在使用的2进制数的前面附加“2#”。	M0 := 2#0; M0 := 2#1;
	8进制数	在使用的8进制数的前面附加“8#”。	M0 := 8#0; M0 := 8#1;
	16进制数	在使用的16进制数的前面附加“16#”。	M0 := 16#0; M0 := 16#1;
INT DINT	2进制数	在使用的2进制数的前面附加“2#”。	D0 := 2#110;
	8进制数	在使用的8进制数的前面附加“8#”。	D0 := 8#377;
	10进制数	对使用的10进制数进行直接输入。 (也可在数值的前面附加“K”)	D0 := 123; D0 := K123;
	16进制数	在使用的16进制数的前面附加 “16#”。 (也可在数值的前面附加“H”)	D0 := 16#FF; D0 := HFF;
REAL		对使用的实数进行直接输入。 (也卡在数值的前面附加“E”)	ABC := 2.34; Rtest := E2.34;
STRING		将字符串用''(或者"")围住。	Stest := 'ABC'; Stest := "ABC";

关于各常数中可指定的范围，请参阅3.2.1项 数据类型。
在3.2.1项 数据类型中未记载的部分将变为以下范围。

[K、H表示]

值的范围	IEC数据类型
K-32768 - K32767	INT,ANY16
K-2147483648 - K2147483647	DINT,ANY32
K0 - K32767	ANY_BIT(字)*1
K0 - K2147483647	ANY_BIT(双字)*2
H0 - HFFFF	INT,ANY16, ANY_BIT(字)*1
H0 - HFFFFFFFF	DINT,ANY32, ANY_BIT(双字)*2

[无K、H表示]

值的范围	IEC数据类型
0 - 1	BOOL
-32768 -32767	INT
-2147483648 - 2147483647	DINT
0 - 4294967295	ANT_BIT(双字)*2
0 - 65535	ANT_BIT(字)*1
-32768 - 65535	ANY16
-2147483648 - 4294967295	ANY32
2#0 - 2#1 8#0 - 8#1 16#0 - 16#1	BOOL
2#0 - 2#1111_1111_1111_1111 8#0 - 8#177777 16#0 - 16#FFFF	INT ANY16 ANY_BIT(字)*1
2#0 - 2#1111_1111_1111_1111_1111_1111_1111_1111 8#0 - 8#37777777777 16#0 - 16#FFFFFFFF	DINT ANY_BIT(双字)*2 ANY32

*1: 作为字软元件处理的情况如下所示。
＜例＞ D0 := NOT(K32767);
*2: 作为双字软元件处理的情况如下所示。
＜例＞ K8M0 := NOT(K2147483647);

3 - 9

要点

- 在字标签、字软元件的运算公式中使用H、2#、8#、16#指定的数值时的注意事项
  运算中处理的值的范围为H8000～HFFFF，对ST程序进行了转换时的运算结果与通过可编程控制器CPU将值代入软元件中时的运算结果有所不同。
  对于进行了ST程序转换时的运算结果，由于不对处理值是单字型还是双字型进行判断，是进行无符号运算，而在可编程控制器CPU中是进行带符号运算。
  ＜使用示例＞
  Data1 = -32768;
  Data2 = 16#8000;
  • ST　Result := Data1 / Data2;　→ -32768÷32768 = -1
  • CPU　Result := Data1 / Data2;　→ -32768÷-32768 = 1

● 在字符串型数据中使用“$”“’”时的注意事项
  “$”被作为换码顺序符使用。
  连续2个“$”的16进制数字将被识别为ASCII码，与ASCII码对应的字符将被插入到字符串中。
  在连续2个“$”的16进制数字不对应ASCII码的情况下，将变为转换出错。
  但是，“$”后面连接了以下字符的情况下将不变为出错状态。

表示	字符串中使用的符号・打印控制码
$$	$
$’	’
$L或$l	换行
$N或$n	改行
$P或$p	换页
$R或$r	返回
$T或$t	TAB

  例)Value := "$'APPLE$' $$100";

● 2进制数・8进制数・10进制数・16进制数・实数表示时的注意事项
  在2进制数・8进制数・10进制数・16进制数・实数表示中，为了易于阅读可以使用“_(下划线)”。“_”不被作为数值而被忽略。
  例) 2#1101_1111　8#377_1　16#01FF_ABCD　22_323　1.0_1
  (K、H、E指定时不能使用“_”。)

3.3.2 标签

在ST程序中不能将数据作为标签使用。
在ST程序中使用标签的情况下，在使用之前应在局部变量设置画面或者全局变量设置画面中进行标签宣言。
(关于标签、结构体标签的宣言方法请参阅“GX Developer Version8操作手册”。)
在ST程序中标签的表示示例如下所示。

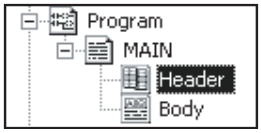
```
例)Switch_A := FALSE;                (*在Switch_A中代入FALSE。                *)

例)IF INT_TO_BOOL(Kosuu) = FALSE THEN
         Daisuu := 2147483647;
   END_IF;
                                     (*如果INT_TO_BOOL(Kosuu)变为FALSE       *)
                                     (*在Daisuu中代入2147483647。            *)

例)Limit_A := E1.0;                  (*在Limit_A中代入1.0。                   *)

例)传送带[4] := Kosuu;               (*在传送带的第5要素中                    *)
                                     (*代入Kosuu的值。                        *)

例)stPressure.Status := TRUE;        (*在stPressure的要素名Status中           *)
                                     (*代入TRUE。                             *)

例)stPressure.eLimit := E1.0;        (*在stPressure的要素名eLimit中           *)
                                     (*代入1.0。                              *)
```

参考

● 进行标签宣言的步骤

标签的宣言是在局部变量设置画面或者全局变量设置画面中进行。
局部变量设置画面可以通过以下操作打开。

启动GX Developer → [打开工程] →



→ 双击Header标签图标 → 局部变量设置画面

在局部变量设置画面中设置了标签的示例如下所示。

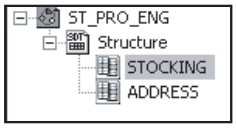
	Au	Label	Constant	Device type
1		Switch_A		BOOL
2		Unit_No		INT
3		Line_No		DINT
4		Limit_A		REAL
5		Conveyer		INT(20)

对结构体标签进行宣言时

1) 对结构体的要素进行宣言。

启动GX Developer → [打开工程] → 点击结构体选项卡 → 新建结构体

→



→ 双击“STOCKING”、“ADDRESS”

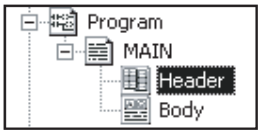
→ 结构体变量设置画面

在结构体变量设置画面中设置了结构体的要素标签的示例如下所示。

	Label	Device type
1	Unit_No	INT
2	Unit_Price	REAL

2) 对结构体标签进行宣言。

结构体标签的宣言是在局部变量设置画面或者全局变量设置画面中进行。局部变量设置画面可以通过以下操作打开。



→ 双击Header标签 → 局部变量设置画面

在局部变量设置画面中设置了结构体标签的示例如下所示。

	Au	Label	Constant	Device type
1		StockingData	Setting detail	STRUCTURE(STOCKI

要点

● 使用指针型・定时器型・计数器型・累计定时器型标签时的注意事项
可以对指针型・定时器型・计数器型・累计定时器型的标签进行宣言，但在ST程序中作为标签使用的情况下，将发生转换出错而无法使用。

● 使用定时器型・计数器型・累计定时器型标签时的注意事项
在结构体的成员变量中，对定时器型・计数器型・累计定时器型的标签进行了定义的情况下，该成员变量不能在ST编辑画面中使用。但是，包含有定时器型・计数器型・累计定时器型的标签的结构体的其它成员变量可以使用。

3 - 13

3.3.3 软元件

(1) 软元件的使用方法

在ST程序中可以不使用标签，直接记述及使用QCPU(Q模式)/LCPU的软元件。软元件可以在表达式的左边・右边，作为函数的自变量・返回值等使用。

[记述示例]

```
M0 := TRUE;                          (*将M0置为ON。                    *)

IF  INT_TO_BOOL(D0)  =  FALSE  (*如果INT_TO_BOOL(D0)变为FALSE   *)
THEN
     W0 := 1000;                     (*在W0中代入1000。                *)
END_IF;
```

备注

- 进行软元件的指定时．．．
  软元件的指定可以使用大写字母及小写字母。
- 可用软元件？
  关于可用软元件，请参阅本书的“2.1.3 (2) 可用软元件”。

(2) 其它使用方法

软元件的修饰方法・指定方法有以下 3 种方法可以使用。将这些方法在梯形图程序中使用时可以以相同的使用方法使用。以下记载了在ST程序中使用时的记述示例及说明。(关于各使用方法的详细内容，请参阅“MELSEC-Q/L 编程手册(公共指令篇)”。)

(a) 变址修饰
(b) 位指定
(c) 位数指定

(a) 变址修饰

变址修饰是使用了变址寄存器的间接地址指定。

如果使用变址寄存器，软元件编号将变为(直接指定的软元件编号)+(变址寄存器的内容)。

[记述示例]

```
(* 根据Z2中输入的数值对成为对象的D软元件编号进行变更。                *)
(* Z2中输入了1的情况下对象软元件编号将变为D(0+1)→D1。                *)
  Z2 := 1;          (*“在变址寄存器Z2中代入1”*)
  D0Z2 := K0;       (*  “在D0Z2中代入K0”      *)
    └▶D1
```

(b) 位指定

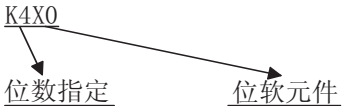
通过对字软元件的各个位No.进行指定，可以作为位软元件使用。

[记述示例]

```
D0.0 = TRUE;      (*将D0软元件的第0位置为ON。   *)
W0.F = FALSE;     (*将W0软元件的第15位置为OFF。*)
```

(c) 位数指定

通过将位软元件的4位、8位、12位．．．作为1位数进行位数指定，在位软元件中可以处理单字数据或者双字数据。



[记述示例]

```
      K4X0 := D0;       (*将从X0软元件开始的16位作为整数型(INT)使用，
                                                    代入D0。*)

     Wtest := K1X0;   (*将从X0软元件开始的4位代入单字型标签Wtest中。*)

   Dwtest := K5X0;  (*将从X0软元件开始的20位代入双字型标签Dwtest中。*)
```

备注

● 使用位数指定时的数据类型. . .
使用位数指定的情况下，变为以下数据类型。
例) 使用了X0的情况下
整数型(INT) : K1X0,K2X0，K3X0,K4X0
双精度整数型(DINT) : K5X0,K6X0，K7X0,K8X0

点

● 使用位数指定时的注意事项1
右边与左边的数据类型不相同的情况下，将变为转换出错。
例) D0 := K5X0;
K5X0为双字，D0为单字型，因此上述程序发生出错。

● 使用位数指定时的注意事项2
右边>左边的情况下，在左边的对象点数范围内进行数据传送。
(关于位数指定的对象点数，请参阅“MELSEC-Q/L编程手册(公共指令篇)”)
例) K5X0 := 2#1011_1101_1111_0111_0011_0001;
K5X0 : 对象点数20点
在K5X0中代入1101_1111_0111_0011_0001(20位)。

4 ST程序的表达式

4.1 代入语句

代入语句具有将右边表达式的结果代入到左边的标签及软元件中的功能。
在代入语句中，右边表达式的结果与左边的数据类型需为相同的类型。如果不相同将发生转换出错。

[记述示例]

・使用了实际软元件的情况下
D0 := 0;
执行了该公式时将10进制数的0代入到D0中。

・使用了标签的情况下
使用了Stest这一字符串型标签的情况下
Stest := ”APPLE”;
执行了该公式时将字符串”APPLE”代入到Stest中。

要点

● 代入字符串时的注意事项
字符串代入的最大字符串长度为32个字符。进行了超出字符串长度32个字符的字符串代入的情况下将发生转换出错。

● 代入语句左边使用软元件时的注意事项
TS・TC・STS・STC・CS・CC・BL・DX・BL\S・BL\TR软元件不能在代入语句的左边使用。如果将上述软元件用于左边将发生转换出错。

4.2 运算符

本节中介绍ST程序中的可用运算符列表及其使用示例。

4.2.1 运算符列表

ST程序中使用的运算符列表及执行运算时的优先顺序如下所示。

运算符	内容	优先顺序
( )	圆括弧式	最高位
函数( )	函数的参数列表	
**	指数(幂)　tei**shisuu	
NOT	布尔补数 (位取反后的值)	
* / MOD	乘法 除法 余数	
+ -	加法 减法	
<,>,<=,>=	比较	
= <>	等式 不等式	
AND,&	逻辑积	
XOR	排他逻辑和	
OR	逻辑和	最低位

优先顺序相同的情况下，通过左侧的运算符进行评判。

关于运算符及对象数据类型以及运算结果数据类型如下表所示。

运算符	对象数据类型	运算结果数据类型
*, /, +, -	ANY_NUM	ANY_NUM
< , >, <=, >=,=, <>	ANY_SIMPLE	BOOL
MOD	ANY_INT	ANY_INT
AND, &, XOR, OR, NOT	ANY_BIT*1	ANY_BIT*1
**	ANY_REAL(底) ANY_NUM(指数)	ANY_REAL

*1: 标签、常数(负的范围)除外。

要点

● 使用运算符时的注意事项1
运算符的右边与左边的数据类型不相同的情况下，将发生转换出错。

● 使用运算符时的注意事项2
运算符应以半角进行记述。

● 使用运算符的注意事项3
1个表达式中可记述的运算符的使用个数最大为1024个。如果使用了1025个以上，将发生转换出错。

备注

● ANY类型的说明. . .
关于ANY类型的说明，请参阅“3.2.2 关于ANY类型”。

4.2.2 运算符的使用示例

以下记载了ST程序中的运算符的使用示例。

(1) 整数型(INT)的运算

(a) 使用了实际软元件的情况下

<使用示例>

```
D0 := D1 * (D2 + K3) / K100;
```

<<运算顺序>>

1) D2＋K3
2) (D2＋K3)*D1
3) (D2＋K3)*D1/K100
4) 将3)的结果代入到D0中

(b) 使用了标签的情况下

・使用了单字型标签Dtest1、Dtest2的情况下

<使用示例>

```
Dtest2 := Dtest1 MOD (D2 + K3) * K100;
```

<<运算顺序>>

1) D2 + K3
2) Dtest1 MOD (D2 + K3)
3) Dtest1 MOD (D2 + K3) * K100
4) 将3)的结果代入到Dtest2中

・使用了双字型标签Dwtest1、Dwtest2的情况下

<使用示例>

```
Dwtest2 := Dwtest1 - Dwtest1 / K100;
```

<<运算顺序>>

1) Dwtest1/K100
2) Dwtest1-Dwtest1/K100
3) 将2)的结果代入到Dwtest2中

要点

● 运算结果超出了数据类型的范围时的注意事项
运算结果超出了数据类型范围的情况下，将无法得出正确的结果。
关于数据类型的范围，请参阅3.2.1项。

(2) 布尔型(BOOL)的运算

(a) 使用了实际软元件的情况下

<使用示例>

```
M0 := X0 AND X1 AND (D1 = 100);
```

<<运算顺序>>

1) 仅在X0 AND X1的结果为ON且D中为100时M0置为ON

(b) 使用了标签的情况下

• 使用了位型标签Btest1、Btest2的情况下

<使用示例>

```
Btest2 := Btest2 OR Btest1;
```

<<运算顺序>>

1) Btest2或Btest1为ON时Btest2置为ON

4 - 5

4.3 控制语句

在ST程序中为了进行比较・重复而配备了条件语句及循环语句。

条件语句: 在某个一定的条件成立时执行选择的语句。
循环语句: 根据某个特定的变量及条件的状态，将1个以上的语句重复执行多次。

4.3.1 控制语句列表

控制语句列表如下所示。

条件语句	IF条件语句
	CASE条件语句
循环语句	FOR・・・DO语句
	WHILE・・・DO语句
	REPEAT・・・UNTIL语句
其它控制语句	RETURN语句
	EXIT语句

点

● 在控制语句使用分级时的注意事项
控制语句的分级最多为16级。即使使用了17级以上也不会发生转换出错，但如果分级过多将可能变为难以理解的程序，因此建议程序的分级级数为4～5级为宜。

4.3.2 条件语句

(1) IF THEN 条件语句

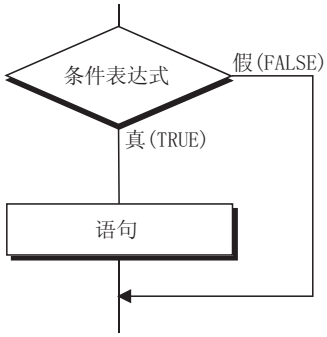
[格式]

```
IF    〈布尔表达式〉 THEN
      〈语句 ・・・〉
END_IF;
```

[说明]

布尔表达式(条件表达式)为真(TRUE)时，执行语句。布尔表达式为假(FALSE)的情况下，不执行语句。

对于布尔表达式，作为单一的位型变量的状态或者包含多个变量的复杂表达式的布尔运算的结果，只要是真(TRUE)或假(FALSE)重复的表达式，则无论什么样的表达式均可使用。



[记述示例]

(a) 布尔表达式中使用了实际软元件的情况下

```
IF X0 THEN              (*如果X0为ON则在D0中代入0。              *)
     D0 := 0;           (*X0的部分X0= TRUE时也变为相同的含义   *)
                        (*。                                    *)
END_IF;
```

(b) 布尔表达式中使用了运算符的情况下

```
IF (D0*D1) <= 200 THEN      (*如果D0*D1为200以下               *)
     D0 := 0;               (*则在D0中代入0。                  *)
END_IF;
```

(c) 布尔表达式中使用了标签的情况下

1) 将标签w_Real指定为实数型的情况下

```
IF w_Real > 2.0 THEN        (*如果w_Real大于2.0                *)
     D0 := 0;               (*则在D0中代入0。                  *)
END_IF;
```

2) 将标签w_Str指定为字符串型的情况下

```
IF w_Str = ”ABC” THEN        (*如果w_Str为”ABC”          *)
     D0 := 0;                (*在D0中代入0。             *)
END_IF;
```

3) 将标签w_Str指定为字符串型的情况下

```
IF w_Str = 'ABC' THEN        (*如果w_Str为'ABC'          *)
     D0 := 0;                (*则在D0中代入0。           *)
END_IF;
```

(d) 布尔表达式中使用了功能块的情况下

功能块名w_FB被设置到局部变量设置中，功能块的输出变量被设置为单字型标签w_Out的情况下

执行功能块后

(关于功能块的使用方法请参阅“GX Developer Version8操作手册”。)

```
IF w_FB.w_Out = 100 THEN     (*如果w_Out为100            *)
     D0 := 0;                (*则在D0中代入0。           *)
END_IF;
```

(e) 布尔表达式中使用了函数的情况下

```
IF INT_TO_BOOL( D0 ) = FALSE THEN
     D0 := 0;                (*如果INT_TO_BOOL(D0)为FALSE *)
END_IF;                      (*则在D0中代入0。           *)
```

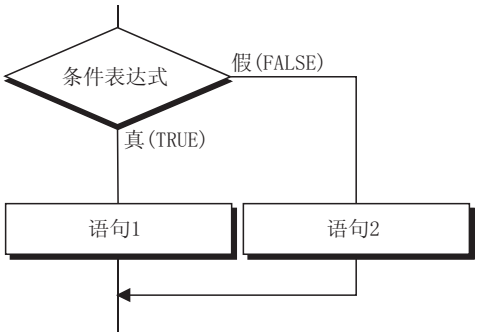
(2) IF…ELSE条件语句

[格式]

```
IF 〈布尔表达式〉 THEN
      〈语句1・・・〉
ELSE
      〈语句2・・・〉
END_IF;
```

[说明]

布尔表达式(条件表达式)为真(TRUE)时，执行语句1。
布尔表达式的值为假(FALSE)的情况下执行语句2。



[记述示例]

(a) 布尔表达式使用了实际软元件的情况下

```
IF X0 THEN              (*X0的部分X0= TRUE时也变为相同的含义。 *)
                        (*。                                   *)
      D0 := 0;          (*如果X0为ON则在D0中代入0。            *)
ELSE                    (*如果X0为ON则在D0中代入1。            *)
      D0 := 1;
END_IF;
```

(b) 布尔表达式中使用了运算符的情况下

```
IF (D0*D1) <= 200 THEN      (*如果D0*D1为200以下               *)
      D0 := 0;              (*则在D0中代入0。                  *)
ELSE                        (*如果D0*D1为200以下               *)
      D0 := 1;              (*则在D0中代入1。                  *)
END_IF;
```

(c) 布尔表达式中使用了函数的情况下

```
IF INT_TO_BOOL(D0) = FALSE THEN  (*如果INT_TO_BOOL(D0)为FALSE  *)
      D0 := 0;                   (*则在D0中代入0。             *)
ELSE                             (*如果INT_TO_BOOL(D0)为FALSE  *)
      D0 := 1;                   (*则在D0中代入1。             *)
END_IF;
```


4 - 9

(3) IF … ELSIF 条件语句

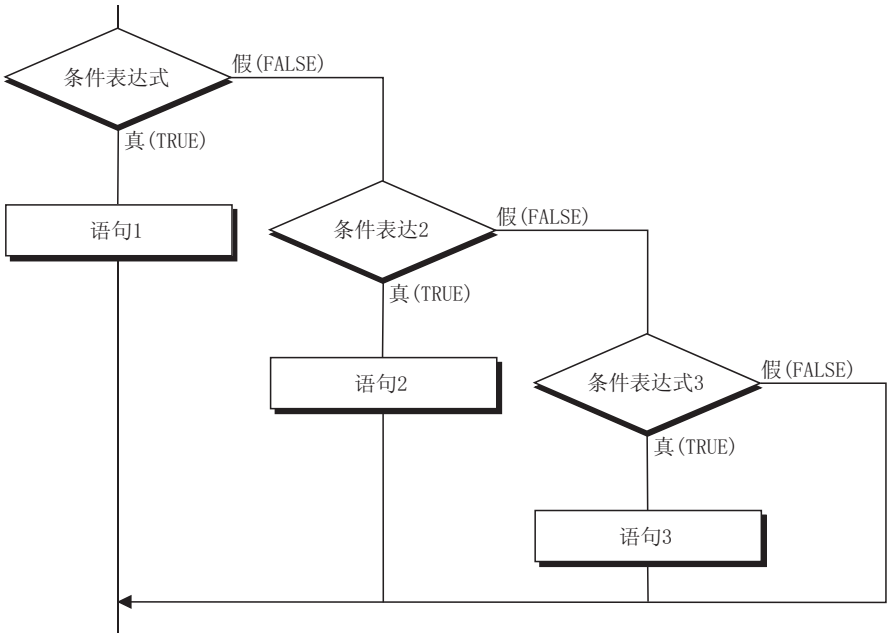
[格式]

```
IF〈布尔表达式1〉THEN
      〈语句1・・・〉
ELSIF〈布尔表达式2〉THEN
      〈语句2・・・〉
ELSIF〈布尔表达式3〉THEN
      〈语句3・・・〉
END_IF;
```

[说明]

布尔表达式(条件表达式)1为真(TRUE)时，执行语句1。布尔表达式1的值为假(FALSE)时且布尔表达式2的值为真(TRUE)的情况下执行语句2。

布尔表达式2的值为假(FALSE)且布尔表达式3的值为真(TRUE)的情况下执行语句3。



[记述示例]

(a) 布尔表达式中使用了实际软元件的情况下

```
IF D0 < 100 THEN              (*如果D0小于100                *)
     D1 := 0;                 (*则在D1中代入0。              *)
ELSIF D0 <= 200 THEN          (*如果D0为200以下              *)
     D1 := 1;                 (*则在D1中代入1。              *)
ELSIF D0 <= 300 THEN          (*如果D0为300以下              *)
     D1 := 2;                 (*则在D1中代入2。              *)
END_IF;
```

(b) 布尔表达式中使用了运算符的情况下

```
IF (D0*D1) < 100 THEN         (*如果D0*D1小于100             *)
     D1 := 0;                 (*则在D1中代入0。              *)
ELSIF (D0*D1) <= 200 THEN     (*如果D0*D1为200以下           *)
     D1 := 1;                 (*则在D1中代入1。              *)
ELSIF (D0*D1) <= 300 THEN     (*如果D0*D1为300以下           *)
     D1 := 2;                 (*则在D1中代入2。              *)
END_IF;
```

(c) 布尔表达式中使用了函数的情况下

```
IF INT_TO_BOOL(D0) = TRUE THEN        (*如果INT_TO_BOOL(D0)   *)
                                      (*为TRUE                *)
     D1 := 0;                         (*则在D1中代入0。       *)
ELSIF INT_TO_BOOL(D2) = TRUE THEN     (*如果INT_TO_BOOL(D2)   *)
                                      (*为TRUE                *)
     D1 := 1;                         (*则在D1中代入1。       *)
END_IF;
```

(4) CASE 条件语句

[格式]

```
CASE <整数表达式> OF
      <整数选择值1>:<语句1 ・・・>
      <整数选择值2>:<语句2 ・・・>
            ・
            ・
            ・
      <整数选择值n>:<语句n ・・・>
ELSE
      <语句n+1 ・・・>
END_CASE;
```

● CASE条件语句的<整数选择值*>中可用的指定方法
关于CASE条件语句中的<整数选择值*>，也可以按以下方式进行1个、多个或者范围指定。
例）

```
1:          (*整数表达式的值为1的情况下          *)
2,3,4:      (*整数表达式的值为2、3、4的情况下    *)
5..10:      (*整数表达式的值为5至10的情况下      *)
```

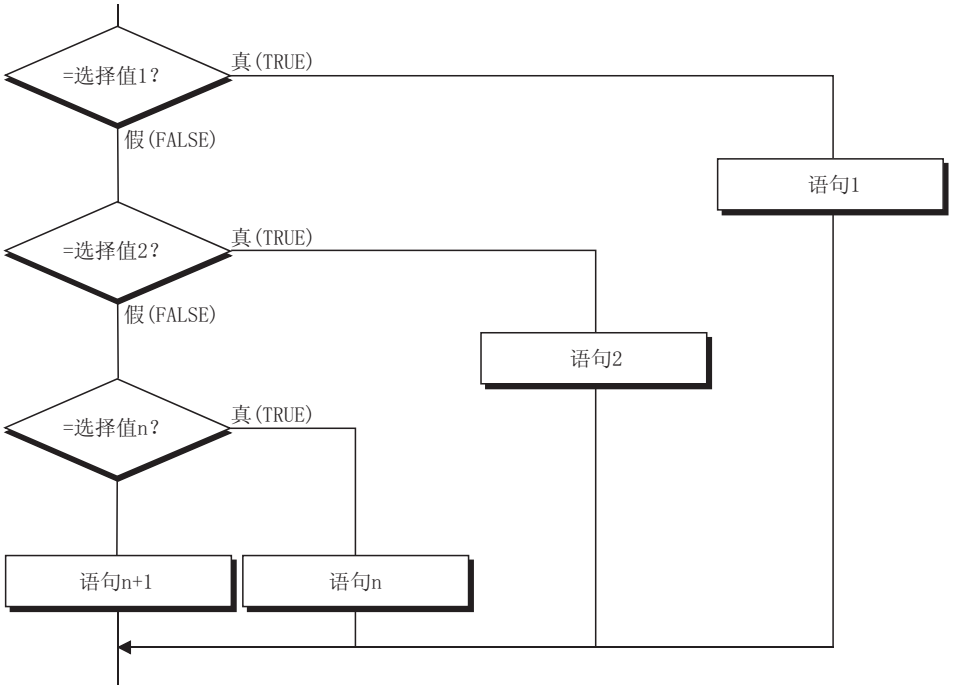
使用“..”进行范围指定时，应使“..”后面的值大于“..”前面的值。
或者也可以将多个以及范围指定进行组合指定。

```
1,2..5,9   (*整数表达式的值为1、2..5、9的情况下 *)
```

● CASE条件语句的<整数表达式>中可用的数据类型
可作为CASE条件语句中的<整数表达式>指定的数据类型为整数型(INT)、双精度整数型(DINT)。可以指定字软元件、单字型・双字型标签。

[说明]

CASE条件语句表达式的结果以整数值返回。例如，该条件语句根据单一的整数值及复杂表达式的结果的整数值，可以在执行选择语句的情况下使用。
具有与整数表达式的值一致的整数选择值的语句将首先被执行，没有一致的值的情况下，执行ELSE后面的语句。



[记述示例]

(a) 整数表达式中使用了实际软元件的情况下

```
CASE D0 OF
   1:
      D1 := 0;      (*如果D0为1则在D1中代入0。                *)
   2,3:
      D1 := 1;      (*如果D0为2、3则在D1中代入1。              *)
   4..6:
      D1 := 2;      (*如果D0为4～6则在D1中代入2。              *)
   ELSE
      D1 := 3;      (*如果D0为除上述以外则在D1中代入3。        *)
END_CASE;
```

(b) 整数表达式中使用了运算结果的情况下

```
CASE D0*D1 OF
   1:
      D1 := 0;      (*如果D0*D1为1则在D1中代入0。              *)
   2,3:
      D1 := 1;      (*如果D0*D1为2、3则在D1中代入0。           *)
   4..6:
      D1 := 2;      (*如果D0*D1为4～6则在D1中代入2。           *)
   ELSE
      D1 := 3;      (*如果D0*D1为除上述以外则在D1中代入3。     *)
END_CASE;
```

(c) 整数表达式中使用了函数的情况下

```
CASE DINT_TO_INT(dData) OF
   1:              (*如果DINT_TO_INT(dData)为1              *)
      D1 := 0;     (*则在D1中代入0。                        *)
   2,3:            (*如果DINT_TO_INT(dData)为2、3           *)
      D1 := 1;     (*则在D1中代入1。                        *)
   4..6:           (*如果DINT_TO_INT(dData)为4～6           *)
      D1 := 2;     (*则在D1中代入2。                        *)
   ELSE            (*如果DINT_TO_INT(dData)为除上述以外     *)
      D1 := 3;     (*则在D1中代入3。                        *)
END_CASE;
```

要点

● 使用整数选择值时的注意事项

CASE条件语句中存在多个相同的整数选择值的情况下，将优先执行上行中记述的语句，后面记述的具有相同整数选择值的语句将不执行。例如，以下的CASE条件语句中,D100的值为3的情况下,具有整数选择值3的语句3将被执行，具有相同整数选择值的语句4将不被执行。

```
CASE D100 OF
   1:      〈语句1 ・・・〉
   2:      〈语句2 ・・・〉
   3:      〈语句3 ・・・〉
   3,4:    〈语句4 ・・・〉
ELSE
           〈语句5 ・・・〉
END_CASE;
```

〈整数选择值*〉的指定中，只能指定无K指定的10进制数。

4.3.3 循环语句

(1) FOR…DO 语句

[格式]

```
FOR <循环变量初始化>
    TO <最终值的表达式>
    BY <增加表达式> DO
    <语句・・・>
END_FOR;
```

循环变量初始化: 对作为循环变量使用的数据进行初始化。
最终值的表达式・增加表达式: 对初始化后的循环变量根据增加表达式进行加法或者减法，反复执行处理直至达到最终值为止。

● FOR语句的<最终值的表达式・增加表达式>中的可用数据类型
可以指定整数值及运算公式的结果的整数值。

[说明]

FOR・・・DO语句根据循环变量的值重复执行若干个语句。

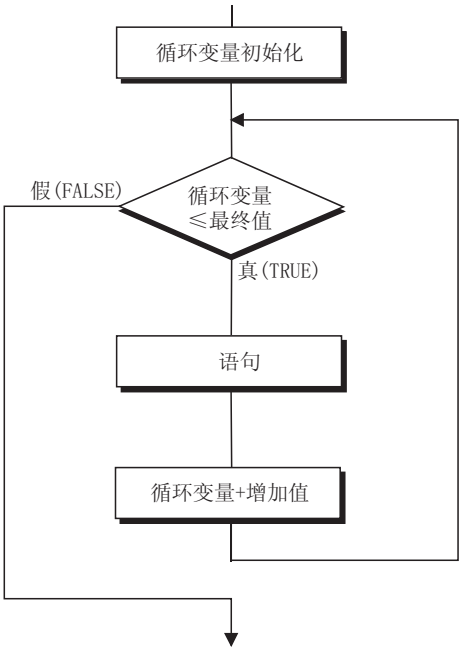
要点

● 使用循环变量时的注意事项
循环变量中，可以使用双精度整数型(DINT)・整数型(INT)，但不能使用结构体要素以及数组要素。
此外，循环变量中使用的类型应与<最终值的表达式>・<增加表达式>的类型一致。

● 使用增加表达式时的注意事项
<增加表达式>可以省略。省略的情况下将<增加表达式>作为1加以执行。此外，如果<增加表达式>中代入了“0”，FOR语句以下有可能不被执行，有可能变为无限循环。

● 使用FOR・・・DO语句时的注意事项
在FOR・・・DO语句中，FOR语句中的<语句・・・>执行后进行循环变量的计数处理。计数处理的执行超出了循环变量的数据类型的最大值或低于最小值的情况下，将发生无限循环。

4 - 15



[记述示例]

(a) 循环变量使用了实际软元件的情况下

```
FOR W1 := 0              (*将W1初始化为0。                *)
    TO 100               (*反复处理直至W1为100。          *)
    BY 1 DO              (*W1每次增加1。                  *)
    W3 := W3 + 1;        (*反复处理期间对W3进行+1。       *)
END_FOR;
```


4 - 16

(2) WHILE…DO 语句

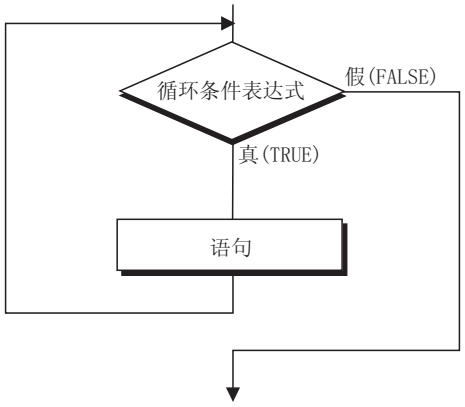
[格式]

```
WHILE <布尔表达式> DO
      <语句・・・>
END_WHILE;
```

[说明]

WHILE・・・DO语句在布尔表达式(条件表达式)为真(TRUE)的期间，执行1个以上的语句。

对于布尔表达式，在语句执行之前判定布尔表达式为假(FALSE)的情况下，WHILE・・・DO内的语句将不被执行。对于WHILE语句中的<布尔表达式>，无论返回的结果为真还是为假均可以，因此IF条件语句中的<布尔表达式>中可指定的表达式均可以使用。



[记述示例]

(a) 布尔表达式中使用了实际软元件・运算符的情况下

```
WHILE W100 < (W2-100) DO      (*W100<(W2-100)为真期间          *)
                              (*反复执行处理。                 *)
      W100 := W100 + 1;       (*反复处理期间对W100进行+1。     *)
END_WHILE;
```

(b) 布尔表达式中使用了函数的情况下

```
WHILE BOOL_TO_DINT(M0) < BOOL_TO_DINT(M1) DO
      D4 := D4 + 1;           (* BOOL_TO_DINT(M0) <            *)
                              (* BOOL_TO_DINT(M1)为真期间      *)
                              (*反复执行处理。                 *)
END_WHILE;                    (*反复处理期间对D4进行+1。       *)
```


4 - 17

(3) REPEAT・・・UNTIL语句

[格式]

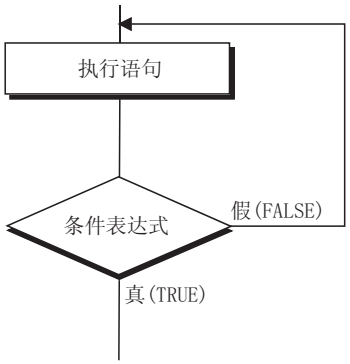
```
REPEAT
      〈语句・・・〉
      UNTIL 〈布尔表达式〉
END_REPEAT;
```

[说明]

对于REPEAT・・・UNTIL语句，在布尔表达式(条件表达式)为假(FALSE)期间，执行1个以上的语句。

对于布尔表达式，在语句执行后进行判定，值为真(TRUE)的情况下，REPEAT・・・UNTIL内的语句不被执行。

对于REPEAT语句中的<布尔表达式>，无论返回的结果为真还是为假均可以，因此IF条件语句中的<布尔表达式>中可指定的表达式均可以使用。



[记述示例]

(a) 布尔表达式中使用了实际软元件的情况下

```
REPEAT
      D1 := D1 + 1;          (*在D1的值小于100之前          *)
      UNTIL D1 < 100         (*对D1进行+1。                 *)
END_REPEAT;
```

(b) 布尔表达式中使用了运算符的情况下

```
REPEAT
      W1 := W0*W1 - D0;      (*在W0*W1的值小于100之前       *)
                             (*将W0*W1-D0                   *)
      UNTIL W0*W1 < 100      (*代入到W1中。                 *)
END_REPEAT;
```


4 - 18

(c) 布尔表达式中使用了函数的情况下

```
REPEAT
      D1 := D1 + 1;                    (*在BOOL_TO_DINT(M0)的值 *)
      UNTIL  BOOL_TO_DINT(M0) < 100    (*小于100之前            *)
END_REPEAT;                            (*对D1进行+1。            *)
```

点

- 使用循环语句时的注意事项1
  循环语句的情况下，应注意避免变为无限循环处理。
- 使用循环语句时的注意事项2
  使用较多的循环语句时，可编程控制器的扫描时间将显著增加，应加以注意。

4 - 19

4.3.4 其它控制语句

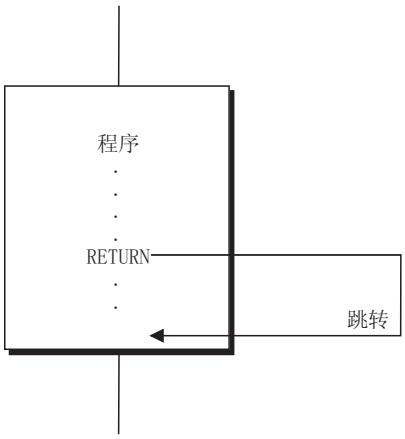
(1) RETURN语句

[格式]

```
RETURN;
```

[说明]

RETURN语句用于中途结束功能块内的程序・ST程序。
如果在程序中使用RETURN语句，RETURN语句以后的处理将全部被忽略，从RETURN被执行之处跳转到ST程序・功能块内的程序的最终行。



[记述示例]

(a) IF条件语句布尔表达式中使用了实际软元件的情况下

```
IF   X0 THEN            (*如果X0变为ON则执行IF内的语句。      *)
     RETURN;            (*RETURN行以后的程序被忽略            *)
END_IF;
```


4 - 20

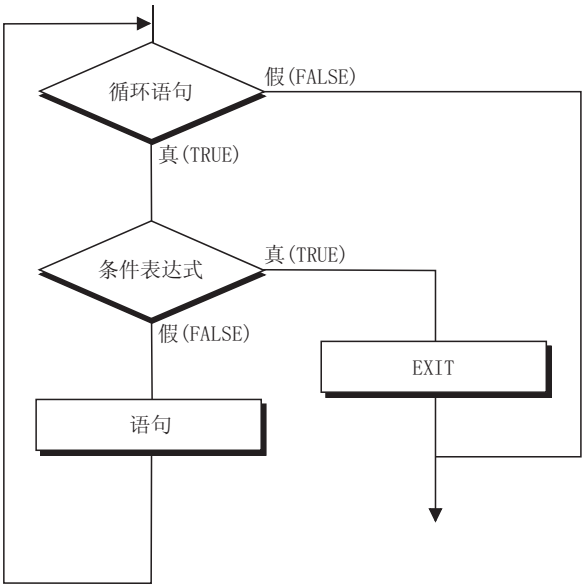
(2) EXIT语句

[格式]

```
EXIT;
```

[说明]

EXIT语句是只能在ST程序的循环语句中使用的语句，使循环回路中途结束。
循环回路的执行过程中如果到达EXIT语句，则EXIT语句以后的循环回路处理将不被执行。从循环回路处理结束的下一行开始继续执行程序。



[记述示例]

(a) IF条件语句布尔表达式中使用了实际软元件的情况下

```
FOR D0 := 0 TO 10 D0      (*D0的值为10以下时重复                *)
                          (*执行。                              *)
   IF D1 > 10 THEN        (*对D1的值是否为10以上进行检查        *)
      EXIT;               (*D1的值为10以上的情况下结束循环处理  *)
   END_IF;
END_FOR;
```


4 - 21

4.3.5 使用控制语句时的注意事项

以下介绍在ST程序中使用了控制语句时的使用步数・运算处理时间、注意事项。

(1) 控制语句使用步数及运算处理时间

以下介绍使用控制语句时的使用步数、运算处理时间。
运算处理时间是通过对各指令的处理时间进行相加来计算。应作为程序创建时的参考。

(a) IF条件语句

IF条件语句1 单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	IF X0 THEN D0 := 100; END_IF;	7	1.534	10.9
列表程序	LD X0 MOV K100 D0	3	0.134	0.90
[备注] 如果在条件语句部分不使用ST，则处理时间将变短。 但是，由于ST的IF条件语句的比较对象为布尔表达式，因此可以使复杂的比较变得容易。				

IF条件语句2 单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	IF D0 = 0 THEN D0 := 100; END_IF;	9	1.6	11.5
列表程序	LD= D0 K0 MOV K100 D0	5	0.20	1.50
[备注] 如果在条件语句部分不使用ST，则处理时间将变短。 但是，由于ST的IF条件语句的比较对象为布尔表达式，因此可以使复杂的比较变得容易。				

(b) CASE条件语句

单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	CASE D0 OF 1,2: D0 := 100; 3..10: D1 := D1 + 1; END_CASE;	29	5.004	36.1
列表程序	LD= D0 K1 AND= D0 K2 MOV K100 D0 LD>= D0 K3 AND<= D0 K10 INC D1	16	0.64	4.6
[备注] 在列表程序中由于无需象ST那样执行CJ、JMP等，因此仅对比较部分进行测定。 比较部分被作为导通状态进行时间计算。				

(c) FOR…DO语句

单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	FOR D0 := 0 TO 10 BY 1 DO D1 := D1 + 1; END_FOR;	16	初始化：0.134 循环：3.308 此时循环部分执行10次动作。	初始化：0.9 循环：24.0
列表程序	FOR K10 LD SM400 INC D1 NEXT	6	2.574	21.6
[备注] 上述运算处理时间取决于循环次数。 在列表程序中只能进行循环次数指定。在ST中根据条件比较可以进行循环等的运算处理。				

4 - 23

(d) WHILE…DO语句

WHILE…DO语句 1　　单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	WHILE X0 DO D0 := 100; END_WHILE;	10	3.034 在X0变为TRUE之前反复执行。	21.9
列表程序	同上	同上	同上	同上
[备注] 在列表程序中即使进行记述，也将变为与ST程序转换结果相同的程序，因此处理时间也与ST相同。				

WHILE…DO语句 2　　单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	WHILE D0= 100 DO D0 := 100; END_WHILE;	15	3.1	22.5
列表程序	同上	同上	同上	同上
[备注] 在列表程序中即使进行记述，也将变为与ST程序转换结果相同的程序，因此处理时间也与ST相同。				

(e) REPEAT…UNTIL语句

REPEAT…UNTIL语句 1　　单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	REPEAT D0 := 100; UNTIL X0 END_REPEAT;	6	1.534 在X0变为TRUE之前反复执行。	10.9
列表程序	同上	同上	同上	同上
[备注] 在列表程序中即使进行记述，也将变为与ST程序转换结果相同的程序，因此处理时间也与ST相同。				

REPEAT…UNTIL语句 2　　单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	REPEAT D0 := 100; UNTIL D0= 0 END_REPEAT;	9	1.6 在D0变为0之前反复执行。	11.5
列表程序	同上	同上	同上	同上
[备注] 在列表程序中即使进行记述，也将变为与ST程序转换结果相同的程序，因此处理时间也与ST相同。				

(f) EXIT语句

单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	－	3	1.4	11
列表程序	同上	同上	同上	同上

[备注]
根据JMP指令移动至反复处理结束之后的指针处。
在列表程序中也与ST程序的动作相同，因此处理时间也与ST相同。

(g) RETURN语句

单位(μs)

		步数	运算处理时间 (Q25H)	运算处理时间 (Q00J)
ST程序	－	3	1.4	11
列表程序	同上	同上	同上	同上

[备注]
根据JMP指令移动至反复处理结束之后的指针处。
在列表程序中也与ST程序的动作相同，因此处理时间也与ST相同。

4 - 26

(2) 使用位软元件时的注意事项

以下介绍在ST程序中使用IF・CASE条件语句创建程序的情况下，通过梯形图程序创建执行相同动作的程序时的注意事项。
在IF条件语句中布尔表达式(条件表达式)一旦成立，在IF条件语句内将位软元件置为ON的情况下，该位软元件将变为常时ON状态。

[ST程序示例1]

```
IF M0 THEN
      Y0 := TRUE;
END_IF;
```

上述程序等同于以下程序:

```
LD M0;
SET Y0;
```

为了避免变为常时ON状态，应对程序按下述方式进行变更。

[ST程序示例2]

```
IF M0 THEN
      Y0 := TRUE;
ELSE
      Y0 := FALSE;
END_IF;
```

上述程序等同于下述程序:

```
(a) LD M0;
    OUT Y0;

(b) Y0 := M0;

(c) OUT_M (M0,Y0);
```

但是，在IF条件语句的语句内使用了OUT_M()的情况下，将变为与[ST程序示例1]相同的状态。

在使用CASE条件语句时也同样需要注意上述事项。
在CASE条件语句中整数表达式(条件表达式)一旦成立，在CASE条件语句内将位软元件置为ON的情况下，该位软元件将变为常时ON状态。

(3) 使用定时器、计数器时的注意事项

以下介绍在ST程序中使用IF・CASE条件语句创建程序的情况下，通过梯形图程序创建执行相同动作的程序时的注意事项。
在IF条件语句中布尔表达式(条件表达式)与定时器・计数器指令的执行条件有所不同。

例)定时器的情况下
[ST程序示例1]

```
IF M0 THEN
    TIMER_M (M1,TC0,K10);
END_IF;
(*M0=ON且M1=ON时，开始计数。                                    *)
(*M0=ON且M1=OFF时，清除计数。                                   *)
(*M0=OFF且M1=ON时，停止计数。计数值不被清除。                   *)
(*M0=OFF且M1=OFF时，停止计数。计数值不被清除。                  *)
```

例)计数器的情况下
[ST程序示例2]

```
IF M0 THEN
    COUNTER_M (M1,CC0,K10);
END_IF;
(*M0=ON且M1=ON/OFF时，对计数进行+1。                           *)
(*M0=OFF且M1=ON/OFF时，不进行计数。                            *)
(*M0=ON/OFF与计数+1不联动。                                     *)
```

上述现象的发生是由于在IF条件语句不成立的情况下，不执行与定时器・计数器相关的语句的缘故。
根据M0的条件及M1的AND条件使定时器・计数器动作的情况下应不使用控制语句，而仅使用MELSEC函数。

[变更ST程序示例]

・使用定时器时　　TIMER_M(M0 & M1,TC0,K10);
・使用计数器时　　COUNTER_M(M0 & M1,CC0,K10);

通过使用变更后的程序可以根据M0及M1的AND条件使定时器・计数器动作。

在使用CASE条件语句时也同样需要注意上述事项。
在CASE条件语句中整数表达式(条件表达式)与定时器・计数器指令的执行条件有所不同。

4.4 功能块的调用

在ST程序中可以使用功能块(FB)。
以下介绍在ST程序中，使用用户创建的FB的方法有关内容。(关于FB的创建方法，请参阅“GX Developer Version8 操作手册(功能块篇)”。)

(1) 功能块的调用
将创建的FB用于ST程序中的情况下，首先需要在局部变量设置画面中进行FB名的定义。(请参阅 参考 。)
在ST程序中，通过对进行了FB名定义的FB名进行记述(FB调用)，可以使用FB。

FB调用时输入变量、输入输出变量均应进行记述。此外，必须对输入变量以及输入输出变量进行值的指定。
对于输出变量，如果不需要输出变量的结果，则可以省略记述。

[记述示例]

FB数据名	: LINE1_FB
输入变量	: I_Test
输出变量	: O_Test
输入输出变量	: IO_Test
FB标签名	: FB1

以下为创建了上述FB情况下FB调用的记述示例。

```
FB1( I_Test := D0,O_Test := D1,IO_Test := D100);
                        ↓
                  输出变量的记述可以省略。
```

(2) 输出结果的获取方法
通过在FB名的后面附加“.”对输出变量名进行指定，可以获取FB的输出。

[记述示例]
将输出变量的结果代入到D1时的记述方法如下所示。

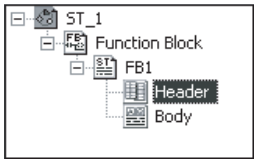
```
D1 := FB1.O_Test;
```


4 - 29

参考

● 对FB的输入变量・输入输出变量・输出变量的标签进行宣言时. . .

启动GX Developer → [打开工程] → 双击FB选项卡 → 新建FB

→ 

→ 双击Header图标

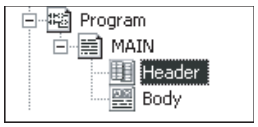
→ FB标签设置画面

通过FB标签设置画面对FB的输入・输出变量标签进行设置的示例如下所示。

	Input/Output	Label	Constant	Device type
1	VAR_INPUT	I_TEST		INT
2	VAR_OUTPUT	O_TEST		INT
3	VAR_IN_OUT	IO_TEST		INT

● 对FB数据名的标签进行宣言时. . .

在进行FB调用前，需要对所使用的FB的标签进行宣言。



→ 双击Header图标

→ 局部(或者全局)变量设置画面

	Au	Label	Constant	Device type
1		BLabel		BOOL
2		DwLabel		DINT
3		label2	Setting detail	FB(FB1)
4				

COUNTER
STOREDTIMER
POINTER
STRUCTURE
FB
FB(FB1)

↓
在软元件类别中选择FB

在局部变量设置画面中对FB标签进行了定义的示例如下所示。

3		label2	Setting detail	FB(FB1)

4 - 30

要点

● 对FB的输出进行获取时的注意事项

FB的输出获取应在FB调用后进行。如果在FB调用前执行则将变为出错状态。

例）FB名 : FB1

输入变量 : I_Test

输出变量 : O_Test

```
D1 := FB1.O_Test;                    (*FB的输出获取 *)
FB1(I_Test := D0,O_Test := D1);      (*FB调用       *)
```

由于是按FB的输出获取、FB调用的顺序，因此变为出错状态。

● 使用输入输出变量时的注意事项

如果象输出变量那样使用输入输出变量的结果将会变为出错状态。

在FB调用时应将输入输出变量的值指定为与输入变量相同。

例）FB名: FB1

输入输出变量 : IO_Test

输出变量 : O_Test

[记述示例]

```
FB1( IO_Test := D1);
D1 := FB1.IO_Test; → 变为出错状态。
```

● 进行FB调用时的注意事项

在ST程序中，通过局部变量设置画面设置的FB只能使用1次。(如果多次使用则会发生出错。)如果要多次使用同一个FB，应事先在局部变量设置画面中按使用次数对FB进行宣言。

例）在局部变量设置画面中对FB标签进行了多次定义的示例如下所示。

	Au	Label	Constant	Device type
1		label	Setting detail	FB(FB1)
2		label1	Setting detail	FB(FB1)
3		label2	Setting detail	FB(FB1)

在程序中按下述方式使用。

```
label(I_Test := D0,IO_Test := D100);
label1(I_Test := D1,IO_Test := D150);
label2(I_Test := D3,IO_Test := D200);
```


4 - 31

4.5 注释

在ST程序中，可以输入注释。用“(*”及“*)”围住的部分将被作为注释处理。 如果在注释中输入注释将会发生出错。

[正常示例]

例1) (*使泵处于待机状态。 *)

例2) (**********)

例3) (*输入开关后使马达运行 *)

例4) (* Flag_A = TRUE 控制开始(* Flag_B = TRUE 制御停止 *)

[出错示例]

例5) (* Flag_A = TRUE 控制开始 *) Flag_A = FALSE 控制停止 *)

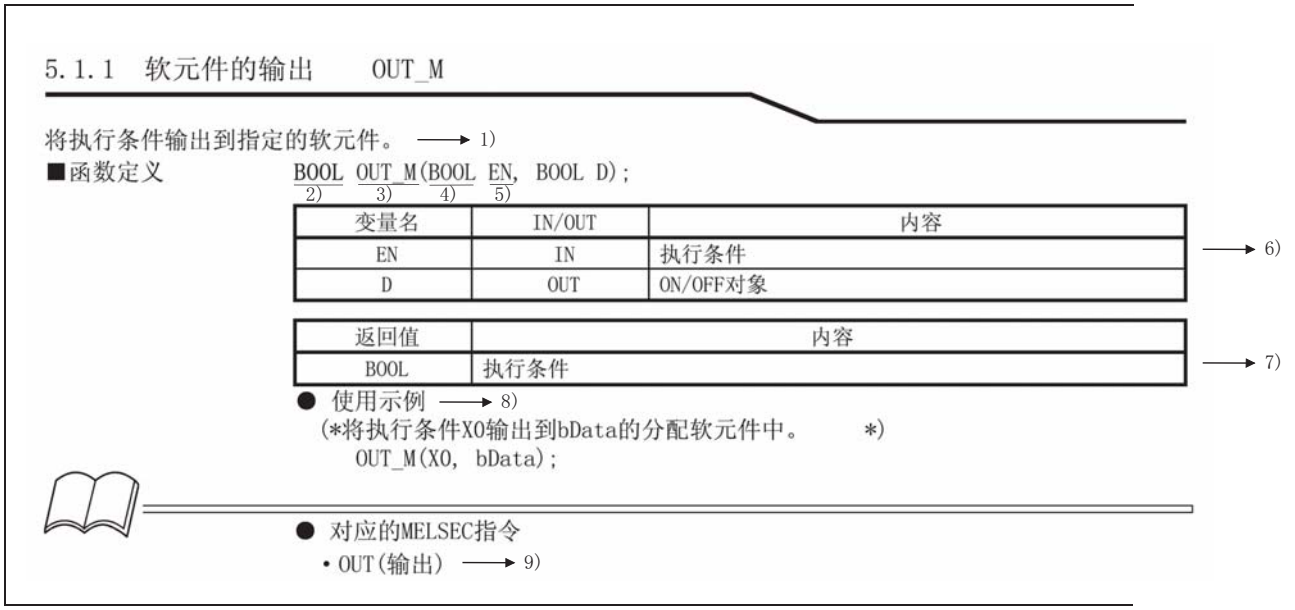
例6) (* START (* 处理中断 *)再开 结束 *)

4 - 32

5 MELSEC函数

函数的阅读方法

本书中记载了MELSEC函数的函数定义・自变量・返回值・使用示例。
MELSEC函数是以MELSEC公共指令为基础创建的。关于函数的兼容CPU、基本动作、详细功能、可用软元件、执行函数时发生的出错，请参阅“MELSEC-Q/L 编程手册(公共指令篇)”。
参照章节为本书中的“・对应的MELSEC指令”中记载的内容。



5.1.1 软元件的输出 OUT_M

将执行条件输出到指定的软元件。 → 1)

■函数定义

BOOL OUT_M(BOOL EN, BOOL D);
2) 3) 4) 5)

变量名	IN/OUT	内容
EN	IN	执行条件
D	OUT	ON/OFF对象

→ 6)

返回值	内容
BOOL	执行条件

→ 7)

● 使用示例 → 8)

```
(*将执行条件X0输出到bData的分配软元件中。      *)
   OUT_M(X0, bData);
```

● 对应的MELSEC指令

・OUT(输出) → 9)

1) 表示函数的功能。
2) 表示函数的数据类型。
3) 表示函数名。
4) 表示变量的数据类型。(STRING型的表示为STRING(字符数)。字符数6的情况下，变为STRING(6)。ARRAY型的表示为数据类型(要素数)。ANY16型的数组中要素数为3的情况下，变为ANY16(3)。)
5) 表示自变量名。
6) 表示函数中使用的自变量的列表(自变量名・IN/OUT・内容)。(STRING型的表示为ARRAY [0..要素数-1] OF 数据类型。ANY16型的数组中要素数为3的情况下，变为ARRAY [0..2] OF ANY16。)
7) 表示函数中使用的返回值的列表(返回值名・内容)。
8) 表示函数的使用示例。(表示使用了实际软元件・标签的示例。)
9) 表示函数对应的QCPU(Q模式)/LCPU MELSEC指令。

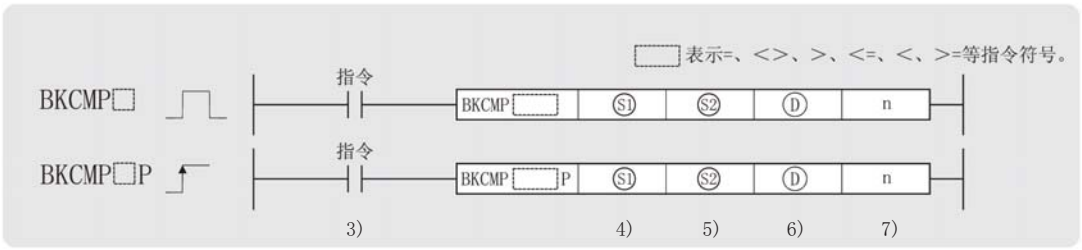
5 - 1

“MELSEC-Q/L 编程手册(公共指令篇)”MELSEC指令与本书MELSEC函数的对应如下所示。

“MELSEC-Q/L 编程手册(公共指令篇)”MELSEC指令

6.1.6 BIN16 位块数据比较 (BKCMP □、BKCMP □ P)

Basic | High performance | Process | Redundant | Universal → 1)



(S1) ：比较数据或者存储比较数据的软元件的起始编号 (BIN16 位)。
(S2) ：存储比较数据的软元件的起始编号 (BIN16 位)。
(D) ：存储运算结果的软元件的起始编号 (位)。
n ：比较的数据数 (BIN16 位)。

设置数据	内部软元件		R、ZR	J□\□		U□\G□	Zn	常数 K、H	其它
	位	字		位	字				
(S1)	--	○		--				○	--
(S2)	--	○		--				--	--
(D)	○	○		--				--	--
n	○	○		○				○	--

→ 2)

本书“MELSEC函数”

5.4.1 块数据比较(=) BKCMP_EQ_M

将从指定软元件算起的n点的BIN16位数据(字单位)用“=”进行比较。

■函数定义 BOOL BKCMP_EQ_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);
8) 9) 10) 11) 12)

1) 可使用的CPU类型
可使用指令的CPU类型如下所示。

2) 可用软元件
•MELSEC函数与MELSEC指令自变量的对应如下所述。(相同的自变量名之间相互对应。)

3)←→8)	4)←→9)	5)←→10)	7)←→11)	6)←→12)

要 点

● 使用MELSEC・IEC函数的自变量时的注意事项
自变量为ANY32型的情况下，可指定的数据类型为DINT型，因此不能指定实际软元件。只能指定双字型的标签。但是，可以指定位数指定。

```
例) BSQR_MD(BOOL EN,ANY16 s,ANY32 d);
                                    (*BSQR_MD的函数定义    *)
    BSQR_MD(X0,D0,dData);           (*程序示例            *)
```

在MELSEC公共指令中，

```
    BSQR(D0,W0);
```

及实际软元件可以记述，在MELSEC・IEC函数中不能记述。

```
    BSQR_MD(X0,D0,W0);    ← 变为出错状态。
```

自变量为REAL型的情况下，可指定的数据类型为实数型的标签或者可以直接记述实数值。
不能指定实际软元件。

```
例) ESTR_M(BOOL EN,REAL s1,ANY16(3) s2,STRING d);
                                    (*ESTR_M的函数定义    *)
    ESTR_M(X0,rData,ArrayData,sData);
                                    (*程序示例            *)
```

在MELSEC公共指令中，

```
    ESTR(R0,R10,D10);
```

及实际软元件可以记述，但在MELSEC・IEC函数中不能记述。

```
    ESTR_M(X0,R0,ArrayData,sData);←←变为出错状态。
```


5 - 3

5.1 输出

5.1.1 软元件的输出 OUT_M

将执行条件输出到指定的软元件。

■函数定义 **BOOL** OUT_M(BOOL EN, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件
D	OUT	ON/OFF对象

返回值	内容
BOOL	执行条件

● 使用示例

```
(*将执行条件X0输出到bData的分配软元件中。      *)
   OUT_M(X0, bData);
```

● 对应的MELSEC指令

• OUT(输出)

5.1.2 低速型 TIMER_M

定时器(低速型、低速累计定时器)的线圈变为ON后进行计测直至达到设置值，如果时间到(计算值≧设置值)则触点变为如下状态。

a 触点: 导通 b触点: 非导通

■函数定义 **BOOL** TIMER_M(BOOL EN, BOOL TCoil, ANY16 TValue);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
TCoil	IN	TS、TC软元件或STS、STC软元件(位数据)
TValue	IN	定时器的设置值(BIN16位数据)

备注) 定时器的设置值中指定常数的情况下，只能指定10进制数。

定时器的设置值指定范围为0～32767。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则TC0变为ON并开始计算直至达到TValue，
 如果时间到                                          *)
(*(计算值≧设置值)则触点变为下述状态。                *)
(* a 触点: 导通  b触点: 非导通                       *)
   TIMER_M(X0, TC0, TValue);
```

●对应的MELSEC指令

• OUT T(低速型)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.1.3 高速型 TIMER_H_M

定时器(高速型、高速累计定时器)的线圈变为ON后开始计算直至达到设置值，如果时间到(计算值≧设置值)则触点变为以下状态。
a 触点: 导通 b触点: 非导通

■函数定义

BOOL TIMER_H_M(BOOL EN, BOOL TCoil, ANY16 TValue);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
TCoil	IN	TS、TC软元件或STS、STC软元件(位数据)
TValue	IN	定时器的设置值(BIN16位数据)

备注) 定时器的设置值中指定常数的情况下，只能指定10进制数。
定时器的设置值可指定范围为0至32767。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，TC0将ON并开始计算直至达到TValue，如果时间到    *)
(*(计算值≧TValue则触点变为以下状态。                                    *)
(*a 触点: 导通  b触点: 非导通                                           *)
   TIMER_H_M(X0, TC0, TValue);
```

●对应的MELSEC指令
• OUTH T(高速型)

5.1.4 计数器 COUNTER_M

对计数器的当前值(计数值)进行+1，如果计数到(当前值=设置值)则触点将变为以下状态。
a 触点: 导通 b触点: 非导通

■函数定义

BOOL COUNTER_M(BOOL EN, BOOL CCoil, ANY16 CValue);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
CCoil	IN	CS、CC软元件的编号(位数据)
CValue	IN	计数器的设置值(BIN16位数据)

备注) 计数器的设置值中指定常数的情况下，只能指定10进制数。
计数器设置值的可指定范围为0至32767。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则在CC0变为OFF→ON时对当前值(计数值)          *)
(*进行+1，如果计数到(当前值= CValue)则触点将变为以下状态。            *)
(*a 触点: 导通  b触点: 非导通                                         *)
   COUNTER_M(X0, CC0, CValue);
```

●对应的MELSEC指令
• OUT C(计数器)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.1.5 软元件的设置 SET_M

执行条件成立时，将指定的软元件置为以下状态。
- 位软元件：将线圈・触点置为ON。
- 字软元件的位指定时：将指定位置为1。

■函数定义 **BOOL** SET_M(BOOL EN, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	OUT	设置数据

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON则将bData的分配软元件置为ON。                    *)
   SET_M(X0, bData);
```

●对应的MELSEC指令
- SET(软元件的设置)

5.1.6 软元件的复位 RST_M

执行条件成立时，将指定的软元件置为以下状态。
- 位软元件：将线圈・触点置为OFF。
- 定时器、计数器：在当前值中代入0后，将线圈・触点置为OFF 。
- 字软元件的位指定时：将指定位置为0 。
- 定时器、计数器以外的字软元件：在软元件内容中代入0。

■函数定义 **BOOL** RST_M(BOOL EN, ANY_SIMPLE D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	OUT	复位数据

备注) 自变量“D”中不能使用DINT/REAL/STRING型。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON则bData的分配软元件变为OFF。                    *)
   RST_M(X0, bData);
```

●对应的MELSEC指令
- RST(软元件的复位)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.1.7 直接输出的脉冲化 DELTA_M

执行条件成立时，对指定的直接访问输出(DY)进行脉冲输出。

■函数定义 **BOOL** DELTA_M(BOOL EN, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	OUT	脉冲输出的数据(DY软元件)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON则进行软元件DY0的脉冲化。                    *)
    DELTA_M(X0, DY0);
```

●对应的MELSEC指令

• DELTA(直接输出的脉冲化)

5.2 移动

5.2.1 软元件的1位移动 SFT_M

执行条件成立时，将指定的软元件置为以下状态。

- 位软元件的情况下:
  将指定的软元件编号的前一个软元件编号的ON・OFF状态移动至指定的软元件处，将前一个软元件编号置为OFF。
- 字软元件的位指定的情况下:
  将指定软元件的位的前一个位的1・0状态移动到指定的位处，将前一个位置为0。

■函数定义

BOOL SFT_M(BOOL EN, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	OUT	移动的数据

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将M10的ON・OFF移动到M11处并将M10置为OFF。  *)
  SFT_M(X0, M11);
(*如果执行条件X0变为ON，则将W100.1的ON・OFF移动到W100.2处并将W100.1置为OFF。*)
  SFT_M(X0, W100.2);
```

●对应的MELSEC指令

- SFT(位软元件移动)

5.3 结束

5.3.1 停止 STOP_M

执行条件成立时，对输出Y进行复位，停止CPU运算。
(与将RUN・STOP拨动开关置于STOP侧的情况相同。)

■函数定义

BOOL STOP_M(BOOL EN);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)

返回值	内容
BOOL	执行条件

●使用示例

```
(* 如果执行条件X0变为ON则停止CPU的运算。                    *)
    STOP_M(X0);
```

●对应的MELSEC指令

・STOP(顺控程序停止)

5 - 9

5.4 比较运算

5.4.1 块数据比较(=) BKCMP_EQ_M

将从指定软元件算起的n点的BIN16位数据(字单位)用“=”进行比较。

■函数定义 **BOOL** BKCMP_EQ_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

自变量名	IN/OUT	内容			
EN	IN	执行条件(仅TRUE时执行函数)			
S1	IN	被比较的数据(BIN16位数据)			
S2	IN	比较数据(BIN16位数据)			
n	IN	比较数据数(BIN16位数据)			
D	OUT	比较结果(位)	比较结果	比较条件成立时	ON
				比较条件不成立时	OFF

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储值的点数的数据与      *)
(*从D200算起的D0中存储值的点数的数据用“=”进行比较运算，并将其结果存储  *)
(*到M0以后。                                                          *)
   BKCMP_EQ_M(X0, D100, D200, D0, M0);
```

●对应的MELSEC指令

• BKCMP=(BIN16位块数据比较(=))

5.4.2 块数据比较(<>) BKCMP_NE_M

将从指定软元件算起n点的BIN16位数据(字单位)用“<>”进行比较。

■函数定义 **BOOL** BKCMP_NE_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

自变量名	IN/OUT	内容			
EN	IN	执行条件(仅TRUE时执行函数)			
S1	IN	被比较的数据(BIN16位数据)			
S2	IN	比较数据(BIN16位数据)			
n	IN	比较数据数(BIN16位数据)			
D	OUT	比较结果(位)	比较结果	比较条件成立时	ON
				比较条件不成立时	OFF

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储值的点数的数据与      *)
(*从D200算起D0中存储的值点数的数据用“<>”进行比较运算，并将其结果      *)
(*存储到M0以后。                                                      *)
   BKCMP_NE_M(X0, D100, D200, D0, M0);
```

●对应的MELSEC指令

• BKCMP<>(BIN16位块数据比较(<>))

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.4.3 块数据比较(>) BKCMP_GT_M

将从指定软元件算起n点的BIN16位数据(字单位)用“>”进行比较。

■函数定义 **BOOL** BKCMP_GT_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

自变量名	IN/OUT	内容			
EN	IN	执行条件(仅TRUE时执行函数)			
S1	IN	被比较的数据(BIN16位数据)			
S2	IN	比较数据(BIN16位数据)			
n	IN	比较数据数(BIN16位数据)			
D	OUT	比较结果(位)	比较结果	比较条件成立时	ON
				比较条件不成立时	OFF

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从D100算起的D0中存储的值点数的数据与      *)
(*从D200算起D0中存储的值点数的数据用“>”进行比较运算，并将其结果    *)
(*存储到M0以后。                                                    *)
   BKCMP_GT_M(X0, D100, D200, D0, M0);
```

●对应的MELSEC指令

• BKCMP>(BIN16位块数据比较(>))

5.4.4 块数据比较(<=) BKCMP_LE_M

将从指定软元件算起n点的BIN16位数据(字单位)用“<=”进行比较。

■函数定义 **BOOL** BKCMP_LE_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

自变量名	IN/OUT	内容			
EN	IN	执行条件(仅TRUE时执行函数)			
S1	IN	被比较的数据(BIN16位数据)			
S2	IN	比较数据(BIN16位数据)			
n	IN	比较数据数(BIN16位数据)			
D	OUT	比较结果(位)	比较结果	比较条件成立时	ON
				比较条件不成立时	OFF

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储值点数的数据与      *)
(*从D200算起D0中存储的值的点数的数据用“<=”进行比较运算，并将其结果  *)
(*存储到M0以后。                                                    *)
   BKCMP_LE_M(X0, D100, D200, D0, M0);
```

●对应的MELSEC指令

• BKCMP<=(BIN16位块数据比较(<=))

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.4.5 块数据比较(<) BKCMP_LT_M

将从指定软元件算起n点的BIN16位数据(字单位)用“<”进行比较。

■函数定义 **BOOL** BKCMP_LT_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

自变量名	IN/OUT	内容			
EN	IN	执行条件(仅TRUE时执行函数)			
S1	IN	被比较的数据(BIN16位数据)			
S2	IN	比较数据(BIN16位数据)			
n	IN	比较数据数(BIN16位数据)			
D	OUT	比较结果(位)	比较结果	比较条件成立时	ON
				比较条件不成立时	OFF

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储值点数的数据与      *)
(*从D200算起D0中存储的值的点数的数据用“<”进行比较运算，并将其结果  *)
(*存储到M0以后。                                                    *)
   BKCMP_LT_M(X0, D100, D200, D0, M0);
```

●对应的MELSEC指令

• BKCMP<(BIN16位块数据比较(<))

5.4.6 块数据比较(>=) BKCMP_GE_M

将从指定软元件算起n点的BIN16位数据(字单位)用“>=”进行比较。

■函数定义 **BOOL** BKCMP_GE_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, BOOL D);

自变量名	IN/OUT	内容			
EN	IN	执行条件(仅TRUE时执行函数)			
S1	IN	被比较的数据(BIN16位数据)			
S2	IN	比较数据(BIN16位数据)			
n	IN	比较数据数(BIN16位数据)			
D	OUT	比较结果(位)	比较结果	比较条件成立时	ON
				比较条件不成立时	OFF

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储值点数的数据与      *)
(*从D200算起D0中存储的值的点数的数据用“>=”进行比较运算，并将其结果  *)
(*存储到M0以后                                                      *)
   BKCMP_GE_M(X0, D100, D200, D0, M0);
```

●对应的MELSEC指令

• BKCMP>=(BIN16位块数据比较(>=))

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5 算术运算

5.5.1 BCD4位的加法(2软元件) BPLUS_M

将指定的2个BCD4位数据进行加法运算。

■函数定义

```
BOOL BPLUS_M(BOOL EN, ANY16 S1, ANY16 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行加法运算的数据(BCD4位数据)
D	IN/OUT	被加的数据・加法结果(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，将D0与D100中存储的BCD4位数据进行加法运算，      *)
(*将加法结果存储到D100中。                                          *)
   BPLUS_M(X0, D0, D100);
```

●对应的MELSEC指令

・B+(BCD4位数据加法)

5.5.2 BCD4位的加法(3软元件) BPLUS_3_M

将指定的2个BCD4位数据进行加法运算。

■函数定义

```
BOOL BPLUS_3_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被加的数据(BCD4位数据)
S2	IN	进行加法运算的数据(BCD4位数据)
D	OUT	加法结果(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D1与D2中存储的BCD4位数据进行加法运算，    *)
(*将加法结果存储到D100中。                                          *)
   BPLUS_3_M(X0, D1, D2, D100);
```

●对应的MELSEC指令

・B+(BCD4位数据加法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.3 BCD4位的减法(2软元件) BMINUS_M

将指定的2个BCD4位数据进行减法运算。

■函数定义 **BOOL** BMINUS_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	减法运算数据(BCD4位数据)
D	IN/OUT	被减数据・减法结果(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0与D100中存储的BCD4位数据进行减法运算，  *)
(*将减法结果存储到D100中。                                    *)
   BMINUS_M(X0, D0, D100);
```

●对应的MELSEC指令

・B-(BCD4位数据减法)

5.5.4 BCD4位的减法(3软元件) BMINUS_3_M

将指定的2个BCD4位数据进行减法运算。

■函数定义 **BOOL** BMINUS_3_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被减数据(BCD4位数据)
S2	IN	进行减法运算的数据(BCD4位数据)
D	OUT	减法结果(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D1与D2中存储的BCD4位数据进行减法运算，  *)
(*将减法结果存储到D100中。                                    *)
   BMINUS_3_M(X0, D1, D2, D100);
```

●对应的MELSEC指令

・B-(BCD4位数据减法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.5 BCD8位的加法(2软元件) DBPLUS_M

将指定的2个BCD8位数据进行加法运算。

■函数定义 **BOOL** DBPLUS_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	加法数据(BCD8位数据)
D	IN/OUT	被加的数据・加法结果(BCD8位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与Result中存储的BCD8位数据进行    *)
(*加法运算，并将加法结果存储到Result中。                              *)
   DBPLUS_M(X0, dwData1, Result);
```

●对应的MELSEC指令

・DB+(BCD8位数据加法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.6 BCD8位的加法(3软元件) DBPLUS_3_M

将指定的2个BCD8位数据进行加法运算。

■函数定义 **BOOL** DBPLUS_3_M(BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被加的数据(BCD8位数据)
S2	IN	进行加法运算的数据(BCD8位数据)
D	OUT	加法结果(BCD8位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与dwData2中存储的BCD8位数据      *)
(*进行加法运算，并将加法结果存储到Result中。                          *)
   DBPLUS_3_M(X0, dwData1, dwData2, Result);
```

●对应的MELSEC指令

・DB+(BCD8位数据加法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.7 BCD8位的减法(2软元件) DBMINUS_M

将指定的2个BCD8位数据进行减法运算。

■函数定义 **BOOL** DBMINUS_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行减法运算的数据。(BCD8位数据)
D	IN/OUT	被减数据・减法结果(BCD8位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与Result中存储的BCD8位数据*)
(*进行减法运算，并将减法结果存储到Result中。                *)
   DBMINUS_M(X0, dwData1,Result);
```

●对应的MELSEC指令

・DB-(BCD8位数据减法)

5.5.8 BCD8位的减法(3软元件) DBMINUS_3_M

将指定的2个BCD8位数据进行减法运算。

■函数定义 **BOOL** DBMINUS_3_M(BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被减数据(BCD8位数据)
S2	IN	进行减法运算的数据。(BCD8位数据)
D	OUT	减法结果(BCD8位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与dwData2中存储的BCD8位数据      *)
(*进行减法运算，并将减法结果存储到Result中。                *)
   DBMINUS_3_M(X0, dwData1, dwData2, Result);
```

●对应的MELSEC指令

・DB-(BCD8位数据减法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.9 BCD4位的乘法 BMULTI_M

将指定的2个BCD4位数据进行乘法运算。

■函数定义 **BOOL** BMULTI_M(BOOL EN, ANY16 S1, ANY16 S2,ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被乘数据(BCD4位数据)
S2	IN	进行乘法运算的数据(BCD4位数据)
D	OUT	乘法结果(BCD8位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D1与D2中存储的BCD4位数据进行乘法运算，  *)
(*并将乘法结果存储到Result中。                                  *)
   BMULTI_M(X0, D1, D2, Result);
```

●对应的MELSEC指令

・B*(BCD4位数据乘法)

5.5.10 BCD4位的除法 BDIVID_M

将指定的2个BCD4位数据进行除法运算。

■函数定义 **BOOL** BDIVID_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16(2) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	被除的数据(BCD4位数据)		
S2	IN	进行除法运算的数据(BCD4位数据)		
D	OUT	除法结果 (ARRAY [0..1] OF ANY16)	D[0] D[1]	商 余数

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D1与D2中存储的BCD4位数据进行除法运算，  *)
(*并将除法结果存储到数组ArrayResult中。                          *)
   BDIVID_M(X0, D1, D2, ArrayResult);
```

●对应的MELSEC指令

・B/(BCD4位数据除法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.11 BCD8位的乘法 DBMULTI_M

将指定的2个BCD8位数据进行乘法运算。

■函数定义

```
BOOL DBMULTI_M(BOOL EN, ANY32 S1, ANY32 S2, ANY16(4) D);
```

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	被乘数据(BCD8位数据)		
S2	IN	进行乘法运算的数据(BCD8位数据)		
D	OUT	乘法结果 (ARRAY [0..3] OF ANY16)	D[0]	低4位
			D[1]	↓
			D[2]	
			D[3]	高4位

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与dwData2中存储的BCD8位数据        *)
(*进行乘法运算，并将乘法结果存储到数组ArrayResult中。          *)
   DBMULTI_M(X0, dwData1, dwData2, ArrayResult);
```

●对应的MELSEC指令

• DB*(BCD8位数据乘法)

5.5.12 BCD8位的除法 DBDIVID_M

将指定的2个BCD8位数据进行除法运算。

■函数定义

```
BOOL DBDIVID_M(BOOL EN, ANY32 S1, ANY32 S2, ANY32(2) D);
```

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	被除数据(BCD8位数据)		
S2	IN	进行除法运算的数据(BCD8位数据)		
D	OUT	除法结果 (ARRAY [0..1] OF ANY32)	D[0]	商
			D[1]	余数

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与dwData2中存储的BCD8位数据        *)
(*进行除法运算，将除法结果存储到数组ArrayResult中。          *)
   DBDIVID_M(X0, dwData1, dwData2, ArrayResult);
```

●对应的MELSEC指令

• DB/(BCD8位数据除法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.13 字符串数据合并(2软元件) STRING_PLUS_M

将指定的字符串数据相连接。

■函数定义

BOOL STRING_PLUS_M(BOOL EN, STRING S1, STRING D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	连接数据(字符串数据)
D	IN/OUT	被连接的数据・连接结果(字符串数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则在StrResult中存储的字符串的后面将字符串“ABC”与其*)
(*合并，并将合并后的字符串存储到StrResult中。                    *)
  STRING_PLUS_M(X0, “ABC”, StrResult);
```

●对应的MELSEC指令

・$+(字符串的合并)

5.5.14 字符串数据合并(3软元件) STRING_PLUS_3_M

将指定的字符串数据相连接。

■函数定义

BOOL STRING_PLUS_3_M(BOOL EN, STRING S1, STRING S2, STRING D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被连接的数据(字符串数据)
S2	IN	连接数据(字符串数据)
D	OUT	连接结果(字符串数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则在StrData1中存储的字符串的后面将StrData2中存储  *)
(*的字符串与其合并，并将合并后的字符串存储到StrResult中。                *)
  STRING_PLUS_3_M(X0, StrData1,StrData2, StrResult);
```

●对应的MELSEC指令

・$+(字符串的合并)

5.5.15 BIN块加法 BKPLUS_M

将从指定软元件算起n点的BIN16位数据进行加法运算。

■函数定义

```
BOOL BKPLUS_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n ,ANY16 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被加的数据(BIN16位数据)
S2	IN	进行加法运算的数据(BIN16位数据)
n	IN	进行加法运算的数据数(BIN16位数据)
D	OUT	加法结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储值的点数的数据与 *)
(*从D200算起D0中存储的值的点数的数据进行加法运算，并将其结果 *)
(*存储到D1000以后。 *)
  BKPLUS_M(X0, D100, D200, D0, D1000);
```

●对应的MELSEC指令

• BK+(块数据加法)

5.5.16 BIN块减法 BKMINUS_M

将从指定软元件算起n点的BIN16位数据进行减法运算。

■函数定义

```
BOOL BKMINUS_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被减数据(BIN16位数据)
S2	IN	进行减法运算的数据(BIN16位数据)
n	IN	进行减法运算的数据数(BIN16位数据)
D	OUT	减法结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储的值的点数的数据与 *)
(*从D200算起D0中存储的值的点数的数据进行减法运算，并将其结果 *)
(*存储到D1000以后。 *)
  BKMINUS_M(X0, D100, D200, D0, D1000);
```

●对应的MELSEC指令

• BK-(块数据减法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.17 递增 INC_M

将指定的BIN16位数据进行递增(+1)。

■函数定义 **BOOL** INC_M(BOOL EN, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	进行加法运算的数据・加法结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的数据进行+1。                    *)
   INC_M(X0, D0);
```

●对应的MELSEC指令

・INC(BIN16位递增)

5.5.18 递减 DEC_M

将指定的BIN16位数据进行递减(-1)。

■函数定义 **BOOL** DEC_M(BOOL EN, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	进行减法运算的数据・减法结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的数据进行-1。                    *)
   DEC_M(X0, D0);
```

●对应的MELSEC指令

・DEC(BIN16位递减)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.5.19 32位BIN递增 DINC_M

将指定的BIN32位数据进行递增(+1)。

■函数定义

BOOL DINC_M(BOOL EN, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	进行加法运算的数据・加法结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1中存储的数据进行+1。                *)
   DINC_M(X0, dwData1);
```

●对应的MELSEC指令

・DINC(BIN32位递增)

5.5.20 32位BIN递减 DDEC_M

将指定的BIN32位数据进行递减(-1)。

■函数定义

BOOL DDEC_M(BOOL EN, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	进行减法运算的数据・减法结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1中存储的数据进行-1。                *)
   DDEC_M(X0, dwData1);
```

●对应的MELSEC指令

・DDEC(BIN32位递减)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6 数据转换

5.6.1 BIN→BCD转换 BCD_M

将指定的BIN16位数据(0～9999)转换为BCD4位数据。

■函数定义 **BOOL** BCD_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D	OUT	转换结果(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的BIN数据进行BCD转换并          *)
(*存储到D100。                                                    *)
   BCD_M(X0, D0, D100);
```

●对应的MELSEC指令

• BCD(BIN→BCD4位)

5.6.2 32位BIN→BCD转换 DBCD_M

将指定的BIN32位数据(0～99999999)转换为BCD8位数据。

■函数定义 **BOOL** DBCD_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D	OUT	转换结果(BCD8位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1中存储的BIN数据进行BCD转换*)
(*并存储到Result中。                                            *)
   DBCD_M(X0, dwData1, Result);
```

●对应的MELSEC指令

• DBCD(BIN→BCD8位)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.3 BCD→BIN转换 BIN_M

将指定的BCD4位数据(0～9999)转换为BIN16位数据。

■函数定义 **BOOL** BIN_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BCD4位数据)
D	OUT	转换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的BCD数据进行BIN转换并          *)
(*存储到D100中。                                                  *)
   BIN_M(X0, D0, D100);
```

●对应的MELSEC指令

• BIN(BCD4位→BIN)

5.6.4 32位BCD→BIN转换 DBIN_M

将指定的BCD8位数据(0～99999999)转换为BIN32位数据。

■函数定义 **BOOL** DBIN_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BCD8位数据)
D	OUT	转换结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1中存储的BCD数据进行BIN转换          *)
(*并存储到Result中。                                              *)
   DBIN_M(X0, dwData1, Result);
```

●对应的MELSEC指令

• DBIN(BCD8位→BIN)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.5 浮动小数点→BIN转换 INT_E_MD

将指定的实数数据转换为BIN16位数据。

■函数定义 **BOOL** INT_E_MD(BOOL EN, REAL S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(实数数据)
D	OUT	转换结果(BIN16位数据)

备注) 自变量 “S1”中指定的实数数据的可指定范围为-32768～32767。
对于转换后的数据，实数的小数点以下第1位将变为四舍五入后的值。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将RealData1的实数数据转换为BIN16位数据，  *)
(*并存储到D0中。                                                    *)
   INT_E_MD(X0, RealData1, D0);
```

●对应的MELSEC指令

• INT(浮动小数点数据→BIN16位转换(单精度))

5.6.6 32位浮动小数点→BIN转换 DINT_E_MD

将指定的实数数据转换为BIN32位数据。

■函数定义 **BOOL** DINT_E_MD(BOOL EN, REAL S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(实数数据)
D	OUT	转换结果(BIN32位数据)

备注) 自变量 “S1” 中指定的实数数据的可指定范围为-2147483648～2147483647。
对于转换后的数据，实数的小数点以下第1位将变为四舍五入后的值。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将实数数据E2.6转换为BIN16位数据，  *)
(*并存储到Result中。                                          *)
   DINT_E_MD(X0, E2.6, Result);
```

●对应的MELSEC指令

• DINT(浮动小数点数据→BIN32位转换(双精度))

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.7 BIN→浮动小数点转换 FLT_M

将指定的BIN16位数据转换为实数数据。

■函数定义 **BOOL** FLT_M(BOOL EN, ANY16 S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D	OUT	转换结果(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100的BIN16位数据转换为实数数据，        *)
(*并存储到Result中。                                              *)
   FLT_M(X0, D100, Result);
```

●对应的MELSEC指令

・FLT(BIN16位→浮动小数点转换(单精度))

5.6.8 32位BIN→浮动小数点转换 DFLT_M

将指定的BIN32位数据转换为实数数据。

■函数定义 **BOOL** DFLT_M(BOOL EN, ANY32 S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D	OUT	转换结果(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的BIN32位数据转换为实数数据，     *)
(*并存储到Result中。                                              *)
   DFLT_M(X0, dwData1, RealResult);
```

●对应的MELSEC指令

・DFLT(BIN32位→浮动小数点(双精度))

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.9 16位BIN→32位BIN转换 DBL_M

将指定的BIN16位数据转换为带符号BIN32位数据。

■函数定义 **BOOL** DBL_M(BOOL EN, ANY16 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D	OUT	转换结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的BIN16位数据转换为带符号BIN32位数据      *)
(*并存储到Result中。                                                    *)
   DBL_M(X0, D0, Result);
```

●对应的MELSEC指令

• DBL(BIN16位→BIN32位)

5.6.10 32位BIN→16位BIN转换 WORD_M

将指定的BIN32位数据转换为带符号BIN16位数据。

■函数定义 **BOOL** WORD_M(BOOL EN, ANY32 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D	OUT	转换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1中存储的BIN32位数据转换为      *)
(*BIN16位数据，并存储到D0中。                                      *)
   WORD_M(X0, dwData1, D0);
```

●对应的MELSEC指令

• WORD(BIN32位→BIN16位)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.11 BIN→格雷码转换 GRY_M

将指定的BIN16位数据转换为格雷码16位数据。

■函数定义 **BOOL** GRY_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D	OUT	转换结果(格雷码16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的BIN16位数据转换为格雷码16位      *)
(*数据并存储到D100中。                                          *)
   GRY_M(X0, D0, D100);
```

●对应的MELSEC指令

• GRY(BIN16位→格雷码)

5.6.12 32位BIN→格雷码转换 DGRY_M

将指定的BIN32位数据转换为格雷码32位数据。

■函数定义 **BOOL** DGRY_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D	OUT	转换结果(格雷码32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的BIN32位数据转换为格雷码32    *)
(*位数据并存储到Result中。                                      *)
   DGRY_M(X0, dwData1, Result);
```

●对应的MELSEC指令

• DGRY(BIN32位→格雷码)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.13 格雷码→BIN转换 GBIN_M

将指定的格雷码16位数据转换为BIN16位数据。

■函数定义 **BOOL** GBIN_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(格雷码16位数据)
D	OUT	转换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100的格雷码16位数据转换为BIN16位数据      *)
(*并存储到D200中。                                                    *)
    GBIN_M(X0, D100, D200);
```

●对应的MELSEC指令

• GBIN(格雷码→BIN16位)

5.6.14 32位格雷码→BIN转换 DGBIN_M

将指定的格雷码32位数据转换为BIN32位数据。

■函数定义 **BOOL** DGBIN_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(格雷码32位数据)
D	OUT	转换结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的格雷码32位数据转换为BIN32      *)
(*位数据并存储到Result中。                                          *)
    DGBIN_M(X0, dwData1, Result);
```

●对应的MELSEC指令

• DGBIN(格雷码→BIN16位)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.15 16位BIN的2的补数 NEG_M

将指定的BIN16位数据的符号进行取反。(2的补数)

■函数定义

```
BOOL NEG_M(BOOL EN, ANY16 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	符号取反的数据・符号取反结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的BIN16位数据的符号取反并存储到D0中。      *)
   NEG_M(X0, D0);
```

●对应的MELSEC指令

・NEG(BIN16位数据2的补数)

5.6.16 32位BIN的2的补数 DNEG_M

将指定的BIN32位数据的符号进行取反。(2的补数)

■函数定义

```
BOOL DNEG_M(BOOL EN, ANY32 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	符号取反的数据・符号取反结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将Result的BIN32位数据的符号取反          *)
(*并存储到Result中。                                              *)
   DNEG_M(X0, Result);
```

●对应的MELSEC指令

・DNEG(BIN32位数据2的补数)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.17 浮动小数点的2的补数 ENEG_M

将指定的实数数据的符号进行取反。(2的补数)

■函数定义

BOOL ENEG_M(BOOL EN, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	符号取反的数据・转换结果(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将Result的实数数据的符号取反          *)
(*并存储到Result中。                                          *)
   ENEG_M(X0, Result);
```

●对应的MELSEC指令

・ENEG(浮动小数点数据符号取反(单精度))

5.6.18 块转换BIN→BCD转换 BKBCD_M

将从指定软元件算起n点的BIN16位数据(0～9999)转换为BCD4位数据。

■函数定义

BOOL BKBCD_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
n	IN	转换的数据数
D	OUT	转换结果(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D0算起的W0中存储的值的点数的BIN16位数据    *)
(*进行BCD转换，并将其结果存储到D100以后。                          *)
   BKBCD_M(X0, D0, W0, D100);
```

●对应的MELSEC指令

・BKBCD(块BIN16位数据→BCD4位转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.6.19 块转换BCD→BIN转换 BKBIN_M

将从指定软元件算起n点的BCD4位数据(0～9999)转换为BIN16位数据。

■函数定义

BOOL BKBIN_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BCD4位数据)
n	IN	转换的数据数
D	OUT	转换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D0算起的W0中存储的值的点数的BCD数据    *)
(*进行BIN转换，并将其结果存储到D100以后。                          *)
  BKBIN_M(X0, D0, W0, D100);
```

●对应的MELSEC指令

・BKBIN(块BCD4位数据→BIN16位转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 32

5.7 数据传送

5.7.1 16位数据否定传送 CML_M

将指定的BIN16位数据逐位取反。

■函数定义 **BOOL** CML_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	位取反的数据(BIN16位数据)
D	OUT	取反结果传送目标(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将M0～M7的数据取并传送至D0。          *)
  CML_M(X0, K2M0, D0);
```

●对应的MELSEC指令

・CML(16位否定传送)

5.7.2 32位数据否定传送 DCML_M

将指定的BIN32位数据逐位取反。

■函数定义 **BOOL** DCML_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	位取反的数据(BIN32位数据)
D	OUT	取反结果传送目标(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的数据逐位取反并传送至Result。    *)
  DCML_M(X0, dwData1, Result);
```

●对应的MELSEC指令

・DCML(32位否定传送)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.7.3 块传送 BMOV_M

将从指定软元件算起n点的BIN16位数据进行批量传送。

■函数定义 **BOOL** BMOV_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	传送的数据(BIN16位数据)
n	IN	传送的数据数(BIN16位数据)
D	OUT	传送目标(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D0中指定软元件算起的W0中存储值的      *)
(*点数的16位数据传送至从D100算起的W0中存储的值的点数。            *)
  BMOV_M(X0, D0, W0, D100);
```

●对应的MELSEC指令

• BMOV(块16位传送)

5.7.4 同一数据块传送 FMOV_M

将指定软元件的16位数据传送指定的软元件算起的n点。

■函数定义 **BOOL** FMOV_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	传送的数据(BIN16位数据)
n	IN	传送的数据数(BIN16位数据)
D	OUT	传送目标(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的16位数据传送从D100算起的W0中存储的点数。*)
  FMOV_M(X0, D0, W0, D100);
```

●对应的MELSEC指令

• FMOV(块16位数据传送)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.7.5 16位数据替换 XCH_M

将指定的2个BIN16位数据进行替换。

■函数定义 **BOOL** XCH_M(BOOL EN, ANY16 D1, ANY16 D2);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D1	IN/OUT	替换的数据・替换结果(BIN16位数据)
D2	IN/OUT	替换的数据・替换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100与D200的16位数据进行替换。        *)
  XCH_M(X0, D100, D200);
```

●对应的MELSEC指令

・XCH(16位数据替换)

5.7.6 32位数据替换 DXCH_M

将指定的2个BIN32位数据进行替换。

■函数定义 **BOOL** DXCH_M(BOOL EN, ANY32 D1, ANY32 D2);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D1	IN/OUT	替换的数据・替换结果(BIN32位数据)
D2	IN/OUT	替换的数据・替换结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与dwData2的32位数据进行替换。    *)
  DXCH_M(X0, dwData1, dwData2);
```

●对应的MELSEC指令

・DXCH(32位数据替换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.7.7 块数据替换 BXCH_M

将指定的软元件算起的n点的BIN16位数据进行替换。

■函数定义 **BOOL** BXCH_M(BOOL EN, ANY16 n, ANY16 D1, ANY16 D2);

变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	替换的数据数(BIN16位数据)
D1	IN/OUT	替换的数据・替换结果(BIN16位数据)
D2	IN/OUT	替换的数据・替换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*将从D100算起3点的16位数据与从D200算起3点的16位数据进行替换。          *)
  BXCH_M(X0, K3, D100, D200);
```

●对应的MELSEC指令

・BXCH(块16位替换)

5.7.8 高低字节替换 SWAP_MD

将指定的软元件的高8位与低8位进行替换。

■函数定义 **BOOL** SWAP_MD(BOOL EN, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	IN/OUT	替换的数据・替换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的高8位与低8位进行替换。          *)
  SWAP_MD(X0, D0);
```

●对应的MELSEC指令

・SWAP(高低字节替换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.8 程序执行控制

5.8.1 中断禁止 DI_M

即使发生了中断程序的中断原因，在EI_M被执行之前禁止中断程序的执行。

■函数定义 **BOOL** DI_M(BOOL EN);

自变量名	IN/OUT	内容
EN	IN	执行条件 (只能指定表示常时有效的值TRUE或常时ON软元件SM400。)

返回值	内容
BOOL	执行条件(常時TRUE)

●使用示例

```
(*在EI_M被执行之前禁止中断程序的执行。                    *)
  DI_M(TRUE);
```

●对应的MELSEC指令

・DI(中断禁止)

5.8.2 中断允许 EI_M

将执行DI_M时的中断禁止状态进行解除，将根据IMASK允许的中断指针编号的中断程序的执行置为允许。

■函数定义 **BOOL** EI_M(BOOL EN);

自变量名	IN/OUT	内容
EN	IN	执行条件 (只能指定表示常时有效的值TRUE或常时ON软元件SM400。)

返回值	内容
BOOL	执行条件(常時TRUE)

●使用示例

```
(*将执行DI_M时的中断禁止状态进行解除。                    *)
  EI_M(TRUE);
```

●对应的MELSEC指令

・EI(中断允许)

5.9 I/O刷新

5.9.1 I/O刷新 RFS_M

将指定的软元件算起 n 点的I/O软元件进行刷新。

■函数定义 **BOOL** RFS_M(BOOL EN, BOOL S1, ANY16 n);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	刷新的软元件(位数据)
n	IN	刷新的数据数(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件M0变为ON，则将从X100算起的32点的软元件进行刷新。        *)
  RFS_M(M0, X100, H20);
```

●对应的MELSEC指令

• RFS(I/O刷新)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10 逻辑运算指令

5.10.1 逻辑积(2软元件) WAND_M

将指定的2个BIN16位数据逐位进行逻辑积运算。

■函数定义

BOOL WAND_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行逻辑积运算的数据(BIN16位数据)
D	IN/OUT	被进行逻辑积运算的数据・运算结果(BIN16位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0与D10的16位数据逐位进行逻辑积，          *)
(*并将其结果存储到D10中。                                          *)
   WAND_M(X0, D0, D10);
```

●对应的MELSEC指令

・WAND(16位数据逻辑积)

5.10.2 逻辑积(3软元件) WAND_3_M

将指定的2个BIN16位数据逐位进行逻辑积运算。

■函数定义

BOOL WAND_3_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑积运算的数据(BIN16位数据)
S2	IN	进行逻辑积运算的数据(BIN16位数据)
D1	OUT	运算结果(BIN16位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0与D10的16位数据逐位进行逻辑积，          *)
(*并将其结果存储到D100中。                                         *)
   WAND_3_M(X0, D0, D10, D100);
```

●对应的MELSEC指令

・WAND(16位数据逻辑积)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.3 32位数据逻辑积(2软元件) DAND_M

将指定的2个BIN32位数据逐位进行逻辑积运算。

■函数定义 **BOOL** DAND_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行逻辑积运算的数据(BIN32位数据)
D	IN/OUT	被进行逻辑积运算的数据・运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与X30～X47的24位数据进行逻辑积，  *)
(*并将结果存储到dwData1中。                                          *)
   DAND_M(X0, K6X30, dwData1);
```

●对应的MELSEC指令

・DAND(32位数据逻辑积)

5.10.4 32位数据逻辑积(3软元件) DAND_3_M

将指定的2个BIN32位数据逐位进行逻辑积运算。

■函数定义 **BOOL** DAND_3_M(BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑积运算的数据(BIN32位数据)
S2	IN	进行逻辑积运算的数据(BIN32位数据)
D	OUT	运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与dwData2的32位数据进行逻辑积，  *)
(*并将结果存储到Result中。                                          *)
   DAND_3_M(X0, dwData1, dwData2, Result);
```

●对应的MELSEC指令

・DAND(32位数据逻辑积)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.5 块数据逻辑积 BKAND_M

将从指定的2个软元件算起n点的16位数据逐位进行逻辑积运算。

■函数定义

BOOL BKAND_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑运算的数据(BIN16位数据)
S2	IN	进行逻辑运算的数据(BIN16位数据)
n	IN	进行运算的数据数(BIN16位数据)
D	OUT	运算结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的D0中存储值的点数的数据与    *)
(*从D200算起D0中存储值的点数的数据进行逻辑积，并将其结果          *)
(*存储到D1000以后。                                          *)
   BKAND_M(X0, D100, D200, D0, D1000);
```

●对应的MELSEC指令

・BKAND(块逻辑积)

5.10.6 逻辑和(2软元件) WOR_M

将指定的2个BIN16位数据逐位进行逻辑和运算。

■函数定义

BOOL WOR_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行逻辑和运算的数据(BIN16位数据)
D	IN/OUT	被进行逻辑和运算的数据・运算结果(BIN16位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D10与D20的数据进行逻辑和，并将其结果    *)
(*存储到D10中。                                              *)
   WOR_M(X0, D10, D20);
```

●对应的MELSEC指令

・WOR(16位数据逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.7 逻辑和(3软元件) WOR_3_M

将指定的2个BIN16位数据逐位进行逻辑和运算。

■函数定义

BOOL WOR_3_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑和运算的数据(BIN16位数据)
S2	IN	进行逻辑和运算的数据(BIN16位数据)
D1	OUT	运算结果(BIN16位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件M0变为ON，则将X10～X1B的数据与D0的数据进行逻辑和，      *)
(*并将其结果输出到Y10～Y1B中。                                        *)
   WOR_3_M(M0, K3X10, D0, K3Y10)
```

●对应的MELSEC指令

・WOR(16位数据逻辑和)

5.10.8 32位数据逻辑和(2软元件) DOR_M

将指定的2个BIN32位数据逐位进行逻辑和运算。

■函数定义

BOOL DOR_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行逻辑和运算的数据(BIN32位数据)
D	IN/OUT	被进行逻辑和运算的数据・运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与Result的数据进行逻辑和，并将其结果 *)
(*输出到Result中。                                                    *)
   DOR_M(X0, dwData1, Result);
```

●对应的MELSEC指令

・DOR(32位数据逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.9 32位数据逻辑和(3软元件) DOR_3_M

将指定的2个BIN32位数据逐位进行逻辑和运算。

■函数定义

BOOL DOR_3_M(BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑和运算的数据(BIN32位数据)
S2	IN	进行逻辑和运算的数据(BIN32位数据)
D	OUT	运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的32位数据与X20～X3F的32位数据        *)
(*进行逻辑和，并将其结果输出到Result中。                                  *)
   DOR_3_M(X0, dwData1, K8X20, Result);
```

●对应的MELSEC指令

・DOR(32位数据逻辑和)

5.10.10 块数据逻辑和 BKOR_M

将从指定的2个软元件算起n点的16位数据逐位进行逻辑和运算。

■函数定义

BOOL BKOR_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑和运算的数据(BIN16位数据)
S2	IN	进行逻辑和运算的数据(BIN16位数据)
n	IN	进行运算的数据数(BIN16位数据)
D	OUT	运算结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D10至D0中存储的值的点数的数据与          *)
(*D20至D0中存储的值的点数的数据进行逻辑和，并将其结果                  *)
(*存储到D100以后。                                                    *)
   BKOR_M(X0, D10, D20, D0, D100)
```

●对应的MELSEC指令

・BKOR(块逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.11 排他逻辑和(2软元件) WXOR_M

将指定的2个BIN16位数据逐位进行排他逻辑和运算。

■函数定义

BOOL WXOR_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行排他逻辑和运算的数据(BIN16位数据)
D	IN/OUT	被进行排他逻辑和运算的数据・运算结果(BIN16位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D10与D20的16位数据进行排他逻辑和，      *)
(*并将其结果存储到D20中。                                          *)
   WXOR_M(X0, D10, D20);
```

●对应的MELSEC指令

・WXOR(16位数据排他逻辑和)

5.10.12 排他逻辑和(3软元件) WXOR_3_M

将指定的2个BIN16位数据逐位进行排他逻辑和运算。

■函数定义

BOOL WXOR_3_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行排他逻辑和运算的数据(BIN16位数据)
S2	IN	进行排他逻辑和运算的数据(BIN16位数据)
D	OUT	运算结果(BIN16位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D10与D20的16位数据进行排他逻辑和，      *)
(*并将其结果存储到D100中。                                         *)
   WXOR_3_M(X0, D10, D20, D100);
```

●对应的MELSEC指令

・WXOR(16位数据排他逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.13 32位数据排他逻辑和(2软元件) DXOR_M

将指定的2个BIN32位数据逐位进行排他逻辑和运算。

■函数定义

BOOL DXOR_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行排他逻辑和运算的数据(BIN32位数据)
D	IN/OUT	被进行排他逻辑和运算的数据・运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与Result的32位数据进行排他逻辑和，  *)
(*并将其结果存储到Result中。                                          *)
   DXOR_M(X0, dwData1, Result);
```

●对应的MELSEC指令

・DXOR(32位数据排他逻辑和)

5.10.14 32位数据排他逻辑和(3软元件) DXOR_3_M

将指定的2个BIN32位数据逐位进行排他逻辑和运算。

■函数定义

BOOL DXOR_3_M(BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行排他逻辑和运算的数据(BIN32位数据)
S2	IN	进行排他逻辑和运算的数据(BIN32位数据)
D	OUT	运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1与dwData2的32位数据进行排他逻辑和，  *)
(*并将其结果存储到Result中。                                          *)
   DXOR_3_M(X0, dwData1, dwData2, Result);
```

●对应的MELSEC指令

・DXOR(32位数据排他逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 45

5.10.15 块数据排他逻辑和　BKXOR_M

将从指定的2个软元件算起n点的16位数据逐位进行排他逻辑和运算。

■函数定义

BOOL BKXOR_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑运算的数据(BIN16位数据)
S2	IN	进行逻辑运算的数据(BIN16位数据)
n	IN	进行运算的数据数(BIN16位数据)
D	OUT	运算结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D10至D0中存储的值的点数的数据与　　*)
(*D20至D0中存储的值的点数的数据进行排他逻辑和，并将其结果　　*)
(*存储到D100以后。　　*)
  BKXOR_M(X0, D10, D20, D0, D100);
```

●对应的MELSEC指令

・BKXOR(块排他逻辑和)

5.10.16　否定排他逻辑和(2软元件)　WXNR_M

将指定的2个BIN16位数据逐位进行否定排他逻辑和运算。

■函数定义

BOOL WXNR_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行否定排他逻辑和运算的数据(BIN16位数据)
D	IN/OUT	被进行否定排他逻辑和运算的数据・运算结果(BIN16位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将X20～X2F的16位数据与D10的16位数据进行　　*)
(*否定排他逻辑和，并将其结果存储到D10中。　　*)
  WXNR_M(M0, K4X20, D10);
```

●对应的MELSEC指令

・WXNR(16位数据否定排他逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.17 否定排他逻辑和(3软元件) WXNR_3_M

将指定的2个BIN16位数据逐位进行否定排他逻辑和运算。

■函数定义

BOOL WXNR_3_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行否定排他逻辑和运算的数据(BIN16位数据)
S2	IN	进行否定排他逻辑和运算的数据(BIN16位数据)
D	OUT	运算结果(BIN16位数据)

备注) 自变量“S1”与“D”、“S2”与“D”可以指定相同的软元件。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将X20～X2F的16位数据与D0的16位数据          *)
(*进行否定排他逻辑和，并将其结果存储到D100中。                        *)
   WXNR_3_M(X0, K4X20, D0, D100);
```

●对应的MELSEC指令

・WXNR(16位数据否定排他逻辑和)

5.10.18 32位数据否定排他逻辑和(2软元件) DXNR_M

将指定的2个BIN32位数据逐位进行否定排他逻辑和运算。

■函数定义

BOOL DXNR_M(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行否定排他逻辑和运算的数据(BIN32位数据)
D	IN/OUT	被进行否定排他逻辑和运算的数据・运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的32位数据与Result的32位数据          *)
(*进行否定排他逻辑和，并将其结果存储到Result中。                        *)
   DXNR_M(X0, dwData1, Result);
```

●对应的MELSEC指令

・DXNR(32位数据否定排他逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.10.19 32位数据否定排他逻辑和(3软元件) DXNR_3_M

将指定的2个BIN32位数据逐位进行否定排他逻辑和运算。

■函数定义

BOOL DXNR_3_M(BOOL EN, ANY32 S1, ANY32 S2, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行否定排他逻辑和运算的数据(BIN32位数据)
S2	IN	进行否定排他逻辑和运算的数据(BIN32位数据)
D	OUT	运算结果(BIN32位数据)

备注) 位软元件的情况下，超出位数指定以上将被作为“0(零)”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的32位数据与dwData2的32位数据      *)
(*进行否定排他逻辑和，并将其结果存储到Result中。                      *)
   DXNR_3_M(X0, dwData1, dwData2, Result);
```

●对应的MELSEC指令

・DXNR(32位数据否定排他逻辑和)

5.10.20 块数据否定排他逻辑和 BKXNR_M

将从指定的2个软元件算起n点的16位数据逐位进行否定排他逻辑和运算。

■函数定义

BOOL BKXNR_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行逻辑运算的数据(BIN16位数据)
S2	IN	进行逻辑运算的数据(BIN16位数据)
n	IN	进行运算的数据数(BIN16位数据)
D	OUT	运算结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100至D0中存储的值的点数的数据与      *)
(*W100至D0中存储的值的点数的数据进行否定排他逻辑和，并将其          *)
(*结果存储到D200以后。                                            *)
   BKXNR_M(X0, D100, W100, D0, D200);
```

●对应的MELSEC指令

・BKXNR(块否定排他逻辑和)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.11 旋转

5.11.1 右旋转(不包含进位标志) ROR_M

将指定的BIN16位数据不包含进位标志向右旋转n位。

■函数定义

BOOL ROR_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～15)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN16位数据)

备注) “D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的数据不包含进位标志向右旋转3位。      *)
  ROR_M(X0, K3, D0);
```

●对应的MELSEC指令

・ROR(16位数据的右旋转)

5.11.2 右旋转(包含进位标志) RCR_M

将指定的BIN16位数据包含进位标志向右旋转n位。

■函数定义

BOOL RCR_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～15)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN16位数据)

备注) “D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的数据包含进位标志向右旋转3位。      *)
  RCR_M(X0, K3, D0);
```

●对应的MELSEC指令

・RCR(16位数据的右旋转)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.11.3 左旋转(不包含进位标志) ROL_M

将指定的BIN16位数据不包含进位标志向左旋转n位。

■函数定义

BOOL ROL_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～15)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN16位数据)

备注) “D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的数据不包含进位标志向左旋转3位。      *)
  ROL_M(X0, K3, D0);
```

●对应的MELSEC指令

・ROL(16位数据的左旋转)

5.11.4 左旋转(包含进位标志) RCL_M

将指定的BIN16位数据包含进位标志向左旋转n位。

■函数定义

BOOL RCL_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～15)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN16位数据)

备注) “D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0的数据包含进位标志向左旋转3位。      *)
  RCL_M(X0, K3, D0);
```

●对应的MELSEC指令

・RCL(16位数据的左旋转)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 50

5.11.5 32位数据右旋转(不包含进位标志) DROR_M

将指定的BIN32位数据不包含进位标志向右旋转n位。

■函数定义 **BOOL** DROR_M(BOOL EN, ANY16 n, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～31)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN32位数据)

备注) “D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的32位数据不包含进位标志，          *)
(*向右旋转D0中存储的值的位数。                                        *)
   DROR_M(X0, D0, dwData1);
```

●对应的MELSEC指令

・DROR(32位数据的右旋转)

5.11.6 32位数据右旋转(包含进位标志) DRCR_M

将指定的BIN32位数据包含进位标志向右旋转n位。

■函数定义 **BOOL** DRCR_M(BOOL EN, ANY16 n, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～31)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN32位数据)

备注) “D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的32位数据包含进位标志          *)
(*向右旋转D0中存储的值的位数。                                      *)
   DRCR_M(X0, D0, dwData1);
```

●对应的MELSEC指令

・DRCR(32位数据的右旋转)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.11.7 32位数据左旋转(不包含进位标志) DROL_M

将指定的BIN32位数据不包含进位标志向左旋转n位。

■函数定义

```
BOOL DROL_M(BOOL EN, ANY16 n, ANY32 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～31)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN32位数据)

备注)“D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的32位数据不包含进位标志          *)
(*向左旋转4位。                                                    *)
   DROL_M(X0, K4, dwData1);
```

●对应的MELSEC指令

・DROL(32位数据的左旋转)

5.11.8 32位数据左旋转(包含进位标志) DRCL_M

将指定的BIN32位数据包含进位标志向左旋转n位。

■函数定义

```
BOOL DRCL_M(BOOL EN, ANY16 n, ANY32 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	旋转次数(0～31)(BIN16位数据)
D	IN/OUT	旋转数据・旋转结果(BIN32位数据)

备注) “D”中指定了位软元件的情况下，按指定位数的数据进行旋转。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dwData1的32位数据包含进位标志          *)
(*向左旋转4位。                                                    *)
   DRCL_M(X0, K4, dwData1);
```

●对应的MELSEC指令

・DRCL(32位数据的左旋转)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.12 移动

5.12.1 n位右移 SFR_M

将指定的BIN16位数据向右移动n位。

■函数定义

BOOL SFR_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	移动次数(0～15)(BIN16位数据)
D	IN/OUT	移动数据・移动结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100的数据向右移动4位。                    *)
   SFR_M(X0, K4, D100);
```

●对应的MELSEC指令

・SFR(16位数据的n位右移)

5.12.2 n位左移 SFL_M

将指定的BIN16位数据向左移动n位。

■函数定义

BOOL SFL_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	移动次数(0～15)(BIN16位数据)
D	IN/OUT	移动数据・移动结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100的数据向左移动4位。                    *)
   SFL_M(X0, K4, D100);
```

●对应的MELSEC指令

・SFL(16位数据的n位左移)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 53

5.12.3 n位数据1位右移 BSFR_M

将从指定软元件算起n点的位数据向右移动1位。

■函数定义

BOOL BSFR_M(BOOL EN, ANY16 n, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	移动软元件的点数(BIN16位数据)
D	IN/OUT	移动数据・移动结果(位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将M100～M104的数据向右移动1位。          *)
  BSFR_M(X0, K5, M100);
```

●对应的MELSEC指令

・BSFR(n位数据的1位右移)

5.12.4 n位数据1位左移 BSFL_M

将从指定软元件算起n点的位数据向左移动1位。

■函数定义

BOOL BSFL_M(BOOL EN, ANY16 n, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	移动软元件的点数(BIN16位数据)
D	IN/OUT	移动数据・移动结果(位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将M100～M104的数据向左移动1位。          *)
  BSFL_M(X0, K5, M100);
```

●对应的MELSEC指令

・BSFL(n位数据的1位左移)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 54

5.12.5 1字右移 DSFR_M

将从指定软元件算起n点的16位数据向右移动1字。

■函数定义 **BOOL** DSFR_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	移动软元件的点数(BIN16位数据)
D	IN/OUT	移动数据・移动结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100～D106的数据向右移动1字。          *)
   DSFR_M(X0,K7, D100);
```

●对应的MELSEC指令

・DSFR(n字数据的1字右移)

5.12.6 1字左移 DSFL_M

将从指定软元件算起n点的16位数据向左移动1字。

■函数定义 **BOOL** DSFL_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	移动软元件的点数(BIN16位数据)
D	OUT	移动数据・移动结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100～D106的数据向左移动1字。          *)
   DSFL_M(X0, K7, D100);
```

●对应的MELSEC指令

・DSFL(n字数据的1字左移)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.13 位处理

5.13.1 字软元件的位设置 BSET_M

对指定的字软元件的第n位进行设置。

■函数定义 **BOOL** BSET_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	进行位设置的位编号(BIN16位数据)
D	IN/OUT	位设置数据・位设置结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对D100的第8位进行设置。                    *)
   BSET_M(X0, K8, D100);
```

●对应的MELSEC指令

・BSET(字软元件的位设置)

5.13.2 字软元件的位复位 BRST_M

对指定的字软元件的第n位进行复位。

■函数定义 **BOOL** BRST_M(BOOL EN, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	进行位复位的位编号(BIN16位数据)
D	IN/OUT	进行位复位的数据・位复位结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对D100的第8位进行复位。                    *)
   BRST_M(X0, K8, D100);
```

●对应的MELSEC指令

・BRST(字软元件的复位)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 56

5.13.3 字软元件的位测试 TEST_MD

将指定的字软元件的指定位置的位状态写入到指定位软元件中。

■函数定义

BOOL TEST_MD(BOOL EN, ANY16 S1, ANY16 S2, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	截取的数据(BIN16位数据)
S2	IN	截取的位的位置(BIN16位数据)
D	OUT	截取数据(位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则按D100的第10位的状态对M0进行ON・OFF。        *)
   TEST_MD(X0, D100, K10, M0);
```

●对应的MELSEC指令

・TEST(位设置)

5.13.4 32位数据的位测试 DTEST_MD

将指定的BIN32位数据的指定位置的位写入到指定位软元件中。

■函数定义

BOOL DTEST_MD(BOOL EN, ANY32 S1, ANY16 S2, BOOL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	截取的数据(BIN32位数据)
S2	IN	截取的位的位置(BIN16位数据)
D	OUT	截取数据(位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData的第10位截取并写入到M0中。        *)
   DTEST_MD(X0, dData, K10, M0);
```

●对应的MELSEC指令

・DTEST(位设置)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.13.5 位软元件批量复位 BKRST_M

将从指定位软元件算起的n点进行复位。

■函数定义

BOOL BKRST_M(BOOL EN, BOOL S1, ANY16 n);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	复位数据的起始(位数据)
n	IN	复位位数(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将M10至D100中存储的点数进行复位。          *)
  BKRST_M(X0, M10, D100);
```

●对应的MELSEC指令

• BKRST(位软元件的批量复位)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 58

5.14 数据处理

5.14.1 数据查找 SER_M

将指定的BIN16位数据作为查找对象，将从指定的BIN16位数据算起的n点进行查找。

■函数定义

BOOL SER_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 n, ANY16(2) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	进行查找的数据(BIN16位数据)		
S2	IN	被进行查找的数据(BIN16位数据)		
n	IN	进行查找的数据数(BIN16位数据)		
D	OUT	查找结果 (ARRAY [0..1] OF ANY16)	D[0]	一致的位置
			D[1]	一致数

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100作为查找对象从D200中查找D300点。      *)
(*将与查找对象一致的个数存储到D[1]中，将从D200的第几点的相对值        *)
(*存储到D[0]中。                                                    *)
   SER_M(X0, D100, D200, D300, D);
```

●对应的MELSEC指令

・SER(16位数据查找)

5.14.2 32位数据查找 DSER_M

将指定的BIN32位数据作为查找对象，从指定的BIN32位数据算起查找2n点。

■函数定义

BOOL DSER_M(BOOL EN, ANY32 S1, ANY32 S2, ANY16 n, ANY16(2) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	进行查找的数据(BIN32位数据)		
S2	IN	被进行查找的数据(BIN32位数据)		
n	IN	进行查找的数据数(BIN16位数据)		
D	OUT	查找结果 (ARRAY [0..1] OF ANY16)	D[0]	一致的位置
			D[1]	一致数

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData1、dData1+1作为查找对象，从dData2   *)
(*算起以32位为单位查找D100中存储的点数。将与查找对象一致的          *)
(*个数存储到ArrayResult [1]中，将从dData2的第几点的相对值存储到      *)
(*ArrayResult[0]中。                                                *)
   DSER_M(X0, dData1, dData2, D100, ArrayResult);
```

●对应的MELSEC指令

・DSER(32位数据查找)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.3 位校验 SUM_M

在指定的BIN16位数据的各个位中对变为1的位数进行计数。

■函数定义 **BOOL** SUM_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	计数数据(BIN16位数据)
D	OUT	计数结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData的各个位中变为1的位数存储到      *)
(*Result中。                                                    *)
   SUM_M(X0, iData, Result);
```

●对应的MELSEC指令

・SUM(16位数据的位校验)

5.14.4 32位数据位校验 DSUM_M

将指定的BIN32位数据的各个位中变为1的位数进行计数。

■函数定义 **BOOL** DSUM_M(BOOL EN, ANY32 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	计数数据(BIN32位数据)
D	OUT	计数结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData的各个位中变为1的位数存储到      *)
(*Result中。                                                    *)
   DSUM_M(X0, dData, Result);
```

●对应的MELSEC指令

・DSUM(32位数据的位校验)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.5 编译 DECO_M

对指定的数据的低位n位进行编译。

■函数定义

BOOL DECO_M(BOOL EN, ANY_SIMPLE S1, ANY16 n, ANY_SIMPLE D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	编译数据
n	IN	有效位长(1～8) ※0：无处理(BIN16位数据)
D	OUT	编译结果

备注) 自变量“S1”、“D”中不能使用DINT/REAL/STRING型。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对D100的低位BitSize位进行编译，并将编译结果 *)
(*存储到从Result算起的2^BitSize位中。 *)
  DECO_M(X0, D100, BitSize, Result);
```

●对应的MELSEC指令

• DECO(8→256位编译)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.6 编码 ENCO_M

对从指定的数据算起2n位的数据进行编码。

■函数定义

BOOL ENCO_M(BOOL EN, ANY_SIMPLE S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	编码数据
n	IN	有效位长(1～8) ※0：无处理(BIN16位数据)
D	OUT	编码结果

备注) 自变量“S1”中不能使用DINT/REAL/STRING型。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对从D100算起的2^BitSize位进行编码 *)
(*并将结构存储到Result中。 *)
  ENCO_M(X0, D100, BitSize, Result);
```

●对应的MELSEC指令

• ENCO(256→8位编码)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.7 7段编译 SEG_M

将指定数据的低4位(0～F)编译为7段显示数据。

■函数定义

BOOL SEG_M(BOOL EN, ANY16 S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	编译数据
D	OUT	编译结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100的低4位编译为7段显示数据          *)
(*并存储到Result中。                                          *)
   SEG_M(X0, D100, Result);
```

●对应的MELSEC指令

・SEG(7段编译)

5.14.8 16位数据的4位分离 DIS_M

对指定的BIN16位数据的低位n位数的数据进行分离，并存储到从指定的软元件算起n点的低4位中。

■函数定义

BOOL DIS_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	分离的数据(BIN16位数据)
n	IN	分离数(1～4) ※0：无处理(BIN16位数据)
D	OUT	分离结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100的低位D200位数(1位数为4位)的数据   *)
(*存储到从Result算起D200点的低4位中。                          *)
   DIS_M(X0, D100, D200, Result);
```

●对应的MELSEC指令

・DIS(16位数据的4位分离)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.9 16位数据的4位合并 UNI_M

将从指定的软元件算起的n点的BIN16位数据的低4位合并到指定的软元件中。

■函数定义

BOOL UNI_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	合并的数据(BIN16位数据)
n	IN	合并数(1～4) ※0：无处理(BIN16位数据)
D	OUT	合并结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从D100算起的3点的16位数据的低4位          *)
(*合并到Result中。                                                  *)
   UNI_M(X0, D100, K3, Result);
```

●对应的MELSEC指令

・UNI(4位数据的16位合并)

5.14.10 任意数据的位分离 NDIS_M

将指定的软元件以后存储的数据的各个位按指定的位数逐个进行分离。

■函数定义

BOOL NDIS_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	分离的数据(BIN16位数据)
S2	IN	分离单位(分离的位数) (BIN16位数据)
D	OUT	分离结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData1以后存储的数据的各个位          *)
(*按iData2的位数逐个进行分离并存储到Result以后。                  *)
   NDIS_M(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・NDIS(任意位数据的分离)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.11 任意数据的位合并 NUNI_M

将指定的软元件以后存储的数据的各位按指定的位进行合并。

■函数定义

BOOL NUNI_M(BOOL EN, ANY16 S1, ANY16 S2, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	合并的数据(BIN16位数据)
S2	IN	合并单位(合并的位数)(BIN16位数据)
D	OUT	合并结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData1以后存储的数据的各个位          *)
(*按iData2指定的位进行合并并存储到Result以后。                    *)
   NUNI_M(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・NUNI(任意位数据的合并)

5.14.12 字节单位数据分离 WTOB_MD

将指定的软元件以后存储的BIN16位数据分离为n字节。

■函数定义

BOOL WTOB_MD(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	按字节单位分离的数据(BIN16位数据)
n	IN	分离的字节数据的个数(BIN16位数据)
D	OUT	分离结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData1以后存储的16位数据          *)
(*分离为iData2字节并存储到Result以后。                        *)
   WTOB_MD(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・WTOB(字节单位数据的分离)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.13 字节单位数据合并 BTOW_MD

将指定的软元件以后的n点的BIN16位数据的低8位合并为字单位。

■函数定义

BOOL BTOW_MD(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	以字节单位合并的数据(BIN16位数据)
n	IN	合并的字节数据的个数(BIN16位数据)
D	OUT	合并结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData1以后的iData2字的16位数据的          *)
(*低8位合并为字单位，并存储到Result以后。                            *)
   BTOW_MD(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・BTOW(字节单位数据的合并)

5.14.14 数据最大值查找 MAX_M

从指定的软元件算起n点数的BIN16位数据中查找最大值。

■函数定义

BOOL MAX_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行查找的数据的起始(BIN16位数据)
n	IN	进行查找的数据数(BIN16位数据)
D	OUT	最大值的查找结果(BIN16位数据)

备注：定时器的设置值中指定常数的情况下，只能指定10进制数。
定时器的设置值指定范围为0～32767。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从iData1以后iData2点的16位BIN数据          *)
(*中查找最大值并存储到Result中。                                     *)
   MAX_M(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・MAX(16位数据最大值查找)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.15 32位数据最大值查找 DMAX_M

从指定的软元件算起n点数的BIN32位数据中查找最大值。

■函数定义

BOOL DMAX_M(BOOL EN, ANY32 S1, ANY16 n, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行查找的数据的起始(BIN32位数据)
n	IN	进行查找的数据数(BIN16位数据)
D	OUT	最大值的查找结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从dData以后的iData点的32位BIN数据中        *)
(*查找最大值并存储到Result中。                                        *)
   DMAX_M(X0, dData, iData, Result);
```

●对应的MELSEC指令

・DMAX(32位数据最大值查找)

5.14.16 数据最小值查找 MIN_M

从指定的软元件算起的n点数的BIN16位数据中查找最小值。

■函数定义

BOOL MIN_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行查找的数据的起始(BIN16位数据)
n	IN	进行查找的数据数(BIN16位数据)
D	OUT	最小值的查找结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从iData1以后算起的iData2点的16位BIN数据中 *)
(*查找最小值并存储到Result中。                                        *)
   MIN_M(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・MIN(16位数据最小值查找)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.17 32位数据最小值查找 DMIN_M

从指定的软元件算起的n点数的BIN32位数据中查找最小值。

■函数定义

BOOL DMIN_M(BOOL EN, ANY32 S1, ANY16 n, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行查找的数据的起始(BIN32位数据)
n	IN	进行查找的数据数(BIN16位数据)
D	OUT	最小值的查找结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从dData以后算起的iData点的32位BIN数据中    *)
(*查找最小值并存储到Result、Result+1中。                              *)
   DMIN_M(X0, dData, iData, Result);
```

●对应的MELSEC指令

・DMIN(32位数据最小值查找)

5.14.18 数据排序 SORT_M

将从指定的软元件算起的n点数的BIN16位数据进行升序・降序的排序。

■函数定义

BOOL SORT_M(BOOL EN, ANY16 S1, ANY16 n, ANY16 S2, BOOL D1, ANY16 D2);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	排序的数据的起始(BIN16位数据)
n	IN	排序的数据数(BIN16位数据)
S2	IN	执行1次后进行比较的数据数(BIN16位数据)
D1	OUT	排序完毕后使其ON的位软元件(位数据)
D2	OUT	系统使用的软元件(BIN16位数据)

备注) 排序是通过SM703的ON・OFF进行指定。SM703 OFF时：升序；SM703 ON时：降序

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从iData1算起的iData2点的BIN16位数据    *)
(*进行升序・降序的排序。                                            *)
   SORT_M(X0, iData1, iData2, iData3, bData, iData4);
```

●对应的MELSEC指令

・SORT(16位数据排序)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.19 32位数据排序 DSORT_M

将从指定的软元件算起n点数的BIN32位数据进行升序・降序的排序。

■函数定义

BOOL DSORT_M(BOOL EN, ANY32 S1, ANY16 n, ANY16 S2, BOOL D1, ANY16 D2);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	排序的数据的起始(BIN32位数据)
n	IN	排序的数据数(BIN16位数据)
S2	IN	执行1次后进行比较的数据数(BIN16位数据)
D1	OUT	排序完毕后使其ON的位软元件(位数据)
D2	OUT	系统使用的软元件(BIN16位数据)

备注) 排序是通过SM703的ON・OFF进行指定。SM703 OFF时：升序；SM703 ON时：降序

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从dData算起的iData1点的BIN32位数据      *)
(*进行升序・降序的排序。                                            *)
   DSORT_M(X0, dData, iData1, iData2, bData, iData3);
```

●对应的MELSEC指令

・DSORT(32位数据排序)

5.14.20 合计值计算 WSUM_M

将从指定的软元件算起n点数的BIN16位数据进行加法运算。

■函数定义

BOOL WSUM_M(BOOL EN, ANY16 S1, ANY16 n, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行合计值计算的数据(BIN16位数据)
n	IN	数据数(BIN16位数据)
D	OUT	合计值存储目标(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从iData1算起的iData2点的16位BIN数据进行  *)
(*加法运算并将结果存储到Result中。                                    *)
   WSUM_M(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・WSUM(16位合计值计算)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.14.21 32位数据合计值计算 DWSUM_M

将从指定的软元件算起n点数的BIN32位数据进行加法运算。

■函数定义

```
BOOL DWSUM_M(BOOL EN, ANY32 S1, ANY16 n, ANY16(4) D);
```

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	进行合计值计算的数据(BIN32位数据)		
n	IN	数据数(BIN16位数据)		
D	OUT	合计值存储目标 (ARRAY[0..3] OF ANY16)	D[0] D[1] D[2] D[3]	高4位 ↓ 低4位

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从dData算起的iData点的32位BIN数据        *)
(*进行加法运算并将结果存储到Result中。                              *)
   DWSUM_M(X0, dData, iData, Result);
```

●对应的MELSEC指令

• DWSUM(32位合计值计算)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 69

5.15 结构化

5.15.1 刷新 COM_M

进行智能功能模块的I/O刷新及一般数据的处理。

■函数定义 **BOOL** COM_M(BOOL EN);

自变量名	IN/OUT	内容
EN	IN	执行条件 (只能指定表示常时有效的值TRUE或常时ON软元件SM400。)

返回值	内容
BOOL	执行条件

●使用示例

```
(*SM755 OFF时：进行智能功能模块的自动刷新及一般数据的处理；          *)
(*SM755 ON时 ：仅进行一般数据的处理。                              *)
  COM_M(TRUE);
```

●对应的MELSEC指令

・COM(刷新指令)

5.16 缓冲存储器访问

5.16.1 智能功能模块1字数据读取 FROM_M

从指定的智能功能模块・特殊功能模块内的缓冲存储器的指定的地址对指定点数的数据进行读取。

■函数定义

```
BOOL FROM_M(BOOL EN, ANY16 n1, ANY16 n2, ANY16 n3, ANY16 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n1	IN	智能功能模块・特殊功能模块的起始输入编号(BIN16位数据)
n2	IN	进行读取的数据的起始地址(BIN16位数据)
n3	IN	读取的数据数(BIN16位数据)
D	OUT	读取的数据(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从输入输出编号040～05F中安装的智能功能        *)
(*模块的缓冲存储器的地址10读取1字到D0中。                              *)
   FROM_M(X0, H4, K10, K1, D0);
```

●对应的MELSEC指令

・FROM(从智能功能模块进行1字数据读取)

5.16.2 智能功能模块2字数据读取 DFRO_M

从指定的智能功能模块・特殊功能模块内缓冲存储器的指定的地址读取指定点数×2的数据。

■函数定义

```
BOOL DFRO_M(BOOL EN, ANY16 n1, ANY16 n2, ANY16 n3, ANY32 D);
```

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n1	IN	智能功能模块・特殊功能模块的起始输入编号(BIN16位数据)
n2	IN	进行读取的数据的起始地址(BIN16位数据)
n3	IN	读取的数据数(BIN16位数据)
D	OUT	读取的数据(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从输入输出编号040～05F中安装的智能        *)
(*功能模块的缓冲存储器的地址602,603中读取2字数据至DwResult。          *)
   DFRO_M(X0, H4, K602, K1, DwResult);
```

●对应的MELSEC指令

・DFRO(从智能功能模块进行2字数据读取)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.16.3 智能功能模块1字数据写入 TO_M

将从指定的软元件算起n3点的数据写入到指定的智能功能模块・特殊功能模块内缓冲存储器的指定的地址以后。

■函数定义

BOOL TO_M(BOOL EN, ANY16 S1, ANY16 n1, ANY16 n2, ANY16 n3);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	写入的数据(BIN16位数据)
n1	IN	智能功能模块・特殊功能模块的起始输入编号(BIN16位数据)
n2	IN	进行数据写入的起始地址(BIN16位数据)
n3	IN	写入数据数(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将3写入到输入输出编号040～05F中安装的智能 *)
(*功能模块的缓冲存储器的地址0中。                                    *)
   TO_M(X0, K3, H4, K0, K1);
```

●对应的MELSEC指令

・TO(至智能功能模块的1字数据写入)

5.16.4 智能功能模块2字数据写入 DTO_M

将从指定的软元件算起n3×2点的数据写入到指定智能功能模块・特殊功能模块内的缓冲存储器的指定地址以后。

■函数定义

BOOL DTO_M(BOOL EN, ANY32 S1, ANY16 n1, ANY16 n2, ANY16 n3);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	写入的数据(BIN32位数据)
n1	IN	智能功能模块・特殊功能模块的起始输入编号(BIN16位数据)
n2	IN	进行(n3×2)点数据写入的起始地址(BIN16位数据)
n3	IN	写入的数据数(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将0写入到输入输出编号040～05F中安装的智能 *)
(*功能模块的缓冲存储器地址41,42中。                                  *)
   DTO_M(X0, K0, H4, K41, K1);
```

●对应的MELSEC指令

・DTO(至智能功能模块的2字数据写入)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17 字符串处理

5.17.1 BIN→10进制ASCII转换 BINDA_S_MD

将指定的BIN16位数据的10进制数显示的各位数的数值分别转换为ASCII码数据。

■函数定义 **BOOL** BINDA_S_MD(BOOL EN, ANY16 S1, STRING(8) D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BIN16位数据)
D	OUT	转换结果(10进制ASCII码数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData中存储的BIN数据以10进制数显示时      *)
(*的各个位的数值分别转换为ASCII码，并将结果存储到sData中。          *)
   BINDA_S_MD(X0, iData, sData);
```

●对应的MELSEC指令

・BINDA(BIN16位→10进制ASCII转换)

5.17.2 32位BIN→10进制ASCII转换 DBINDA_S_MD

将指定的BIN32位数据的10进制数显示的各位数的数值分别转换为ASCII码数据。

■函数定义 **BOOL** DBINDA_S_MD(BOOL EN, ANY32 S1, STRING(12) D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BIN32位数据)
D	OUT	转换结果(10进制ASCII码数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData中存储的BIN数据以10进制数显示时      *)
(*的各个位的数值分别转换为ASCII码，并将结果存储到sData中。          *)
   DBINDA_S_MD(X0, dData, sData);
```

●对应的MELSEC指令

・DBINDA(BIN32位→10进制ASCII转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.3 BIN→16进制ASCII转换 BINHA_S_MD

将指定的BIN16位数据的16进制数显示的各位数的数值分别转换为ASCII码数据。

■函数定义

BOOL BINHA_S_MD(BOOL EN, ANY16 S1, STRING(6) D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BIN16位数据)
D	OUT	转换结果(16进制ASCII码数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData中存储的BIN数据以16进制数显示时      *)
(*的各个位的数值分别转换为ASCII码，并将结果存储到sData中。          *)
   BINHA_S_MD(X0, iData, sData);
```

●对应的MELSEC指令

・BINHA(BIN16位→16进制ASCII转换)

5.17.4 32位BIN→16进制ASCII转换 DBINHA_S_MD

将指定的BIN32位数据的16进制数显示的各位数的数值分别转换为ASCII码数据。

■函数定义

BOOL DBINHA_S_MD(BOOL EN, ANY32 S1, STRING(10) D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BIN32位数据)
D	OUT	转换结果(16进制ASCII码数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData中存储的BIN数据以16进制数显示时      *)
(*的各个位的数值分别转换为ASCII码，并将结果存储到sData中。          *)
   DBINHA_S_MD(X0, dData, sData);
```

●对应的MELSEC指令

・DBINHA(BIN32位→16进制ASCII转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.5 BCD4位→10进制ASCII转换 BCDDA_S_MD

将指定的BCD4位数据的各位数的数值分别转换为ASCII码。

■函数定义 **BOOL** BCDDA_S_MD(BOOL EN, ANY16 S1, STRING(6) D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BCD4位数据)
D	OUT	转换结果(10进制ASCII码数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData中存储的BCD数据以10进制数显示时      *)
(*的各个位的数值分别转换为ASCII码，并将结果存储到sData中。          *)
   BCDDA_S_MD(X0, iData, sData);
```

●对应的MELSEC指令

・BCDDA(BCD4位→10进制ASCII转换)

5.17.6 BCD8位→10进制ASCII转换 DBCDDA_S_MD

将指定的BCD8位数据的各位数的数值分别转换为ASCII码。

■函数定义 **BOOL** DBCDDA_S_MD(BOOL EN, ANY32 S1, STRING(10) D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BCD8位数据)
D	OUT	转换结果(10进制ASCII码数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData中存储的BCD数据以10进制数显示时      *)
(*的各个位的数值分别转换为ASCII码，并将结果存储到sData中。          *)
   DBCDDA_S_MD(X0, dData, sData);
```

●对应的MELSEC指令

・DBCDDA(BCD8位→10进制ASCII转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.7 10进制ASCII→BIN转换 DABIN_S_MD

将指定的10进制ASCII码数据转换为BIN16位数据。

■函数定义 **BOOL** DABIN_S_MD(BOOL EN, STRING(6) S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(10进制ASCII码数据)
D	OUT	转换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中存储的10进制ASCII数据          *)
(*转换为BIN16位数据，并将结果存储到iData中。                      *)
   DABIN_S_MD(X0, sData, iData);
```

●对应的MELSEC指令

• DABIN(10进制ASCII→BIN16位转换)

5.17.8 10进制ASCII→32位BIN转换 DDABIN_S_MD

将指定的10进制ASCII码数据转换为BIN32位数据。

■函数定义 **BOOL** DDABIN_S_MD(BOOL EN, STRING(11) S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(10进制ASCII码数据)
D	OUT	转换结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中存储的10进制ASCII数据          *)
(*转换为BIN32位数据，并将结果存储到dData中。                      *)
   DDABIN_S_MD(X0, sData, dData);
```

●对应的MELSEC指令

• DDABIN(10进制ASCII→32位转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.9 16进制ASCII→BIN转换 HABIN_S_MD

将指定的16进制ASCII码数据转换为BIN16位数据。

■函数定义 **BOOL** HABIN_S_MD(BOOL EN, STRING(4) S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(16进制ASCII数据)
D	OUT	转换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中存储的16进制ASCII数据转换为BIN16位  *)
(*数据，并将结果存储到iData中。                                          *)
   HABIN_S_MD(X0, sData, iData);
```

●对应的MELSEC指令

・HABIN(16进制ASCII→BIN16位转换)

5.17.10 16进制ASCII→32位BIN转换 DHABIN_S_MD

将指定的16进制ASCII码数据转换为BIN32位数据。

■函数定义 **BOOL** DHABIN_S_MD(BOOL EN, STRING(8) S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(16进制ASCII数据)
D	OUT	转换结果(BIN32位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中存储的16进制ASCII数据转换为BIN32位  *)
(*数据，并将结果存储到dData中。                                          *)
   DHABIN_S_MD(X0, sData, dData);
```

●对应的MELSEC指令

・DHABIN(16进制ASCII→32位BIN转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.11 10进制ASCII→BCD4位转换 DABCD_S_MD

将指定的10进制ASCII码数据转换为BCD4位数据。

■函数定义 **BOOL** DABCD_S_MD(BOOL EN, STRING(4) S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(10进制ASCII码数据)
D	OUT	转换结果(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中存储的10进制ASCII数据          *)
(*转换为BCD4位数据，并将结果存储到iData中。                      *)
   DABCD_S_MD( X0, sData, iData );
```

●对应的MELSEC指令

• DABCD(10进制ASCII→BCD4位转换)

5.17.12 10进制ASCII→BCD8位转换 DDABCD_S_MD

将指定的10进制ASCII码数据转换为BCD8位数据。

■函数定义 **BOOL** DDABCD_S_MD(BOOL EN, STRING(8) S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(10进制ASCII码数据)
D	OUT	转换结果(BCD8位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中存储的10进制ASCII数据          *)
(*转换为BCD8位数据，并将结果存储到dData中。                      *)
   DDABCD_S_MD(X0, sData, dData);
```

●对应的MELSEC指令

• DDABCD(10进制ASCII→BCD8位转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.13 软元件的注释数据读取 COMRD_S_MD

将指定的软元件的注释以ASCII码数据进行读取。

■函数定义 **BOOL** COMRD_S_MD(BOOL EN, ANY_SIMPLE S1, STRING(32) D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行注释读取的数据
D	OUT	进行注释读取的结果(ASCII码数据)

备注) 自变量“S1”中不能使用DINT/REAL/STRING型。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D100中设置的注释进行读取，并以ASCII码    *)
(*存储到sData中。                                                    *)
   COMRD_S_MD(X0, D100, sData);
```

●对应的MELSEC指令

・COMRD(软元件的注释数据读取)

5.17.14 字符串的长度检测 LEN_S_MD

求出指定的字符串的长度。

■函数定义 **BOOL** LEN_S_MD(BOOL EN, STRING S1, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行字符串长度检测的数据(字符串数据)
D	OUT	检测结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对sData中指定的字符串的长度进行检测并将    *)
(*结果存储到iData中。                                                *)
   LEN_S_MD(X0, sData, iData);
```

●对应的MELSEC指令

・LEN(字符串的长度检测)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.15 BIN→字符串转换 STR_S_MD

在指定的BIN16位数据的指定的位置处附加小数点后转换为字符串。

■函数定义

BOOL STR_S_MD(BOOL EN, ANY32 S1, ANY16 S2, STRING(9) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	转换数值的位数 (BIN32位数据)	S1	全部位数(2～8位)
			S1+1	小数部分位数 (0～5位)
S2	IN	转换的数据(BIN16位数据)		
D	OUT	转换结果(字符串数据)		

备注) 在“S1”中不能进行位软元件的位数指定。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData中指定的BIN16位数据在dData中指定的    *)
(*位置处附加小数点后转换为字符串，并将结果存储到sData中。              *)
  STR_S_MD(X0, dData, iData, sData);
```

●对应的MELSEC指令

• STR(BIN16位→字符串转换)

5.17.16 32位BIN→字符串转换 DSTR_S_MD

在指定的BIN32位数据的指定的位置处附加小数点后转换为字符串。

■函数定义

BOOL DSTR_S_MD(BOOL EN, ANY32 S1, ANY32 S2, STRING(14) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	进行转换的数值的位数 (BIN32位数据)	S1	全部位数(2～8位)
			S1+1	小数部分位数 (0～5位)
S2	IN	转换的数据(BIN32位数据)		
D	OUT	转换结果(字符串数据)		

备注) “S1”中不能进行位软元件的位数指定。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData1中指定的BIN32位数据在dData2中指定的   *)
(*位置处附加小数点后转换为字符串，并将结果存储到sData中。              *)
  DSTR_S_MD(X0, dData1, dData2, sData);
```

●对应的MELSEC指令

• DSTR(BIN32位→字符串转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.17 字符串→BIN转换 VAL_S_MD

将指定的字符串转换为BIN16位数据，并获取其位数及BIN16位数据。

■函数定义

BOOL VAL_S_MD(BOOL EN, STRING(8) S1, ANY32 D1, ANY16 D2);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(字符串数据) 备注) S1中指定的字符串内变为小数部分的字符数为0～5字符。但是，应指定为(全部位数-3)以下。
D1	OUT	转换结果(位数)(BIN32位数据)
D2	OUT	转换结果(BIN16位数据)

备注) “D1”中不能进行位软元件的位数指定。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中指定的字符串转换为BIN16位数据，      *)
(*并将位数存储到dData中，将BIN数据存储到iData中。                      *)
  VAL_S_MD(X0, sData, dData, iData);
```

●对应的MELSEC指令

• VAL(字符串→BIN16位转换)

5.17.18 字符串→32位BIN转换 DVAL_S_MD

将指定的字符串转换为BIN32位数据，并获取其位数及BIN32位数据。
将获取结果存储到指定的软元件中。

■函数定义

BOOL DVAL_S_MD(BOOL EN, STRING(13) S1, ANY32 D1, ANY32 D2);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(字符串数据) 备注) S1中指定的字符串内变为小数部分的字符数为0～5字符。但是，应指定为(全部位数-3)以下。
D1	OUT	转换结果(位数)(BIN32位数据)
D2	OUT	转换结果(BIN32位数据)

备注) “D1”中不能进行位软元件的位数指定。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中指定的字符串转换为BIN32位数据，     *)
(*并将位数存储到dData1中，将BIN数据存储到dData2中。                     *)
  DVAL_S_MD(X0, sData, dData1, dData2);
```

●对应的MELSEC指令

• DVAL(字符串→BIN32位转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.19 浮动小数点→字符串转换 ESTR_M

将指定的实数数据按照指定的显示指示转换为字符串。

■函数定义

BOOL ESTR_M(BOOL EN, REAL S1, ANY16(3) S2, STRING(24) D);

自变量名	IN/OUT	内容	
EN	IN	执行条件(仅TRUE时执行函数)	
S1	IN	转换的数据(实数数据)	
S2	IN	转换数值的显示指定(ARRAY [0..2] OF ANY16)	
		S2[0]	显示格式(0: 小数点格式；1: 指数格式)
		S2[1]	全部位数(2～24位) 小数部分位数为“0”时……位数(最大: 24)≧ 2 小数部分位数为“0”以外时…位数(最大: 24)≧(小数部分位数+3)
		S2[2]	小数部位数(0～7位)
D	OUT	转换结果(字符串数据)	

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData中指定的实数数据按照ArrayData中      *)
(*指定的显示指示转换为字符串，并将结果存储到sData中。                  *)
   ESTR_M(X0, rData, ArrayData, sData);
```

●对应的MELSEC指令

・ESTR(浮动小数点→字符串转换)

5.17.20 字符串→浮动小数点转换 EVAL_M

将指定的字符串数据转换为实数数据。

■函数定义

BOOL EVAL_M(BOOL EN, STRING(24) S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(字符串数据)
D	OUT	转换结果(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中指定的字符串转换为实数数据，      *)
(*并将结果存储到rData中。                                          *)
   EVAL_M(X0, sData, rData);
```

●对应的MELSEC指令

・EVAL(字符串→浮动小数点转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.21 BIN→ASCII转换 ASC_S_MD

将指定的BIN16位数据通过16进制数处理转换为指定字符数的ASCII数据。

■函数定义

BOOL ASC_S_MD(BOOL EN, ANY16 S1, ANY16 n, STRING D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BIN16位数据)
n	IN	存储的字符数(BIN16位数据)
D	OUT	转换结果(ASCII数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData1中指定的BIN16位数据通过16进制数处 *)
(*理转换为ASCII，并按iData2中指定的字符数范围存储到sData中指定的软元 *)
(*件编号以后。                                                    *)
   ASC_S_MD(X0, iData1, iData2, sData);
```

●对应的MELSEC指令

・ASC(BIN16进制数据→ASCII转换)

5.17.22 ASCII→BIN转换 HEX_S_MD

将指定的字符数中存储的16进制ASCII数据转换为BIN16位数据。

■函数定义

BOOL HEX_S_MD(BOOL EN, STRING S1, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(16进制ASCII数据)
n	IN	转换的字符数(BIN16位数据)
D	OUT	转换结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData中指定的软元件编号以后的iData1中指 *)
(*定的字符数中存储的16进制数ASCII数据转换为BIN值并将结果存储到iData2中。 *)
   HEX_S_MD(X0, sData, iData1, iData2);
```

●对应的MELSEC指令

・HEX(ASCII→BIN16进制转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.23 从字符串右侧截取 RIGHT_M

从指定的字符串数据的右侧(字符串的最终)获取n字符的数据。

■函数定义

BOOL RIGHT_M(BOOL EN, STRING S1, ANY16 n, STRING D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	获取的数据(字符串数据)
n	IN	获取的字符数(BIN16位数据)
D	OUT	获取结果(n字符的字符串数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData中指定的字符串的右侧(字符串的最终)    *)
(*将iData字符的数据存储到Result中。                                    *)
   RIGHT_M(X0, sData, iData, Result);
```

●对应的MELSEC指令

・RIGHT(从字符串的右侧截取)

5.17.24 从字符串左侧截取 LEFT_M

从指定的字符串数据的左侧(字符串的起始)获取n字符的数据。

■函数定义

BOOL LEFT_M(BOOL EN, STRING S1, ANY16 n, STRING D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	获取的数据(字符串数据)
n	IN	获取的字符数(BIN16位数据)
D	OUT	获取结果(n字符的字符串数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData中指定的字符串的左侧(字符串的起始)    *)
(*将iData字符的数据存储到Result中。                                    *)
   LEFT_M(X0, sData, iData, Result);
```

●对应的MELSEC指令

・LEFT(从字符串的左侧截取)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.25 字符串中的任意截取 MIDR_M

从指定的字符串数据的S2[0]处开始，获取S2[1]字符数的数据。

■函数定义

BOOL MIDR_M(BOOL EN, STRING S1, ANY16(2) S2, STRING D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	获取的数据(字符串数据)		
S2	IN	起始字符的位置以及获取的字符数的存储目标(ARRAY [0..1] OF ANY16)	S2[0]	起始字符的位置
			S2[1]	获取的字符数
D	OUT	获取结果(字符串数据)		

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则在sData中指定的字符串的左侧(字符串的起始) *)
(*从StrArray [0]中指定的位置开始将StrArray [1]中指定的字符数 *)
(*的数据存储到Result中。 *)
   MIDR_M(X0, sData, StrArray, Result);
```

●对应的MELSEC指令

・MIDR(字符串中的任意截取)

5.17.26 字符串中的任意替换 MIDW_M

从指定的字符串数据的S2[0]开始存储S2[1]中指定的字符数的数据。

■函数定义

BOOL MIDW_M(BOOL EN, STRING S1, ANY16(2) S2, STRING D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	获取的数据(字符串数据)		
S2	IN	起始字符的位置以及获取的字符数的存储目标(ARRAY [0..1] OF ANY16)	S2[0]	替换目标的起始字符的位置
			S2[1]	获取的字符数
D	IN/OUT	替换的数据・替换结果(字符串数据)		

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将从sData1中指定的字符串的左侧 *)
(*(字符串的起始)算起的StrArray[1]中指定的字符数的数据，存储到 *)
(*从sData2中存储的字符串数据左侧算起的StrArray [0]中指定的位置以后。 *)
   MIDW_M(X0, sData1, StrArray, sData2);
```

●对应的MELSEC指令

・MIDW(字符串中的任意替换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.27 字符串查找 INSTR_M

从指定的字符串数据的左侧第n个字符开始查找指定的字符串数据。

■函数定义

BOOL INSTR_M(BOOL EN, STRING S1, STRING S2, ANY16 n, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行查找的数据(字符串数据)
S2	IN	被进行查找的数据(字符串数据)
n	IN	查找开始位置(从左侧第n个字符开始)(BIN16位数据)
D	OUT	查找结果(S2中指定的字符串数据的起始算起的第几个字符)(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData2中指定的字符串的左侧(字符串的起始)*)
(*第iData个字符开始查找sData1中指定的字符串，并将查找结果存储到      *)
(*Result中。                                                    *)
   INSTR_M(X0, sData1, sData2, iData, Result);
```

●对应的MELSEC指令

• INSTR(字符串查找)

5.17.28 浮动小数点→BCD分解 EMOD_M

将指定的实数数据按照指定的小数部分位数分解为BCD型浮动小数点格式。

■函数定义

BOOL EMOD_M(BOOL EN, REAL S1, ANY16 S2, ANY16(5) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	分解的数据(实数数据)		
S2	IN	小数部分位数数据(BIN16位数据)		
D	OUT	BCD分解的数据的存储目标(ARRAY[0..4] OF ANY16)	D[0]	符号(正: 0；负: 1)
			D[1]	BCD7位
			D[2]	
			D[3]	指数部分符号(正: 0；负: 1)
			D[4]	BCD指数

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData中指定的实数数据按照iData中指定的  *)
(*小数部分位数分解为BCD型浮动小数点格式，并将结果存储到Result中。    *)
   EMOD_M(X0, rData, iData, Result);
```

●对应的MELSEC指令

• EMOD(浮动小数点数据→BCD分解)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.17.29 BCD格式数据→浮动小数点 EREXP_M

将指定的BCD型浮动小数点格式数据按照指定的小数部分位数转换为实数数据。

■函数定义

BOOL EREXP_M(BOOL EN, ANY16 S1, ANY16 S2, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据(BCD型浮动小数点格式数据)
S2	IN	小数部分位数数据(BIN16位数据)
D	OUT	转换结果(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData1中指定的BCD型浮动小数点格式数据          *)
(*按照iData2中指定的小数部分位数转换为实数数据，并将结果存储到Result中。*)
   EREXP_M(X0, iData1, iData2, Result);
```

●对应的MELSEC指令

・EREXP(BCD格式数据→浮动小数点)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 87

5.18 特殊函数

5.18.1 浮动小数点SIN运算 SIN_E_MD

进行指定角度的SIN(正弦)值运算。

■函数定义 **BOOL** SIN_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行SIN(正弦)运算的角度数据(实数数据) 备注) 指定的角度是以弧度单位(角度×π/180)进行设置。
D	OUT	运算结果(SIN值)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中指定的角度的SIN值进行计算，      *)
(*并将结果存储到Result中。                                        *)
   SIN_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・SIN(浮动小数点SIN运算(单精度))

5.18.2 浮动小数点COS运算 COS_E_MD

进行指定的角度的COS(余弦)值运算。

■函数定义 **BOOL** COS_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行COS(余弦)运算的角度数据(实数数据) 备注) 指定的角度是以弧度单位(角度×π/180)进行设置。
D	OUT	运算结果(COS值)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中指定的角度的COS值进行计算，      *)
(*并将结果存储到Result中。                                        *)
   COS_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・COS(浮动小数点COS运算(单精度))

5.18.3 浮动小数点TAN运算 TAN_E_MD

进行指定角度的TAN(正切)值运算。

■函数定义

BOOL TAN_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行TAN(正切)运算的角度数据(实数数据) 备注) 指定的角度是以弧度单位(角度×π/180)进行设置。
D	OUT	运算结果(TAN值)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中指定的角度的TAN值进行计算，      *)
(*并将结果存储到Result中。                                        *)
   TAN_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・TAN(浮动小数点TAN运算(单精度))

5.18.4 浮动小数点SIN^{-1}运算 ASIN_E_MD

进行指定的SIN值的SIN^{-1}(反正弦)运算。

■函数定义

BOOL ASIN_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	运算数据SIN值(-1.0～1.0)(实数数据)
D	OUT	运算结果(弧度单位的角度数据)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从rData中指定的SIN值进行角度运算      *)
(*并将结果存储到Result中。                                      *)
   ASIN_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・ASIN(浮动小数点SIN^{-1}运算(单精度))

5.18.5 浮动小数点COS^{-1}运算 ACOS_E_MD

进行指定COS值的COS-1(反余弦)运算。

■函数定义

BOOL ACOS_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	运算数据COS值(-1.0～1.0)(实数数据)
D	OUT	运算结果(弧度单位的角度数据)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从rData中指定的COS值进行角度运算          *)
(*并将结果存储到Result中。                                        *)
   ACOS_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・ACOS(浮动小数点COS^{-1}运算(单精度))

5.18.6 浮动小数点TAN^{-1}运算 ATAN_E_MD

进行指定的TAN值的TAN^{-1}(反正切)运算。

■函数定义

BOOL ATAN_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	运算数据TAN值(实数数据)
D	OUT	运算结果(弧度单位的角度数据)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从rData中指定的TAN值进行角度运算          *)
(*并将结果存储到Result中                                          *)
   ATAN_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・ATAN(浮动小数点TAN^{-1}运算(单精度))

5.18.7 浮动小数点角度→弧度 RAD_E_MD

将指定角度的大小单位从度单位转换为弧度单位。

■函数定义

BOOL RAD_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换的数据 度单位的角度数据(实数数据)
D	OUT	转换结果(弧度单位)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData中指定的度单位的角度数据转换为      *)
(*弧度单位，并将结果存储到Result中。                                  *)
   RAD_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・RAD(浮动小数点角度→弧度(单精度))

5.18.8 浮动小数点弧度→角度转换 DEG_E_MD

将指定角度的大小单位从弧度单位转换为度单位。

■函数定义

BOOL DEG_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据 弧度值数据(实数数据)
D	OUT	转换结果(度单位)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将角度的大小单位从rData中指定的弧度单位      *)
(*转换为度单位，并将结果存储到Result中。                              *)
   DEG_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

・DEG(浮动小数点弧度→角度(单精度))

5.18.9 浮动小数点平方根 SQR_E_MD

进行指定值的平方根运算。

■函数定义 **BOOL** SQR_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	运算数据(只能指定正数)(实数数据)
D	OUT	运算结果(实数数据)

备注) “S1”中指定的值只能为正数。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中指定的值进行平方根运算，          *)
(*并将结果存储到Result中。                                          *)
   SQR_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

• SQR(浮动小数点平方根(单精度))

5.18.10 浮动小数点指数运算 EXP_E_MD

对指定的值进行以e为底的自然指数运算。

■函数定义 **BOOL** EXP_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	运算的指数部分数据(实数数据)
D	OUT	运算结果(e^{S1})(实数数据)

备注) 将底(e)以“2.71828”进行运算。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData作为指数进行自然指数运算，          *)
(*并将结果存储到Result中。                                          *)
   EXP_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

• EXP(浮动小数点指数运算(单精度))

5.18.11 浮动小数点自然对数运算 LOG_E_MD

对指定的值进行以e为底时的对数(自然对数)运算。

■函数定义 **BOOL** LOG_E_MD(BOOL EN, REAL S1, REAL D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	运算数据(只能指定正数)(实数数据)
D	OUT	运算结果($\log_e$S1)(实数数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中指定的值进行以e为底时的          *)
(*对数(自然对数)运算，并将结果存储到Result中。                      *)
   LOG_E_MD(X0, rData, Result);
```

●对应的MELSEC指令

• LOG(浮动小数点自然对数运算(单精度))

5.18.12 随机数生成 RND_M

生成0～32767的随机数。

■函数定义 **BOOL** RND_M(BOOL EN, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
D	OUT	随机数生成结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则生成0～32767的随机数，并将结果存储到Result中。*)
   RND_M(X0, Result);
```

●对应的MELSEC指令

• RND(随机数生成)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.18.13 系列变更 SRND_M

根据指定的16位BIN数据的内容对随机数系列进行变更。

■函数定义 **BOOL** SRND_M(BOOL EN, ANY16 S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	随机数系列变更结果(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则根据iData中指定的软元件中存储的          *)
(*16位BIN数据的内容对随机数系列进行变更。                          *)
   SRND_M(X0, iData);
```

●对应的MELSEC指令

・SRND(系列变更)

5.18.14 BCD4位平方根 BSQR_MD

进行指定的BCD4位数据的平方根运算。

■函数定义 **BOOL** BSQR_MD(BOOL EN, ANY16 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行运算的BCD4位数据(BIN16位数据)
D	OUT	运算结果(BIN32位数据)

备注) “D”中不能进行位软元件的位数指定。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则进行iData中指定的值的平方根运算，        *)
(*并将结果存储到dData中。                                          *)
   BSQR_MD(X0, iData, dData);
```

●对应的MELSEC指令

・BSQR(BCD4位平方根)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.18.15 BCD8位平方根 BDSQR_MD

进行指定的BCD8位数据的平方根运算。

■函数定义 **BOOL** BDSQR_MD(BOOL EN, ANY32 S1, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行运算的BCD8位数据(BIN32位数据)
D	OUT	运算结果(BIN32位数据)

备注) “D”中不能进行位软元件的位数指定。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对dData中指定的值进行平方根运算，          *)
(*并将结果存储到Result中。                                        *)
   BDSQR_MD(X0, dData, Result);
```

●对应的MELSEC指令

• BDSQR(BCD8位平方根)

5.18.16 BCD型SIN运算 BSIN_MD

对指定角度的BCD4位数据进行SIN(正弦)运算。

■函数定义 **BOOL** BSIN_MD(BOOL EN, ANY16 S1, ANY16(3) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	运算数据(BCD4位数据)		
D	OUT	运算结果 (ARRAY [0..2] OF ANY16)	D[0]	符号(正: 0；负: 1)
			D[1]	整数部分(BCD4位数据)
			D[2]	小数部分(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则进行iData中指定角度的SIN值运算，          *)
(*并将运算结果符号存储到ArrayData[0]中，将运算结果的整数部分        *)
(*存储到ArrayData[1]中，将小数部分存储到ArrayData[2]中。           *)
   BSIN_MD(X0, iData, ArrayData);
```

●对应的MELSEC指令

• BSIN(BCD型SIN运算)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.18.17 BCD型COS运算 BCOS_MD

对指定角度的BCD4位数据进行COS(余弦)运算。

■函数定义

BOOL BCOS_MD(BOOL EN, ANY16 S1, ANY16(3) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	运算数据(BCD4位数据)		
D	OUT	运算结果 (ARRAY [0..2] OF ANY16)	D[0]	符号(正: 0; 负: 1)
			D[1]	整数部分(BCD4位数据)
			D[2]	小数部分(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对iData中指定的角度进行COS值运算，          *)
(*并将运算结果符号存储到ArrayData[0]中，将运算结果的整数部分          *)
(*存储到ArrayData[1]中，将小数部分存储到ArrayData[2]中。              *)
   BCOS_MD(X0, iData, ArrayData);
```

●对应的MELSEC指令

・BCOS(BCD型COS演算)

5.18.18 BCD型TAN运算 BTAN_MD

对指定角度的BCD4位数据进行TAN(正切)运算。

■函数定义

BOOL BTAN_MD(BOOL EN, ANY16 S1, ANY16(3) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	运算数据(BCD4位数据)		
D	OUT	运算结果 (ARRAY [0..2] OF ANY16)	D[0]	符号(正: 0; 负: 1)
			D[1]	整数部分(BCD4位数据)
			D[2]	小数部分(BCD4位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对iData中指定的角度进行TAN值运算，          *)
(*并将运算结果符号存储到ArrayData[0]中，将运算结果的整数部分          *)
(*存储到ArrayData[1]中，将小数部分存储到ArrayData[2]中。              *)
   BTAN_MD(X0, iData, ArrayData);
```

●对应的MELSEC指令

・BTAN(BCD型TAN演算)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.18.19 BCD型SIN^{-1}运算 BASIN_MD

进行指定的BCD值的SIN^{-1}(反正弦)值运算。

■函数定义

BOOL BASIN_MD(BOOL EN, ANY16(3) S1, ANY16 D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	运算数据 (ARRAY [0..2] OF ANY16)	S[0]	符号(正: 0; 负: 1)
			S[1]	整数部分(BCD4位数据)
			S[2]	小数部分(BCD4位数据)
D	OUT	运算结果(软元件的起始编号)(BCD4位数据)		

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对BasinArrayData中指定的值进行SIN-1值运算，*)
(*并将结果存储到Result中。                                    *)
   BASIN_MD(X0, BasinArrayData , Result);
```

●对应的MELSEC指令

・BASIN(BCD型SIN^{-1}演算)

5.18.20 BCD型COS^{-1}运算 BACOS_MD

进行指定的BCD值的COS^{-1}(反余弦)值运算。

■函数定义

BOOL BACOS_MD(BOOL EN, ANY16(3) S1, ANY16 D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	进行COS^{-1}(反余弦)运算的数据 (ARRAY [0..2] OF ANY16)	S[0]	符号(正: 0; 负: 1)
			S[1]	整数部分(BCD4位数据)
			S[2]	小数部分(BCD4位数据)
D	OUT	运算结果(软元件的起始编号)(BCD4位数据)		

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对BacosArrayData中指定的值进行COS-1值运算，*)
(*并将结果存储到Result中。                                    *)
   BACOS_MD(X0, BacosArrayData, Result);
```

●对应的MELSEC指令

・BACOS(BCD型COS^{-1}运算)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.18.21 BCD型TAN^{-1}运算 BATAN_MD

进行指定的BCD值的TAN^{-1}(反正切)值运算。

■函数定义

BOOL BATAN_MD(BOOL EN, ANY16(3) S1, ANY16 D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	存储运算数据的软元件的起始编号 (ARRAY [0..2] OF ANY16)	S[0]	符号(正: 0; 负: 1)
			S[1]	整数部分(BCD4位数据)
			S[2]	小数部分(BCD4位数据)
D	OUT	运算结果(BCD4位数据)		

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则进行BatanArrayData中指定的值的TAN-1值运算，*)
(*并将结果存储到Result中。                                    *)
   BATAN_MD(X0, BatanArrayData, Result);
```

●对应的MELSEC指令

・BATAN(BCD型TAN^{-1}运算)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 98

5.19 数据控制

5.19.1 上下限极限控制 LIMIT_MD

根据指定的BIN16位数据是否在上下限极限值的范围内，对输出值进行控制。

■函数定义

BOOL LIMIT_MD(BOOL EN, ANY16 S1, ANY16 S2, ANY16 S3, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	下限极限值(BIN16位数据)
S2	IN	上限极限值(BIN16位数据)
S3	IN	输入值(BIN16位数据)
D	OUT	输出值(BIN16位数据)

备注) 输出值按以下方式被控制。

S1(下限极限值)＞ S3(输入值)时・・・・・・・S1(下限极限值)→ D(输出值)
S2(上限极限值)＜ S3(输入值)时・・・・・・・S2(上限极限值)→ D(输出值)
S1(下限极限值)≦S3(输入值)≦ S2(上限极限值)时
・・・・S3(输入值)→ D(输出值)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则根据iData3中指定的输入值是否在iData1、        *)
(*iData2中指定的上下限极限值的范围内，将输出值存储到Result中。        *)
   LIMIT_MD(X0, iData1, iData2, iData3, Result);
```

●对应的MELSEC指令

・LIMIT(16位上下限极限控制)

5.19.2 32位数据上下限极限控制 DLIMIT_MD

根据指定的BIN32位数据是否在上下限极限值的范围内，对输出值进行控制。

■函数定义

BOOL DLIMIT_MD(BOOL EN, ANY32 S1, ANY32 S2, ANY32 S3, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	下限极限值(BIN32位数据)
S2	IN	上限极限值(BIN32位数据)
S3	IN	输入值(BIN32位数据)
D	OUT	输出值(BIN32位数据)

备注) 输出值按以下方式被控制。

S1(下限极限值)＞ S3(输入值)时・・・・・・・・S1(下限极限值)→ D(输出值)
S2(上限极限值)＜ S3(输入值)时・・・・・・・・S2(上限极限值)→ D(输出值)
S1(下限极限值)≦S3(输入值)≦ S2(上限极限值)时
・・・・S3(输入值)→ D(输出值)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则根据dData3中指定的输入值是否在dData1、        *)
(*dData2中指定的上下限极限值的范围内，将输出值存储到Result中。        *)
   DLIMIT_MD(X0, dData1, dData2, dData3, Result);
```

●对应的MELSEC指令

・DLIMIT(32位上下限极限控制)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5 - 99

5.19.3 死区控制 BAND_MD

根据指定的BIN16位数据是否在指定的死区的上下限范围内，对输出值进行控制。

■函数定义

BOOL BAND_MD(BOOL EN, ANY16 S1, ANY16 S2, ANY16 S3, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	死区的下限值数据(BIN16位数据)
S2	IN	死区的上限值数据(BIN16位数据)
S3	IN	输入值(BIN16位数据)
D	OUT	输出值(BIN16位数据)

备注) 输出值按以下方式被控制。

S1(下限值)＞ S3(输入值)时・・・・・・・・ S3(输入值)- S1(下限值)→ D(输出值)
S2(上限值)＜ S3(输入值)时・・・・・・・・ S3(输入值)- S2(上限值)→ D(输出值)
S1(下限值)≦S3(输入值)≦ S2(上限值)时・・・・・・・・・・・ 0 → D(输出值)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则根据iData3中指定的输入值是否在iData1、        *)
(*iData2中指定的死区的上下限范围内，将输出值存储到Result中。*)
   BAND_MD(X0, iData1, iData2, iData3, Result);
```

●对应的MELSEC指令

・BAND(16位死区控制)

5.19.4 32位数据死区控制 DBAND_MD

根据指定的BIN32位数据是否在指定的死区的上下限范围内，对输出值进行控制。

■函数定义

BOOL DBAND_MD(BOOL EN, ANY32 S1, ANY32 S2, ANY32 S3, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	死区的下限值数据(BIN32位数据)
S2	IN	死区的上限值数据(BIN32位数据)
S3	IN	输入值(BIN32位数据)
D	OUT	输出值(BIN32位数据)

备注) 输出值按以下方式被控制。

S1(下限值)＞ S3(输入值)时・・・・・・・・ S3(输入值)- S1(下限值)→ D(输出值)
S2(上限值)＜ S3(输入值)时・・・・・・・・ S3(输入值)- S2(上限值)→ D(输出值)
S1(下限值)≦S3(输入值)≦ S2(上限值)时・・・・・・・・・・・ 0 → D(输出值)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则根据iData3中指定的输入值是否在iData1、        *)
(*iData2中指定的死区的上下限范围内，将输出值存储到Result中。*)
   DBAND_MD(X0, dData1, dData2, dData3, Result);
```

●对应的MELSEC指令

・DBAND(32位死区控制)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.19.5 位域控制 ZONE_MD

对指定的BIN16位数据附加偏置值，对输出值进行区域控制。

■函数定义

BOOL ZONE_MD(BOOL EN, ANY16 S1, ANY16 S2, ANY16 S3, ANY16 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	加到输入值中的负偏置值(BIN16位数据)
S2	IN	加到输入值中的正偏置值(BIN16位数据)
S3	IN	输入值(BIN16位数据)
D	OUT	输出值(BIN16位数据)

备注) 输出值按以下方式被控制。

S3(输入值)＞0时・・・・・・・・・・・・・S3(输入值)+ S1(负偏置值)→ D(输出值)
S3(输入值)＝0时・・・・・・・・・・・・・・・・・・・・・・・・・0 → D(输出值)
S3(输入值)＞0时・・・・・・・・・・・・・S3(输入值)+ S1(正偏置值)→ D(输出值)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则在iData3中指定的输入值中附加iData1或      *)
(*iData2中指定的偏置值，并将结果存储到Result中。                    *)
   ZONE_MD(X0, iData1, iData2, iData3, Result);
```

●对应的MELSEC指令

・ZONE(16位域控制)

5.19.6 32位数据位域控制 DZONE_MD

在指定的BIN32位数据中附加指定的偏置值，对输出值进行区域控制。

■函数定义

BOOL DZONE_MD(BOOL EN, ANY32 S1, ANY32 S2, ANY32 S3, ANY32 D);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	加到输入值中的负偏置值(BIN32位数据)
S2	IN	加到输入值中的正偏置值(BIN32位数据)
S3	IN	输入值(BIN32位数据)
D	OUT	输出值(BIN32位数据)

备注) 输出值按以下方式被控制。

S3(输入值)＞0时・・・・・・・・・・・・・S3(输入值)+ S1(负偏置值)→ D(输出值)
S3(输入值)＝0时・・・・・・・・・・・・・・・・・・・・・・・・・0 → D(输出值)
S3(输入值)＞0时・・・・・・・・・・・・・S3(输入值)+ S1(正偏置值)→ D(输出值)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则在iData3中指定的输入值中附加iData1或      *)
(*iData2中指定的偏置值，并将结果存储到Result中。*)
   DZONE_MD(X0, dData1, dData2, dData3, Result);
```

●对应的MELSEC指令

・DZONE(32位域控制)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.19.7 文件寄存器的块No.切换 RSET_MD

将程序中使用的文件寄存器的块No.变更为指定的块No.。

■函数定义 **BOOL** RSET_MD(BOOL EN, ANY16 S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	变更的块No.数据(BIN16位数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将程序中使用的文件寄存器的块          *)
(*No.变更为iData中指定的软元件中存储的块No.。                  *)
   RSET_MD(X0, iData);
```

●对应的MELSEC指令

・RSET(文件寄存器的块No.切换)

5.19.8 文件寄存器用文件的设置 QDRSET_M

将程序中使用的文件寄存器的文件名变更为指定的文件名。

■函数定义 **BOOL** QDRSET_M(BOOL EN, STRING S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	要变更的文件寄存器的“驱动器No.: 文件名”(字符串数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将驱动器No.1的文件寄存器的文件名      *)
(*变更为“ABS.QDR”。                                          *)
   QDRSET_M(X0, “1:ABC”);
```

●对应的MELSEC指令

・QDRSET(文件寄存器用文件的设置)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.19.9 注释用文件的设置 QCDSET_M

将程序中使用的注释文件的文件名变更为指定的文件名。

■函数定义 **BOOL** QDRSET_M(BOOL EN, STRING S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	要变更的注释文件的“驱动器No.: 文件名”(字符串数据)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将驱动器No.3的注释文件的文件名          *)
(*变更为“DEF.QCD”。                                          *)
   QCDSET_M(X0, “3:DEF”);
```

●对应的MELSEC指令

• QCDSET(注释用文件的设置)

5.20 时钟

5.20.1 时钟数据的读取 DATERD_MD

根据QCPU/LCPU的时钟因子读取“年、月、日、时、分、秒、星期”。以BIN值存储到存储目标中。

■函数定义

BOOL DATERD_MD(BOOL EN, ANY16(7) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
D	OUT	读取的时钟数据 (ARRAY [0..6] OF ANY16)	D[0]	年(公历: 1980～2079)
			D[1]	月(1～12)
			D[2]	日(1～31)
			D[3]	时(0～23)
			D[4]	分(0～59)
			D[5]	秒(0～59)
			D[6]	星期(0～6)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则根据QCPU/LCPU的时钟因子读取“年、月、日、 *)
(*时、分、秒、星期”，以BIN值存储到TimeData中指定的软元件中。       *)
   DATERD_MD(X0, TimeData);
```

●对应的MELSEC指令

• DATERD(时钟数据的读取)

5.20.2 时钟数据的写入 DATEWR_MD

将时钟数据“年、月、日、时、分、秒、星期”写入到QCPU/LCPU的时钟因子中。

■函数定义

BOOL DATEWR_MD(BOOL EN, ANY16(7) S);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S	IN	写入的时钟数据 (ARRAY [0..6] OF ANY16)	S[0]	年(公历: 1980～2079)
			S[1]	月(1～12)
			S[2]	日(1～31)
			S[3]	时(0～23)
			S[4]	分(0～59)
			S[5]	秒(0～59)
			S[6]	星期(0～6)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，将TimeData中存储的时钟数据写入到QCPU/LCPU的 *)
(*时钟因子中。                                                  *)
   DATEWR_MD(X0, TimeData);
```

●对应的MELSEC指令

• DATEWR(时钟数据的写入)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.20.3 时钟数据的加法 DATEPLUS_M

将指定的时间数据与指定的时间数据进行加法运算。

■函数定义

BOOL DATEPLUS_M(BOOL EN, ANY16(3) S1, ANY16(3) S2, ANY16(3) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	进行加法运算的时间数据 (ARRAY[0..2] OF ANY16)	S1[0]	时(0～23)
			S1[1]	分(0～59)
			S1[2]	秒(0～59)
S2	IN	进行加法运算的时间数据 (ARRAY[0..2] OF ANY16)	S2[0]	时(0～23)
			S2[1]	分(0～59)
			S2[2]	秒(0～59)
D	OUT	加法运算结果时间数据 (ARRAY[0..2] OF ANY16)	D[0]	时(0～23)
			D[1]	分(0～59)
			D[2]	秒(0～59)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将TimeData1中指定的时间数据与TimeData2      *)
(*中指定的时间数据进行加法运算，并将加法运算结果存储到Result中。      *)
   DATEPLUS_M(X0, TimeData1, TimeData2, Result);
```

●对应的MELSEC指令

・DATE+(时钟数据的加法)

5.20.4 时钟数据的减法 DATEMINUS_M

将指定的时间数据与指定的时间数据进行减法运算。

■函数定义

BOOL DATEMINUS_M(BOOL EN, ANY16(3) S1, ANY16(3) S2, ANY16(3) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	被进行减法运算的时间数据 (ARRAY[0..2] OF ANY16)	S1[0]	时(0～23)
			S1[1]	分(0～59)
			S1[2]	秒(0～59)
S2	IN	进行减法运算的时间数据 (ARRAY[0..2] OF ANY16)	S2[0]	时(0～23)
			S2[1]	分(0～59)
			S2[2]	秒(0～59)
D	OUT	减法运算结果时间数据 (ARRAY[0..2] OF ANY16)	D[0]	时(0～23)
			D[1]	分(0～59)
			D[2]	秒(0～59)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将TimeData1中指定的时间数据与TimeData2      *)
(*中指定的时间数据进行减法运算，并将减法结果存储到Result中。          *)
   DATEMINUS_M(X0, TimeData1, TimeData2, Result);
```

●对应的MELSEC指令

・DATE-(时钟数据的减法)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.20.5 时钟数据格式转换(时、分、秒→秒) SECOND_M

将指定的时间数据转换为秒。

■函数定义

BOOL SECOND_M(BOOL EN, ANY16(3) S, ANY32 D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S	IN	换算的时钟数据 (ARRAY[0..2] OF ANY16)	S[0]	时(0～23)
			S[1]	分(0～59)
			S[2]	秒(0～59)
D	OUT	换算结果时钟数据(秒)(BIN32位数据)		

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将TimeData中指定的时间数据换算为秒        *)
(*并将结果存储到Result中。                                        *)
   SECOND_M(X0, TimeData, Result);
```

●对应的MELSEC指令

・SECOND(时钟数据的格式转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.20.6 时钟数据格式转换(秒→时、分、秒) HOUR_M

将指定的秒数据换算为时、分、秒。

■函数定义

BOOL HOUR_M(BOOL EN, ANY32 S1, ANY16(3) D);

自变量名	IN/OUT	内容		
EN	IN	执行条件(仅TRUE时执行函数)		
S1	IN	换算的时钟数据(秒)(BIN32位数据)		
D	OUT	换算结果时钟数据 (ARRAY[0..2] OF ANY16)	D[0]	时(0～23)
			D[1]	分(0～59)
			D[2]	秒(0～59)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData中指定的秒数据换算为时、分、秒      *)
(*并将结果存储到TimeData中。                                      *)
   HOUR_M(X0, dData, TimeData);
```

●对应的MELSEC指令

・HOUR(时钟数据的格式转换)

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

5.21 程序控制

5.21.1 程序待机 PSTOP_M

将指定文件名的程序置为待机状态。

■函数定义 **BOOL** PSTOP_M(BOOL EN, STRING S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	置为待机状态的程序的文件名 (字符串数据)

备注) 只有程序存储器(驱动器编号: 0)中存储的程序才可以置为待机状态。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将文件名为“ABC”的程序置为待机状态。      *)
  PSTOP_M(X0, “ABC”);
```

●对应的MELSEC指令

• PSTOP(程序待机指令)

5.21.2 程序输出OFF待机 POFF_M

将指定文件名的程序置为非执行并进入待机状态。

■函数定义 **BOOL** POFF_M(BOOL EN, STRING S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	置为非执行并进入待机状态的程序文件名(字符串数据)

备注) 只有程序存储器(驱动器编号: 0)中存储的程序才可以置为非执行并进入待机状态。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将文件名为“ABC”的程序置为非执行      *)
(*并进入待机状态。                                          *)
  POFF_M(X0, sData);
```

●对应的MELSEC指令

• POFF(程序输出OFF待机指令)

5.21.3 程序扫描执行登录 PSCAN_M

将指定文件名的程序置为扫描执行状态。

■函数定义 **BOOL** PSCAN_M(BOOL EN, STRING S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	置为扫描执行状态的程序文件名(字符串数据)

备注) 只有程序存储器(驱动器编号: 0)中存储的程序才可以置为扫描执行状态。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将文件名为“ABC”的程序置为扫描          *)
(*执行状态。                                                    *)
   PSCAN_M(X0, sData);
```

●对应的MELSEC指令

• PSCAN(程序扫描执行登录指令)

5.21.4 程序低速执行登录 PLOW_M

将指定文件名的程序置为低速执行状态。

■函数定义 **BOOL** PLOW_M(BOOL EN, STRING S1);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	置为低速执行状态的程序文件名(字符串数据)

备注) 只有程序存储器(驱动器编号:0)中存储的程序才可以置为低速执行状态。

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将文件名为“ABC”的程序置为低速执行状态。    *)
   PLOW_M(X0, “ABC”);
```

●对应的MELSEC指令

• PLOW(程序低速执行登录指令)

5.22 其它

5.22.1 WDT复位 WDT_M

通过顺控程序对看门狗定时器进行复位。

■函数定义 **BOOL** WDT_M(BOOL EN);

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)

返回值	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则通过顺控程序对看门狗定时器                *)
(*进行复位。                                                        *)
   WDT_M(X0);
```

●对应的MELSEC指令

• WDT(看门狗定时器复位)

5 - 109

备忘录

5 - 110

6 IEC函数

函数的阅读方法

本手册中记载了IEC函数的函数定义・自变量・返回值・使用示例等。
IEC函数是由MELSEC公共指令组合而成。关于IEC函数的可用软元件、执行函数时发生的出错、可以使用的CPU类型，请参阅“MELSEC-Q/L 编程手册(公共指令篇)”。参阅章节记载在本手册的“・使用示例表内的使用指令”一栏中。

6.1.6 双精度整数型(DINT)→实数型(REAL)转换　DINT_TO_REAL / DINT_TO_REAL_E

将双精度整数型(DINT)数据转换为实数型(REAL)数据。 → 1)

■函数定义

REAL DINT_TO_REAL(DINT S1);
2) 3) 4) 5)

●变量 → 6)

变量名	IN/OUT	内容
S1	IN	转换数据(BIN32位数据)

●返回值 → 7)

返回值名	内容
REAL	转换结果(实数数据)

●使用示例 → 8)

变量类型	ST程序	转换结果		使用指令
DINT	r_data1 := DINT_TO_REAL(di_data1);	LD DFLT	SM400 di_data1 r_data1	LD,DFLT

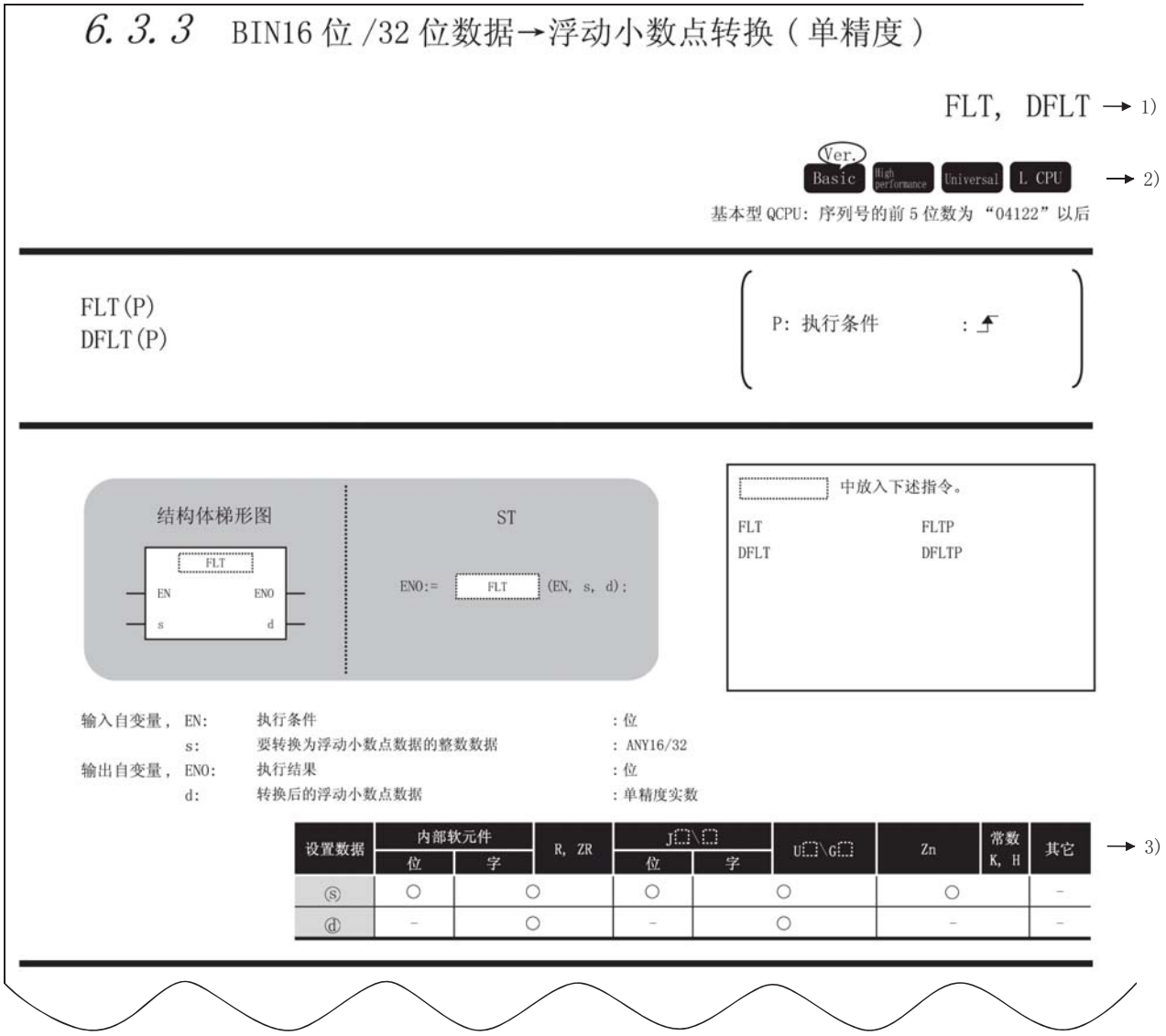
9) 10) 11)

1) 表示函数的功能。
2) 表示函数的数据类型。
3) 表示函数名称。
4) 表示变量的数据类型。(STRING型是通过STRING(字符数)表示。字符数6的情况下，变为STRING(6)。)
5) 表示自变量名称。
6) 表示函数中使用的自变量的列表(自变量名・IN/OUT・内容)。
7) 表示函数中使用的返回值的列表(返回值名・内容)。
8) 表示函数的使用示例。(表示使用了实际软元件・标签的示例。)
9) 本例是使用了REAL型(实数型)标签的示例。
10) 本例是使用了DINT型(双字型)标签的示例。
11) 表示与函数对应的QCPU(Q模式)/LCPU MELSEC公共指令。

6 - 1

“MELSEC-Q/L 编程手册(公共指令篇)”MELSEC指令与本手册的IEC函数的对应如下所示。

MELSEC-Q/L 编程手册(公共指令篇)“MELSEC指令”



6.3.3 BIN16 位 /32 位数据→浮动小数点转换（单精度）

FLT, DFLT → 1)

Ver. Basic | High performance | Universal | L CPU → 2)

基本型 QCPU：序列号的前 5 位数为 “04122” 以后

FLT(P)
DFLT(P)

P：执行条件 ：

结构体梯形图

ST

ENO:= FLT (EN, s, d);

□ 中放入下述指令。

FLT	FLTP
DFLT	DFLTP

输入自变量，	EN:	执行条件	：位
	s:	要转换为浮动小数点数据的整数数据	：ANY16/32
输出自变量，	ENO:	执行结果	：位
	d:	转换后的浮动小数点数据	：单精度实数

设置数据	内部软元件		R, ZR	J□\□		U□\G□	Zn	常数 K, H	其它
	位	字		位	字				
ⓢ	○	○		○		○	○		-
ⓓ	-	○		-		○	-		-

→ 3)

本书“MELSEC函数”

使用示例

自变量类型	ST程序	转换结果		使用指令
DINT	r_data1 := DINT_TO_REAL(di_data1);	LD	SM400	LD,DFLT ↓ 4)
		DFLT	di_data1	
			r_data1	

1) MELSEC指令参照目标
2) 可以使用的CPU类型
表示可以使用指令的CPU类型。
3) 可用软元件
4) 进行参照的MELSEC公共指令

6.1 类型转换功能

6.1.1 布尔型(BOOL)→双精度整数型(DINT)转换 BOOL_TO_DINT BOOL_TO_DINT_E

将指定的布尔型(BOOL)数据转换为双精度整数型(DINT)数据。

■函数定义

```
DINT BOOL_TO_DINT( BOOL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(位数据)

●返回值

返回值名	内容
DINT	转换结果(BIN32位数据)

备注) 在返回值的最低位处存储转换数据(位数据)。

●使用示例

自变量型	ST程序	转换结果	使用指令
BOOL	di_data1 := BOOL_TO_DINT(b_data1);	LD b_data1 DMOV K1 di_data1 LDI b_data1 DMOV K0 di_data1	LD, DMOV, LDI

■函数定义

```
BOOL BOOL_TO_DINT_E( BOOL EN, BOOL S1, DINT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(位数据)
D1	OUT	转换结果(BIN32位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将bData的布尔型数据转换为双精度整数型(DINT)，        *)
(*并将结果存储到Result中。                                                    *)
    M0 := BOOL_TO_DINT_E( X0, bData, Result ) ;
```

6.1.2 布尔型(BOOL)→整数型(INT)转换 BOOL_TO_INT BOOL_TO_INT_E

将布尔型(BOOL)数据转换为整数型(INT)数据。

■函数定义

```
INT BOOL_TO_INT( BOOL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(位数据)

●返回值

返回值名	内容
INT	转换结果(BIN16位数据)

备注) 转换数据(位数据)被存储在返回值的最低位处。

●使用示例

自变量类型	ST程序	转换结果	使用指令
INT	D50 := BOOL_TO_INT(M100);	LD M100 MOV K1 D50 LDI M100 MOV K0 D50	LD, MOV, LDI

■函数定义

```
BOOL BOOL_TO_INT_E( BOOL EN, BOOL S1, INT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(位数据)
D1	OUT	转换结果(BIN16位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将bData的布尔型(BOOL)转换为整数型(INT)   *)
(*并将结果存储到Result中。                                          *)
 M0 := BOOL_TO_INT_E( X0, bData, Result ) ;
```

6.1.3 布尔型(BOOL)→字符串型(STRING)转换　BOOL_TO_STR
BOOL_TO_STR_E

将布尔型(BOOL)数据转换为字符串型(STRING)数据。

■函数定义　STRING(2) BOOL_TO_STR(BOOL S1);

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(位数据)

●返回值

返回值名	内容
STRING(2)	转换结果(字符串数据)

备注) 转换数据(位数据)为0的情况下，返回值将变为“0”。
转换数据(位数据)为1的情况下，返回值将变为“1”。

●使用示例

自变量类型	ST程序	转换结果	使用指令
BOOL	s_ary1 := BOOL_TO_STR(b_data1);	LD　b_data1 MOV　K49 　　s_ary1 LDI　b_data1 MOV　K48 　　s_ary1	LD, MOV, LDI

■函数定义　BOOL BOOL_TO_STR_E(BOOL EN, BOOL S1, STRING(2) D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(位数据)
D1	OUT	转换结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将bData的布尔型(BOOL)数据转换为字符串型，          *)
(*并将结果存储到Result中。                                              *)
    M0 := BOOL_TO_STR _E( X0, bData, Result ) ;
```

6.1.4 双精度整数型(DINT)→布尔型(BOOL)转换 DINT_TO_BOOL DINT_TO_BOOL_E

将双精度整数型(DINT)数据转换为布尔型(BOOL)数据。

■函数定义

```
BOOL DINT_TO_BOOL( DINT S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN32位数据)

●返回值

返回值名	内容
BOOL	转换结果(位数据)

备注) 转换数据(BIN32位数据)为0的情况下，返回值将变为“0”。
转换数据(BIN32位数据)为0以外的情况下，返回值将变为“1”。

●使用示例

自变量类型	ST程序	转换结果	使用指令
DINT	M100 := DINT_TO_BOOL(di_data1);	LDD<> di_data1 K0 OUT M100	LDD<>, OUT

■函数定义

```
BOOL DINT_TO_BOOL _E( BOOL EN, DINT S1, BOOL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D1	OUT	转换结果(位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData的双精度整数型(DINT)数据转换为        *)
(*布尔型(BOOL)，并将结果存储到Result中。                              *)
    M0 := DINT_TO_BOOL_E( X0, dData, Result ) ;
```

6.1.5 双精度整数型(DINT)→整数型(INT)转换　DINT_TO_INT
DINT_TO_INT_E

将双精度整数型(DINT)数据转换为整数型(INT)数据。

■函数定义　INT DINT_TO_INT(DINT S1);

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN32位数据)

●返回值

返回值名	内容
INT	转换结果(BIN16位数据)

备注) 将转换数据(BIN32位数据)的低16位存储到返回值中。
高16位将被舍去。

●使用示例

变量类型	ST程序	转换结果		使用指令
DINT	i_data1 := DINT_TO_INT(di_data1);	LD MOV	SM400 di_data1 i_data1	LD,MOV

■函数定义　BOOL DINT_TO_INT_E(BOOL EN, DINT S1, INT D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D1	OUT	转换结果(BIN16位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData的双精度整数型(DINT)数据转换为        *)
(*整数型(INT)数据，并将结果存储到Result中。                            *)
    M0 := DINT_TO_INT_E( X0, dData, Result ) ;
```

6.1.6 双精度整数型(DINT)→实数型(REAL)转换 DINT_TO_REAL DINT_TO_REAL_E

将双精度整数型(DINT)数据转换为实数型(REAL)数据。

■函数定义

```
REAL DINT_TO_REAL( DINT S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN32位数据)

●返回值

返回值名	内容
REAL	转换结果(实数数据)

●使用示例

变量类型	ST程序	转换结果		使用指令
DINT	r_data1 := DINT_TO_REAL(di_data1);	LD DFLT	SM400 di_data1 r_data1	LD,DFLT

■函数定义

```
BOOL DINT_TO_REAL_E( BOOL EN, DINT S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D1	OUT	转换结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData的双精度整数型(DINT)数据转换为          *)
(*实数型(REAL) 数据，并将结果存储到Result中。                          *)
    M0 := DINT_TO_REAL_E( X0, dData, Result ) ;
```

6.1.7 双精度整数型(DINT)→字符串型(STRING)转换 DINT_TO_STR DINT_TO_STR_E

将双精度整数型(DINT)数据转换为字符串型(STRING)数据。

■函数定义

```
STRING(12) DINT_TO_STR( DINT S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN32位数据)

●返回值

返回值名	内容
STRING(12)	转换结果(字符串数据)

备注) 本函数在基本型QCPU中不能使用。

●使用示例

变量类型	ST程序	转换结果	使用指令
DINT	s_ary1 := DINT_TO_STR(K65535);	LD SM400 DBINDA K65535 s_ary1	LD,DBINDA

■函数定义

```
BOOL DINT_TO_STR _E( BOOL EN, DINT S1, STRING(12) D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN32位数据)
D1	OUT	转换结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将dData的双精度整数型(DINT)数据转换为          *)
(*字符串型数据，并将结果存储到Result中。                                  *)
     M0 := DINT_TO_STR_E( X0, dData, Result ) ;
```

6.1.8 整数型(INT)→布尔型(BOOL)转换 INT_TO_BOOL INT_TO_BOOL_E

将整数型(INT)数据转换为布尔型(BOOL)数据。

■函数定义

```
BOOL INT_TO_BOOL( INT S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN16位数据)

●返回值

返回值名	内容
BOOL	转换结果(位数据)

备注) 转换数据(BIN16位数据)为0的情况下，返回值将变为“0”。
转换数据(BIN16位数据)为0以外的情况下，返回值将变为“1”。

●使用示例

变量类型	ST程序	转换结果	使用指令
INT	b_data1 := INT_TO_BOOL(i_data1);	LD<> i_data1 K0 OUT b_data1	LD<>,OUT

■函数定义

```
BOOL INT_TO_BOOL _E( BOOL EN, INT S1, BOOL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D1	OUT	转换结果(位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData的整数型(INT)数据转换为布尔型          *)
(*(BOOL)数据，并将结果存储到Result中。                                  *)
    M0 := INT_TO_BOOL_E( X0, iData, Result ) ;
```

6.1.9 整数型(INT)→双精度整数型(DINT)转换 INT_TO_DINT INT_TO_DINT_E

将整数型(INT)数据转换为双精度整数型(DINT)数据。

■函数定义

```
DINT INT_TO_DINT( INT S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN16位数据)

●返回值

返回值名	内容
DINT	转换结果(BIN32位数据)

●使用示例

自变量型	ST程序	转换结果		使用指令
INT	di_data1 := INT_TO_DINT(D500);	LD DBL	SM400 D500 di_data1	LD,DBL

■函数定义

```
BOOL INT_TO_DINT _E( BOOL EN, INT S1, DINT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D1	OUT	转换结果(BIN32位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData的整数型(INT)数据转换为双精度整数型  *)
(*(DINT)数据，并将结果存储到Result中。                                  *)
    M0 := INT_TO_DINT_E( X0, iData, Result ) ;
```

6.1.10 整数型(INT)→实数型(REAL)转换 INT_TO_REAL INT_TO_REAL_E

将整数型(INT)数据转换为实数型(REAL)数据。

■函数定义

```
REAL INT_TO_REAL( INT S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN16位数据)

●返回值

返回值名	内容
REAL	转换结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
INT	w_Real1:= INT_TO_REAL(D0);	LD SM400 FLT D0 w_Real1	LD,FLT

■函数定义

```
BOOL INT_TO_REAL_E( BOOL EN, INT S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D1	OUT	转换结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData的整数型(INT)数据转换为实数型          *)
(*(REAL)数据，并将结果存储到Result中。                                    *)
     M0 := INT_TO_REAL_E( X0, iData, Result ) ;
```

6.1.11 整数型(INT)→字符串型(STRING)转换 INT_TO_STR INT_TO_STR_E

将整数型(INT)数据转换为字符串型(STRING)数据。

■函数定义

```
STRING(8) INT_TO_STR( INT S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(BIN16位数据)

●返回值

返回值名	内容
STRING(8)	转换结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。

●使用示例

自变量型	ST程序	转换结果	使用指令
INT	w_Str1 := INT_TO_STR(D0);	LD SM400 BINDA D0 w_Str1	LD,BINDA

■函数定义

```
BOOL INT_TO_STR _E( BOOL EN, INT S1, STRING(8) D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(BIN16位数据)
D1	OUT	转换结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将iData的整数型(INT)数据转换为字符串型数据，      *)
(*将结果存储到Result中。                                                    *)
    M0 := INT_TO_STR_E( X0, iData, Result ) ;
```

6.1.12 实数型(REAL)→双精度整数型(DINT)转换 REAL_TO_DINT REAL_TO_DINT_E

将指定的实数型(REAL)数据转换为双精度整数型(DINT)数据。

■函数定义

```
DINT REAL_TO_DINT( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(实数数据)

●返回值

返回值名	内容
DINT	转换结果(BIN32位数据)

●使用示例

自变量型	ST程序	转换结果		使用指令
REAL	w_DWord1:= REAL_TO_DINT(w_Real 1);	LD DINT	SM400 w_Real1 w_DWord1	LD,DINT

■函数定义

```
BOOL REAL_TO_DINT_E( BOOL EN, REAL S1, DINT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(实数数据)
D1	OUT	转换结果(BIN32位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData的实数型(REAL)数据转换为双精度          *)
(*整数型(DINT)数据，并将结果存储到Result中。                              *)
    M0 := REAL_TO_DINT_E( X0, rData, Result ) ;
```

6.1.13 实数型(REAL)→整数型(INT)转换 REAL_TO_INT REAL_TO_INT_E

将实数型(REAL)数据转换为整数型(INT)数据。

■函数定义

```
INT REAL_TO_INT( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(实数数据)

●返回值

返回值名	内容
INT	转换结果(BIN16位数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Word1:= REAL_TO_INT(w_Real1);	LD SM400 INT w_Real1 w_Word1	LD,INT

■函数定义

```
BOOL REAL_TO_INT_E( BOOL EN, REAL S1, INT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(实数数据)
D1	OUT	转换结果(BIN16位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData的实数型(REAL)数据转换为整数型      *)
(*(INT)数据，并将结果存储到Result中。                                *)
    M0 := REAL_TO_INT_E( X0, rData, Result ) ;
```

6.1.14 实数型(REAL)→字符串型(STRING)转换 REAL_TO_STR REAL_TO_STR_E

将实数型(REAL)数据转换为字符串型数据。

■函数定义

```
STRING(14) REAL_TO_STR( REAL S1);
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(实数数据)

●返回值

返回值名	内容
STRING(14)	转换结果(字符串数据)

注) ESTR指令的显示格式为指数格式，全部位数为13，小数部分位数为5。

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Str1:= REAL_TO_STR(w_Real1);	LD SM400 MOV K1 D10237 MOV K13 D10238 MOV K5 D10239 ESTR w_Real1 D10237 w_Str1	LD,MOV,ESTR

■函数定义

```
BOOL REAL_TO_STR _E( BOOL EN, REAL S1, STRING(14) D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(实数数据)
D1	OUT	转换结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData的实数型(REAL)数据转换为字符串型数据，      *)
(*并将结果存储到Result中。                                              *)
    M0 := REAL_TO_STR_E( X0, rData, Result ) ;
```

6.1.15 字符串型(STRING)→布尔型(BOOL)转换 STR_TO_BOOL STR_TO_BOOL_E

将字符串型(STRING)的数据转换为布尔型(BOOL)数据。

■函数定义

```
BOOL STR_TO_BOOL( STRING(2) S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(字符串数据)

●返回值

返回值名	内容
BOOL	转换结果(位数据)

备注) 转换数据(字符串数据)为0的情况下，返回值将变为“0”。
转换数据(字符串数据)为0以外的情况下，返回值将变为“1”。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	w_Bit1:= STR_TO_BOOL(w_Str1);	LD<> w_Str1 K48 OUT w_Bit1	LD<>,OUT

■函数定义

```
BOOL STR_TO_BOOL_E( BOOL EN, STRING(2) S1, BOOL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(字符串数据)
D1	OUT	转换结果(位数据

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData的字符串型数据转换为布尔型数据，        *)
(*并将结果存储到Result中。                                              *)
    M0 := STR_TO_BOOL _E( X0, sData, Result ) ;
```

6.1.16 字符串型(STRING)→双精度整数型(DINT)转换 STR_TO_DINT STR_TO_DINT_E

将字符串型(STRING)数据转换为双精度整数型(DINT)数据。

■函数定义

```
DINT STR_TO_DINT( STRING(12) S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(字符串数据)

●返回值

返回值名	内容
DINT	转换结果(BIN32位数据)

备注) 本函数不能在基本型QCPU中使用。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	w_DWord1:= STR_TO_DINT("123");	LD SM400 DDABIN "123" w_DWord1	LD,DDABIN

■函数定义

```
BOOL STR_TO_DINT _E( BOOL EN, STRING(12) S1, DINT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(字符串数据)
D1	OUT	转换结果(BIN32位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData的字符串型数据转换为                    *)
(*双精度整数型(DINT)数据，并将结果存储到Result中。                        *)
     M0 := STR_TO_DINT_E( X0, sData, Result ) ;
```

6.1.17 字符串型(STRING)→整数型(INT)转换 STR_TO_INT STR_TO_INT_E

将字符串型(STRING)数据转换为整数型(INT)数据。

■函数定义

```
INT STR_TO_INT( STRING(6) S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(字符串数据)

●返回值

返回值名	内容
INT	转换结果(BIN16位数据)

备注) 本函数不能在基本型QCPU中使用。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	w_Word1:= STR_TO_INT(w_Str1);	LD SM400 DABIN w_Str1 w_Word1	LD,DABIN

■函数定义

```
BOOL STR_TO_INT _E( BOOL EN, STRING(6) S1, INT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(字符串数据)
D1	OUT	转换结果(BIN16位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData的字符串型数据转换为整数型(INT)数据，      *)
(*并将结果存储到Result中。                                                  *)
    M0 := STR_TO_INT_E( X0, sData, Result ) ;
```

6.1.18 字符串型(STRING)→实数型(REAL)转换 STR_TO_REAL STR_TO_REAL_E

将字符串型(STRING)数据转换为实数型(REAL)数据。

■函数定义

```
REAL STR_TO_REAL( STRING(24) S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	转换数据(字符串数据)

●返回值

返回值名	内容
REAL	转换结果 (实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	w_Real1:= STR_TO_REAL(w_Str1);	LD SM400 EVAL w_Str1 w_Real1	LD,EVAL

■函数定义

```
BOOL STR_TO_REAL _E( BOOL EN, STRING(24) S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	转换数据(字符串数据)
D1	OUT	转换结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，将sData的字符串型数据转换为实数型(REAL)数据，        *)
(*并将结果存储到Result中。                                                  *)
    M0 := STR_TO_REAL_E( X0, sData, Result ) ;
```

6.2 数值功能(一般函数)

6.2.1 绝对值 ABS
ABS_E

对指定数据的绝对值进行运算。

■函数定义

```
ANY_NUM ABS( ANY_NUM S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	求出其绝对值的数据

●返回值

返回值名	内容
ANY_NUM	绝对值运算结果

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	r_data1 := ABS(r_data2);	LD SM400 EMOV r_data2 r_data1 LDE< r_data2 E0 E* E-1 r_data2 r_data1	LD,EMOV, LDE<, E*
INT	D0 := ABS(D1);	LD SM400 MOV D1 D0 LD< D1 K0 NEG D0	LD,MOV, LD<, NEG
DINT	di_data1 := ABS(di_data2);	LD SM400 DMOV di_data2 di_data1 LDD< di_data2 K0 DCML di_data2 di_data1 D+ K1 di_data1	LD,DMOV, LDD<, DCML D+

■函数定义

```
BOOL ABS_E( BOOL EN, ANY_NUM S1, ANY_NUM D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	求出其绝对值的数据
D1	OUT	绝对值运算结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算出iData中存储的数据的绝对值，          *)
(*并将结果存储到Result中。                                        *)
    M0 := ABS_E( X0, iData, Result ) ;
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.2.2 平方根 SQRT SQRT_E

对指定数据的平方根进行运算。

■函数定义

```
REAL SQRT( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	求出其平方根的数据(实数数据)

●返回值

返回值名	内容
REAL	平方根运算结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	r_data1 := SQRT(r_data2);	LD SM400 SQR r_data2 r_data1	LD,SQR

■函数定义

```
BOOL SQRT_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	求出其平方根的数据(实数数据)
D1	OUT	平方根运算结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算出rData中存储的数据的平方根，          *)
(*并将结果存储到Result中。                                        *)
    M0 := SQRT_E( X0, rData, Result ) ;
```

6.3 数值功能(对数函数)

6.3.1 自然对数 LN
LN_E

对指定数据的自然对数进行运算。

■函数定义

```
REAL LN( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	求出其自然对数的数据(实数数据)

●返回值

返回值名	内容
REAL	自然对数运算结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	r_data1 := LN(1.23456);	LD SM400 LOG E1.23456 r_data1	LD,LOG

■函数定义

```
BOOL LN_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	求出其自然对数的数据(实数数据)
D1	OUT	自然对数运算结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算出rData中存储的数据的自然对数，          *)
(*并将结果存储到Result中。                                          *)
     M0 := LN_E( X0, rData, Result ) ;
```

6.3.2 自然指数 EXP
EXP_E

对指定的数据的自然指数进行运算。

■函数定义

```
REAL EXP( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	求出其自然指数的数据(实数数据)

●返回值

返回值名	内容
REAL	自然指数运算结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	r_data1 := EXP(r_data2);	LD SM400 EXP r_data2 r_data1	LD,EXP

■函数定义

```
BOOL EXP_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	求出其自然指数的数据(实数数据)
D1	OUT	自然指数运算结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算出rData中存储的数据的自然指数，          *)
(*并将结果存储到Result中。                                          *)
    M0 := EXP_E( X0, rData, Result ) ;
```

6.4 数值功能(三角函数)

6.4.1 浮动小数点SIN运算　SIN
SIN_E

对指定的角度的SIN(正弦)值进行运算。

■函数定义

```
REAL SIN( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	进行SIN(正弦)运算的角度数据(实数数据)

备注) 指定的角度是以弧度单位(角度×π/180)进行设置。

●返回值

返回值名	内容
REAL	SIN运算结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	r_data1 := SIN(1.23456);	LD　SM400 SIN　E1.23456 　　r_data1	LD,SIN

■函数定义

```
BOOL SIN_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行SIN(正弦)运算的角度数据(实数数据) 备注) 指定的角度是以弧度单位(角度×π/180)进行设置。
D1	OUT	SIN运算结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算出rData中存储的角度数据的SIN值，　　*)
(*并将结果存储到Result中。　　*)
    M0 := SIN_E( X0, rData, Result ) ;
```

6.4.2 浮动小数点COS运算 COS
COS_E

对指定的角度的COS(余弦)值进行运算。

■函数定义

```
REAL COS( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	进行COS(余弦)运算的角度数据(实数数据)

备注) 指定的角度是以弧度单位(角度×π/180)进行设置。

●返回值

返回值名	内容
REAL	COS运算结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Real1 := COS(w_Real2);	LD SM400 COS w_Real2 w_Real1	LD,COS

■函数定义

```
BOOL COS_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行COS(余弦)运算的角度数据(实数数据) 备注) 指定的角度是以弧度单位(角度×π/180)进行设置。
D1	OUT	COS运算结果(实数数据)

备注) 指定的角度是以弧度单位(角度×π/180)进行设置。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算出rData中存储的角度数据的COS值，          *)
(*并将结果存储到Result中。                                          *)
    M0 := COS_E( X0, rData, Result ) ;
```

6.4.3 浮动小数点TAN运算　TAN
TAN_E

对指定的角度的TAN(正切)值进行运算。

■函数定义

```
REAL TAN( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	进行TAN(正切)运算的角度数据(实数数据)

备注) 指定的角度是以弧度单位(角度×π/180)进行设置。

●返回值

返回值名	内容
REAL	TAN运算结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Real1 := TAN(w_Real2);	LD　SM400 TAN　w_Real2 　　w_Real1	LD,TAN

■函数定义

```
BOOL TAN_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行TAN(正切)运算的角度数据(实数数据) 备注) 指定的角度是以弧度单位(角度×π/180)进行设置。
D1	OUT	TAN运算结果(实数数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算出rData中存储的角度的TAN值，                *)
(*并将结果存储到Result中。                                              *)
     M0 := TAN_E( X0, rData, Result ) ;
```

6.4.4 浮动小数点SIN^{-1}运算 ASIN
ASIN_E

对指定的SIN值的SIN^{-1}(反正弦)进行运算。

■函数定义 REAL ASIN(REAL S1);

●自变量

自变量名	IN/OUT	内容
S1	IN	进行SIN^{-1}(反正弦) 运算的SIN值(-1.0～1.0)(实数数据)

●返回值

返回值名	内容
REAL	SIN^{-1}运算结果(实数数据)

备注) 本函数不能在基本型QCPU中使用。
运算结果是以弧度为单位的角度数据。

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Real1 := ASIN(w_Real2);	LD SM400 ASIN w_Real2 w_Real1	LD,ASIN

■函数定义 BOOL ASIN_E(BOOL EN, REAL S1, REAL D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行SIN^{-1}(反正弦)运算的SIN值(-1.0～1.0)(实数数据)
D1	OUT	SIN^{-1}运算结果(实数数据)

备注) 运算结果是以弧度为单位的角度数据。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中存储的SIN值进行角度运算，          *)
(*并将结果存储到Result中。                                          *)
    M0 := ASIN_E( X0, rData, Result ) ;
```

6.4.5 浮动小数点COS^{-1}运算 ACOS
ACOS_E

对指定的COS值的COS^{-1}(反余弦)进行运算。

■函数定义

```
REAL ACOS( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	进行COS^{-1}(反余弦)运算的COS值(-1.0～1.0)(实数数据)

●返回值

返回值名	内容
REAL	COS^{-1}运算结果(实数数据)

备注) 本函数不能在基本型QCPU中使用。
运算结果是以弧度为单位的角度数据。

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Real1 := ACOS(w_Real2);	LD SM400 ACOS w_Real2 w_Real1	LD,ACOS

■函数定义

```
BOOL ACOS_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行COS^{-1}(反余弦)运算的COS值(-1.0～1.0)(实数数据)
D1	OUT	COS^{-1}运算结果(实数数据)

备注) 运算结果是以弧度为单位的角度数据。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中存储的COS值进行角度运算，          *)
(*并将结果存储到Result中。                                          *)
    M0 := ACOS_E( X0, rData, Result ) ;
```

6.4.6 浮动小数点TAN^{-1}运算 ATAN
ATAN_E

对指定的TAN值的TAN^{-1}(反正切)进行运算。

■函数定义

```
REAL ATAN( REAL S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	进行TAN^{-1}(反正切)运算的TAN值(实数数据)

●返回值

返回值名	内容
REAL	TAN^{-1}运算结果(实数数据)

备注) 本函数不能在基本型QCPU中使用。
运算结果是以弧度为单位的角度数据。

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Real1 := ATAN(w_Real2);	LD SM400 ATAN w_Real2 w_Real1	LD,ATAN

■函数定义

```
BOOL ATAN_E( BOOL EN, REAL S1, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行TAN^{-1}(反正切)运算的TAN值(实数数据)
D1	OUT	TAN^{-1}运算结果(实数数据)

备注) 运算结果是以弧度为单位的角度数据。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对rData中存储的TAN值进行角度运算，          *)
(*并将结果存储到Result中。                                          *)
    M0 := ATAN_E( X0, rData, Result ) ;
```

6.5 算术运算功能

6.5.1 加法 ADD_E

对指定的多个数据进行加法运算。

■函数定义

```
BOOL ADD_E( BOOL EN, ANY_NUM S1, ANY_NUM S2,....,ANY_NUM Sn, ANY_NUM D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	进行加法运算的数据
D1	OUT	加算运算结果

●返回值

返回值名	内容
BOOL	执行条件(位数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := ADD_E(b_select, r_data1, r_data2, r_data3);	LD b_select E+ r_data1 r_data2 r_data3 LD b_select OUT b_result	LD, E+, OUT
INT	b_result := ADD_E(b_select, D10, D20, D30, D40);	LD b_select + D10 D20 D40 + D30 D40 LD b_select OUT b_result	LD, +, OUT
DINT	b_result := ADD_E(b_select, di_data1, di_data2, di_data3);	LD b_select D+ di_data1 di_data2 di_data3 LD b_select OUT b_result	LD,D+,OUT

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.5.2 乘法 MUL_E

对指定的多个数据进行乘法运算。

■函数定义

BOOL MUL_E(BOOL EN, ANY_NUM S1, ANY_NUM S2,....,ANY_NUM Sn, ANY_NUM D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	进行乘法运算的数据
D1	OUT	乘法运算结果

●返回值

返回值名	内容
BOOL	执行条件(位数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := MUL_E(b_select, r_data1, r_data2, r_data3);	LD b_select E* r_data1 r_data2 r_data3 LD b_select OUT b_result	LD, E*, OUT
INT	b_result := MUL_E(b_select, D10, D20, D30, D40);	LD b_select * D10 D20 D10238 * D10238 D30 D10236 MOV D10236 D40 LD b_select OUT b_result	LD, *, MOV, OUT
DINT	b_result := MUL_E(b_select, di_data1, di_data2, di_data3);	LD b_select D* di_data1 di_data2 D10236 DMOV D10236 di_data3 LD b_select OUT b_result	LD, D*, DMOV, OUT

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.5.3 减法　SUB_E

在指定的数据之间进行减法运算。

■函数定义

```
BOOL SUB_E( BOOL EN, ANY_NUM S1, ANY_NUM S2, ANY_NUM D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被减数据
S2	IN	进行减法运算的数据。
D1	OUT	减法运算结果

●返回值

返回值名	内容
BOOL	执行条件(位数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := SUB_E(b_select, r_data1, r_data2, r_data3);	LD b_select E- r_data1 r_data2 r_data3 LD b_select OUT b_result	LD, E-, OUT
INT	b_result := SUB_E(b_select, 32767, D100, i_data1);	LD b_select - K32767 D100 i_data1 LD b_select OUT b_result	LD, -, OUT
DINT	b_result := SUB_E(b_select, di_data1, di_data2, di_data3);	LD b_select D- di_data1 di_data2 di_data3 LD b_select OUT b_result	LD, D-, OUT

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 33

6.5.4 除法 DIV_E

在指定的数据之间进行除法运算。

■函数定义

```
BOOL DIV_E( BOOL EN, ANY_NUM S1, ANY_NUM S2, ANY_NUM D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被除数据
S2	IN	进行除法运算的数据
D1	OUT	除法运算结果

●返回值

返回值名	内容
BOOL	执行条件(位数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := DIV_E(b_select, r_data1, r_data2, r_data3);	LD b_select E/ r_data1 r_data2 r_data3 LD b_select OUT b_result	LD, E/, OUT
INT	b_result := DIV_E(b_select, D10, D20, D30);	LD b_select / D10 D20 D10238 MOV D10238 D30 LD b_select OUT b_result	LD, /, MOV, OUT
DINT	b_result := DIV_E(b_select, di_data1, di_data2, di_data3);	LD b_select D/ di_data1 di_data2 D10236 DMOV D10236 di_data3 LD b_select OUT b_result	LD, D/, DMOV, OUT

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 34

6.5.5 余数　MOD
MOD_E

在指定的数据之间进行除法运算，并对其余数进行运算。

■函数定义

```
BOOL MOD_E( BOOL EN, ANY_INT S1, ANY_INT S2, ANY_INT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被除数据
S2	IN	进行除法运算的数据
D1	OUT	余数运算结果

●返回值

返回值名	内容
BOOL	执行条件(位数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
INT	B100 := MOD_E(M1, D10, D20, D30);	LD M1 / D10 D20 D10238 MOV D10239 D30 LD M1 OUT B100	LD, /, MOV, OUT
DINT	b_result := MOD_E(b_select, di_data1, di_data2, di_data3);	LD b_select D/ di_data1 di_data2 D10236 DMOV D10238 di_data3 LD b_select OUT b_result	LD, D/, DMOV, OUT

※MOD只能作为运算符使用。

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.5.6 指数　EXPT
EXPT_E

根据指定的设置为底的数据及设置为指数的数据进行指数运算。

■函数定义　REAL EXPT(REAL S1, ANY_NUM S2);

●自变量

自变量名	IN/OUT	内容
S1	IN	设置为底的数据
S2	IN	设置为指数的数据

●返回值

返回值名	内容
REAL	运算结果(实数数据)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	r_data1 := EXPT(r_data2, r_data3);	LD SM400 LOG r_data2 r_data1 E* r_data1 r_data3 r_data1 EXP r_data1 r_data1	LD, LOG, E*, EXP
INT	r_data1 := EXPT(1.123, k32767);	LD SM400 LOG E1.123 r_data1 FLT K32767 D10238 E* r_data1 D10238 r_data1 EXP r_data1 r_data1	LD, LOG, FLT, E*, EXP
DINT	r_data1 := EXPT(r_data2, di_data1);	LD SM400 LOG r_data2 r_data1 DFLT di_data1 D10238 E* r_data1 D10238 r_data1 EXP r_data1 r_data1	LD,LOG, DFLT, E*, EXP

■函数定义

```
BOOL EXPT_E( BOOL EN, REAL S1, ANY_NUM S2, REAL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	设置为底的数据
S2	IN	设置为指数的数据
D1	OUT	运算结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将rData中存储的数据以iData中存储的          *)
(*数据进行指数运算并将结果存储到Result中。                          *)
    M0 := EXPT_E( X0, rData, iData, Result ) ;
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 37

6.5.7 代入　MOVE
MOVE_E

将指定的数据代入指定的存储目标中。

■函数定义　ANY MOVE(ANY S1);

●自变量

自变量名	IN/OUT	内容
S1	IN	代入的数据

●返回值

返回值名	内容
ANY	代入结果数据

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	W_Real1:= MOVE(W_Real2);	LD　SM400 EMOV　w_Real2 　　w_Real1	LD,EMOV
INT	D1 :=MOVE(D0);	LD　SM400 MOV　D0　D1	LD,MOV
DINT	w_DWord1:= MOVE(2147483647);	LD　SM400 DMOV　K2147483647 　　w_DWord1	LD,DMOV
BOOL	w_Bit1:= MOVE(w_Bit2);	LD　SM400 MPS AND　w_Bit2 SET　w_Bit1 MRD ANI　w_Bit2 RST　w_Bit1 MPP OUT　M8191	LD,MPS,AND,SET,MRD, ANI,RST,MPP,OUT
STRING	w_Str1 := MOVE("ABCDEFG");	LD　SM400 $MOV　"ABCDEFG" 　　w_Str1	LD,$MOV

■函数定义　BOOL MOVE_E(BOOL EN, ANY S1, ANY D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	代入的数据
D1	OUT	代入结果数据

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

(*如果执行条件X0变为ON，则将iData中存储的数据存储到Result中。　*)

```
M0 := MOVE_E( X0, iData, Result ) ;
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.6 位移功能

6.6.1 位左移 SHL
SHL_E

将指定数据向左移动n位。

■函数定义

```
ANY_BIT SHL( ANY_BIT S1, ANY_BIT n );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	移动的数据
n	IN	移动的位数 备注)移动的位数只能指定为常数。

●返回值

返回值名	内容
ANY_BIT	被移动的数据 备注)从最低位算起的n位变为0。

●使用示例

自变量型	ST程序	转换结果	使用指令
INT	D0 := SHL(D1,1);	LD SM400 MOV D1 D0 SFL D0 K1	LD,MOV,SFL

■函数定义

```
BOOL SHL_E( BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	移动的数据
n	IN	移动的位数 备注)移动的位数只能指定为常数。
D1	OUT	被移动的数据 备注)从最低位算起的n位变为0。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的数据向左移动2位，          *)
(*并将其结果存储到D100中。                                    *)
    M0:=SHL_E( X0, D0, 2, D100 );
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 39

6.6.2 位右移 SHR
SHR_E

将指定数据向右移动n位。

■函数定义

```
ANY_BIT SHR( ANY_BIT S1, ANY_BIT n );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	移动的数据
n	IN	移动的位数 备注)移动的位数只能指定为常数。

●返回值

返回值名	内容
ANY_BIT	被移动的数据 备注)从最高位算起的n位变为0。

●使用示例

自变量型	ST程序	转换结果	使用指令
INT	D0 := SHR(D1,1);	LD SM400 MOV D1 D0 SFR D0 K1	LD,MOV,SFR

■函数定义

```
BOOL SHR_E( BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	移动的数据
n	IN	移动的位数 备注)移动的位数只能指定为常数。
D1	OUT	被移动的数据 备注)从最高位算起的n位变为0。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的数据向右移动2位，                    *)
(*并将其结果存储到D100中。                                                *)
    M0:=SHR_E(X0, D0, 2, D100);
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.6.3 右旋转 ROR
ROR_E

向右侧旋转n位。

■函数定义

```
ANY_BIT ROR( ANY_BIT S1, ANY_BIT n );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	旋转数据
n	IN	旋转位数 注)旋转位数只能指定为常数。

●返回值

返回值名	内容
ANY_BIT	旋转结果数据

●使用示例

自变量型	ST程序	转换结果	使用指令
INT	D0 := ROR(D1,1);	LD SM400 MOV D1 D0 ROR D0 K1	LD,MOV,ROR

■函数定义

```
BOOL ROR_E( BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	旋转数据
n	IN	旋转位数 备注)旋转位数只能指定为常数。
D1	OUT	旋转结果数据

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的数据向右旋转1位                    *)
(*并将结果存储到D100中。                                              *)
    M0:=ROR_E( X0, D0, 1, D100 );
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.6.4 左旋转 ROL
ROL_E

向左侧旋转n位。

■函数定义

```
ANY_BIT ROL( ANY_BIT S1, ANY_BIT n );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	旋转数据
n	IN	旋转位数 备注)旋转位数只能指定为常数。

●返回值

返回值名	内容
ANY_BIT	旋转结果数据

●使用示例

自变量型	ST程序	转换结果	使用指令
INT	D0 := ROL(D1,1);	LD SM400 MOV D1 D0 ROL D0 K1	LD,MOV,ROL

■函数定义

```
BOOL ROL_E( BOOL EN, ANY_BIT S1, ANY_BIT n, ANY_BIT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	旋转数据
n	IN	旋转位数 备注)旋转位数只能指定为常数。
D1	OUT	旋转结果数据

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将D0中存储的数据向左旋转1位                    *)
(*并将结果存储到D100中。                                                *)
    M0:=ROL_E( X0, D0, 1, D100 );
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.7 位型布尔功能

6.7.1 逻辑积 AND_E

对指定的多个数据进行逻辑积运算。

■函数定义

```
BOOL  AND_E( BOOL EN, ANY_BIT S1, ANY_BIT S2,….., ANY_BIT Sn, ANY_BIT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	进行逻辑积运算的数据
D1	OUT	逻辑积运算结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
BOOL	b_result := AND_E(b_select, b_data1, b_data2, b_data3, b_data4);	LD b_data1 AND b_data2 AND b_data3 OUT M8191 LD b_select AND M8191 SET b_data4 LD b_select ANI M8191 RST b_data4 LD b_select OUT b_result	LD,AND,OUT,SET,ANI,RST
字软元件	b_result := AND_E(b_select, d0, d1, d2, d3);	LD b_select WAND D0 D1 D10239 WAND D10239 D2 D3 LD b_select OUT b_result	LD, WAND, OUT

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 43

6.7.2 逻辑和 OR_E

将指定的多个数据进行逻辑和运算。

■函数定义

```
BOOL OR_E( BOOL EN, ANY_BIT S1, ANY_BIT S2,…., ANY_BIT Sn, ANY_BIT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	进行逻辑和运算的数据
D1	OUT	逻辑和运算结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
BOOL	b_result := OR_E(TRUE, b_data1, b_data2, b_data3);	LD b_data1 OR b_data2 OUT M8191 LD SM400 AND M8191 SET b_data3 LD SM400 ANI M8191 RST b_data3 LD SM400 OUT b_result	LD,OR,OUT,AND,SET, ANI,RST
字软元件	B1 := OR_E(TRUE, D0, D1, D2);	LD SM400 WOR D0 D1 D2 LD SM400 OUT B1	LD, WOR, OUT

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 44

6.7.3 排他逻辑和 XOR_E

对指定的多个数据进行排他逻辑和运算。

■函数定义

BOOL XOR_E(BOOL EN, ANY_BIT S1, ANY_BIT S2,…., ANY_BIT Sn, ANY_BIT D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	进行排他逻辑和运算的数据
D1	OUT	排他逻辑和运算结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
BOOL	b_result := XOR_E(b_select, b_data1, b_data2, b_data3);	LD b_data1 ANI b_data2 LDI b_data1 AND b_data2 ORB OUT M8191 LD b_select AND M8191 SET b_data3 LD b_select ANI M8191 RST b_data3 LD b_select OUT b_result	LD,ANI,LDI,AND,ORB, OUT,SET,RST
字软元件	b_result := XOR_E(TRUE, d0z2, d1z3, d2z4);	LD SM400 WXOR D0Z2 D1Z3 D2Z4 LD SM400 OUT b_result	LD,WXOR, OUT

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 45

6.7.4 逻辑否定 NOT
NOT_E

对指定的数据进行逻辑否定运算。

■函数定义 ANY_BIT NOT(ANY_BIT S1);

●自变量

自变量名	IN/OUT	内容
S1	IN	进行逻辑否定运算的数据

●返回值

返回值名	内容
ANY_BIT	逻辑否定运算结果

●使用示例

自变量型	ST程序	转换结果	使用指令
BOOL	b_result := NOT(b_data1);	LDI b_data1 OUT b_result	LDI, OUT
字软元件	d0z2 := NOT(d1z3);	LD SM400 CML D1Z3 D0Z2	LD, CML

■函数定义 BOOL NOT_E(BOOL EN, ANY_BIT S1, ANY_BIT D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	进行逻辑否定运算的数据
D1	OUT	逻辑否定运算结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则计算处D0中存储的数据的逻辑否定，          *)
(*并将结果存储到D100中。                                        *)
    M0:=NOT_E( X0, D0, D100 );
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6.8 选择功能

6.8.1 二进制的选择 SEL
SEL_E

根据选择条件从指定的2个数据中选择1个数据。

■函数定义

```
ANY SEL( BOOL S1, ANY S2, ANY S3 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	选择条件
S2	IN	S1为FALSE的情况下选择的数据
S3	IN	S1为TRUE的情况下选择的数据

●返回值

返回值名	内容
ANY	选择结果 S1为FALSE的情况下・・・返回值=S2 S1为TRUE的情况下 ・・・返回值=S3

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	r_data1 := SEL(b_select, r_data2, r_data3);	LDI b_select EMOV r_data2 r_data1 LD b_select EMOV r_data3 r_data1	LDI, EMOV, LD,
INT	D1 := SEL(X1, D2, D3);	LDI X1 MOV D2 D1 LD X1 MOV D3 D1	LDI, MOV, LD
DINT	K8X100 := SEL(X1, K8X10, K2147483647);	LDI X1 DMOV K8X10 K8X100 LD X1 DMOV K2147483647 K8X100	LDI, DMOV, LD
BOOL	b_result := SEL(b_select, b_data1, b_data2);	LDI b_select MPS AND b_data1 SET b_result MPP ANI b_data1 RST b_result LD b_select MPS AND b_data2 SET b_result MPP ANI b_data2 RST b_result	LDI, MPS,AND, SET, MPP, ANI, RST,LD
STRING	s_result := SEL(b_select, s_ary1, s_ary2);	LDI b_select $MOV s_ary1 s_result LD b_select $MOV s_ary2 s_result	LDI, $MOV,LD

6 - 47

■函数定义

```
BOOL SEL_E( BOOL EN, BOOL S1, ANY S2, ANY S3, ANY D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	选择条件
S2	IN	S1为FALSE的情况下选择的数据
S3	IN	S1为TRUE的情况下选择的数据
D1	OUT	选择结果 S1为FALSE的情况下・・・返回值=S2 S1为TRUE的情况下 ・・・返回值=S3

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则bData的位数据为FALSE的情况下将iData1中          *)
(*存储的数据存储到Result中，bData的位数据为TRUE的情况下将iData2中          *)
(*存储的数据存储到Result中。                                               *)
     M0 := SEL_E( X0, bData, iData1, iData2, Result ) ;
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 48

6.8.2 最大值 MAX
MAX_E

从指定的数据中查找最大值。

■函数定义

```
ANY_SIMPLE MAX( ANY_SIMPLE S1, ANY_SIMPLE S2,….., ANY_SIMPLE Sn );
```

●自变量

自变量名	IN/OUT	内容
S1～Sn	IN	查找对象数据

●返回值

返回值名	内容
ANY_SIMPLE	查找结果

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	w_Real4 := MAX(w_Real1,w_Real2,w_Real3);	LD SM400 EMOV w_Real1 w_Real4 LDE< w_Real4 w_Real2 EMOV w_Real2 w_Real4 LDE< w_Real4 w_Real3 EMOV w_Real3 w_Real4	LD,EMOV,LDE<
INT	D0 := MAX(D1,D2,D3);	LD SM400 MOV D1 D0 LD< D0 D2 MOV D2 D0 LD< D0 D3 MOV D3 D0	LD,MOV,LD<
DINT	w_DWord4 := MAX(-2147483648,0,2147483647);	LD SM400 DMOV K2147483647 w_DWord4	LD,DMOV
BOOL	w_Bit4 := MAX(w_Bit1,w_Bit2,w_Bit3);	LD w_Bit1 OR w_Bit2 OR w_Bit3 OUT w_Bit4	LD,OR,OUT
STRING	w_Str4 := MAX("ABC","DEF","GHI");	LD SM400 $MOV "ABC" w_Str4 LD$< w_Str4 "DEF" $MOV "DEF" w_Str4 LD$< w_Str4 "GHI" $MOV "GHI" w_Str4	LD,MOV,LD<

■函数定义

BOOL MAX_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,….., ANY_SIMPLE Sn, ANY_SIMPLE D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	查找对象数据
D1	OUT	查找结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从iData1、iData2及iData3中存储的数据中查找      *)
(*最大值数据，并将结果存储到Result中。                                      *)
    M0 := MAX_E( X0, iData1, iData2, iData3, Result ) ;
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 50

6.8.3 最小值 MIN
MIN_E

从指定的数据中查找最小值。

■函数定义

```
ANY_SIMPLE MIN( ANY_SIMPLE S1, ANY_SIMPLE S2,….., ANY_SIMPLE Sn );
```

●自变量

自变量名	IN/OUT	内容
S1～Sn	IN	查找对象数据

●返回值

返回值名	内容
ANY_SIMPLE	查找结果

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	Real4:= MIN(Real1,Real2,Real3);	LD SM400 EMOV Real1 Real4 LDE> Real4 Real2 EMOV Real2 Real4 LDE> Real4 Real3 EMOV Real3 Real4	LD,EMOV,LDE>
INT	Int4:= MIN(Int1,Int2,Int3);	LD SM400 MOV Int1 Int4 LD> Int4 Int2 MOV Int2 Int4 LD> Int4 Int3 MOV Int3 Int4	LD,MOV,LD>
DINT	Dint4:= MIN(Dint1,Dint2,Dint3);	LD SM400 DMOV Dint1 Dint4 LDD> Dint4 Dint2 DMOV Dint2 Dint4 LDD> Dint4 Dint3 DMOV Dint3 Dint4	LD,DMOV,LDD>
BOOL	bBit4:= MIN(bBit1,bBit2,bBit3);	LD bBit1 AND bBit2 AND bBit3 OUT bBit4	LD,AND,OUT
STRING	Str4:= MIN(Str1,Str2,Str3);	LD SM400 $MOV Str1 Str4 LD$> Str4 Str2 $MOV Str2 Str4 LD$> Str4 Str3 $MOV Str3 Str4	LD,MOV,LD>

■函数定义

BOOL MIN_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,…., ANY_SIMPLE Sn, ANY_SIMPLE D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	查找对象数据
D1	OUT	查找结果

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从iData1、iData2及iData3中存储的数据中          *)
(*将最小值存储到Result中。                                                *)
    M0 := MIN_E( X0, iData1, iData2, iData3, Result ) ;
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 52

6.8.4 限制器　LIMIT
LIMIT_E

根据指定的数据是否在上下限极限值(最小・最大输出界限值)的范围内对输出值进行控制。

■函数定义

```
ANY_SIMPLE LIMIT( ANY_SIMPLE MIN, ANY_SIMPLE S1, ANY_SIMPLE MAX );
```

●自变量

自变量名	IN/OUT	内容
MIN	IN	最小输出界限值
S1	IN	输入值
MAX	IN	最大输出界限值

●返回值

返回值名	内容
ANY_SIMPLE	输出值 输出值MIN(下限值)＞S1(输入值) 的情况下 ・・・返回值=MIN(下限值) MAX(上限值)＜S1(输入值) 的情况下 ・・・返回值=MAX(上限值) MIN(下限值)≦S1(输入值)≦MAX(上限值)的情况下 ・・・返回值=S1(输入值)

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	Real4:= LIMIT(Real1,Real2, Real3);	LDE>= Real2 Real1 ANDE<= Real2 Real3 EMOV Real2 Real4 LDE< Real2 Real1 EMOV Real1 Real4 LDE> Real2 Real3 EMOV Real3 Real4	LDE>=,ANDE<=,EMOV, LDE<,LDE>
INT	Int4:= LIMIT(Int1,Int2,In t3);	LD SM400 LIMIT Int1 Int3 Int2 Int4	LD,LIMIT
DINT	Dint4:= LIMIT(Dint1,Dint2, Dint3);	LD SM400 DLIMIT Dint1 Dint3 Dint2 Dint4	LD,DLIMIT
BOOL	bBit4:= LIMIT(bBit1,bBit2,b Bit3);	LD bBit2 OR bBit1 AND bBit3 OUT bBit4	LD,OR,AND,OUT
STRING	Str4:= LIMIT(Str1,Str2,Str 3);	LD$>= Str2 Str1 AND$<= Str2 Str3 $MOV Str2 Str4 LD$< Str2 Str1 $MOV Str1 Str4 LD$> Str2 Str3 $MOV Str3 Str4	LD$>=,AND$<=,MOV, LD<,LD$>

■函数定义

BOOL LIMIT_E(BOOL EN, ANY_SIMPLE MIN, ANY_SIMPLE S1, ANY_SIMPLE MAX, ANY_SIMPLE D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
MIN	IN	最小输出界限值
S1	IN	输入值
MAX	IN	最大输出界限值
D1	OUT	输出值 输出值MIN(下限值)＞S1(输入值) 的情况下 ・・・D1=MIN(下限值) MAX(上限值)＜S1(输入值)的情况下 ・・・D1=MAX(上限值) MIN(下限值)≦S1(输入值)≦MAX(上限值)的情况下 ・・・D1=S1(输入值)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*执行条件X0变为ON时，如果iData2的数据小于最小值iData1的数据          *)
(*则将iData1的值存储到Result中；如果iData2的数据大于最大值          *)
(*iData3的数据则将iData3的值存储到Result中；除上述两种情况          *)
(*以外时则将iData2的值存储到Result中。                              *)
 M0 := LIMIT_E( X0, iData1, iData2, iData3, Result ) ;
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 54

6.8.5 多路调制器 MUX
MUX_E

根据指定的选择条件，从指定的数据中选择一个。

■函数定义 ANY MUX(INT n, ANY S1, ANY S2,….,ANY Sn);

●自变量

自变量名	IN/OUT	内容
n	IN	选择条件
S1～Sn	IN	选择对象数据

●返回值

返回值名	内容
ANY	选择结果 n=1的情况下 返回值=S1 n=2的情况下 返回值=S2 : : n=n的情况下 返回值=Sn

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	Real4 := MUX(Int1, Real1,Real2, Real3);	LD= Int1 K1 EMOV Real1 Real4 LD= Int1 K2 EMOV Real2 Real4 LD= Int1 K3 EMOV Real3 Real4	LD=,EMOV
INT	Int4:= MUX(wCon1 , Int1 , Int2, Int3);	LD= wCon1 K1 MOV Int1 Int4 LD= wCon1 K2 MOV Int2 Int4 LD= wCon1 K3 MOV Int3 Int4	LD=,MOV
DINT	Dint4:= MUX(D0, Dint1,Dint2,Dint3) ;	LD= D0 K1 DMOV Dint1 Dint4 LD= D0 K2 DMOV Dint2 Dint4 LD= D0 K3 DMOV Dint3 Dint4	LD=,DMOV
BOOL	bBit4:= MUX(3,bBit1,bBit2,b Bit3);	LD= K3 K1 MPS AND bBit1 SET bBit4 MPP ANI bBit1 RST bBit4 LD= K3 K2 MPS AND bBit2 SET bBit4 MPP ANI bBit2 RST bBit4 LD= K3 K3 MPS AND bBit3 SET bBit4 MPP ANI bBit3 RST bBit4	LD=,MPS,AND,SET, MPP,ANI,RST

■函数定义

BOOL MUX_E(BOOL EN, INT n, ANY S1, ANY S2,….,ANY Sn, ANY D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
n	IN	选择条件
S1～Sn	IN	选择对象数据
D1	OUT	选择结果 n=1的情况下　D1=S1 n=2的情况下　D1=S2 ：　　： n=n的情况下　D1=Sn

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则根据iData1的数据进行判断，从iData2、iData3、     *)
(*iData4及iData5中存储的数据内选择一个将其存储到Result中。                 *)
     M0 := MUX_E( X0, iData1, iData2, iData3, iData4, iData5, Result );
```

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 56

6.9 比较功能

6.9.1 大于号(>) GT_E

在指定的所有数据中，对＞(大于)关系是否成立进行获取。

■函数定义

BOOL GT_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,….,ANY_SIMPLE Sn,BOOL D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	比较对象数据
D1	OUT	比较结果

备注)D1 = (S1＞S2) &(S2＞S3)&・・・&(Sn-1＞Sn)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	GT_E(M0 , Real1, Real2, Real3, bBit1);	LDE> Real1 Real2 ANDE> Real2 Real3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LDE>,ANDE>,OUT, LD,AND,SET,ANI, RST
INT	GT_E(M0 , Int1, Int2, Int3, bBit1);	LD> Int1 Int2 AND> Int2 Int3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD>,AND>,OUT,LD, AND,SET,ANI,RST
DINT	GT_E(M0 , Dint1, Dint2 , Dint3, bBit1);	LDD> Dint1 Dint2 ANDD> Dint2 Dint3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LDD>,ANDD>,OUT LD,AND,SET,ANI, RST

自变量型	ST程序	转换结果		使用指令
BOOL	GT_E(M0 , M100, M101, M102, M103, bBit1);	LD	M100	LD,ANI,ANB,OUT, AND,SET,RST
		ANI	M101	
		LD	M101	
		ANI	M102	
		ANB		
		LD	M102	
		ANI	M103	
		ANB		
		OUT	M8191	
		LD	M0	
		AND	M8191	
		SET	bBit1	
		LD	M0	
		ANI	M8191	
		RST	bBit1	
STRING	GT_E(M0 , Str1, Str2 , Str3, bBit1);	LD$>	Str1 Str2	LD$>,AND$>,OUT LD,AND,SET,ANI RST
		AND$>	Str2 Str3	
		OUT	M8191	
		LD	M0	
		AND	M8191	
		SET	bBit1	
		LD	M0	
		ANI	M8191	
		RST	bBit1	

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 58

6.9.2 大于等于号(>=) GE_E

在指定的所有数据中，对≧(大于等于号)关系是否成立进行获取。

■函数定义

BOOL GE_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,….,ANY_SIMPLE Sn,BOOL D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	比较对象数据
D1	OUT	比较结果

备注)D1 = (S1≧S2) &(S2≧S3)&・・・&(Sn-1≧Sn)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	GE_E(M0 , Real1, Real2, Real3, bBit1);	LDE>= Real1 Real2 ANDE>= Real2 Real3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LDE>=,ANDE>=, OUT,LD,AND,SET, ANI,RST
INT	GE_E(M0 , Int1, Int2, Int3, bBit1);	LD>= Int1 Int2 AND>= Int2 Int3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD>=,AND>=,OUT LD,AND,SET,ANI, RST
DINT	GE_E(M0 , Dint1, Dint2 , Dint3, bBit1);	LDD>= Dint1 Dint2 ANDD>= Dint2 Dint3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LDD>=,ANDD>=, OUT,LD,AND,SET ANI,RST

自变量型	ST程序	转换结果	使用指令
BOOL	GE_E(M0 , M100, M101, M102, M103, bBit1);	LD M100 ORI M101 LD M101 ORI M102 ANB LD M102 ORI M103 ANB OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD,ORI,ANB,OUT AND,SET,ANI,RST
STRING	GE_E(M0 , Str1, Str2 , Str3, bBit1);	LD$>= Str1 Str2 AND$>= Str2 Str3 OUT M8191 LD M0 AND M8191 SET bBit1 LD M0 ANI M8191 RST bBit1	LD$>=,AND$>=, OUT,LD,AND,SET, LD,ANI,RST

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 60

6.9.3 等号(=) EQ_E

在指定的所有数据中，对=(等号)关系是否成立进行获取。

■函数定义

BOOL EQ_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,….,ANY_SIMPLE Sn,BOOL D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	比较对象数据
D1	OUT	比较结果

备注)D1 = (S1=S2) &(S2=S3)&・・・&(Sn-1=Sn)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := EQ_E(b_select, r_data1, r_data2, r_data3, b_data1);	LDE= r_data1 r_data2 ANDE= r_data2 r_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST data1 LD b_select OUT b_result	LDE=,ANDE=,OUT LD,AND,SET,ANI, RST
INT	B100 := EQ_E(M20, D10, D20, D30, M200);	LD= D10 D20 AND= D20 D30 OUT M8191 LD M20 AND M8191 SET M200 LD M20 ANI M8191 RST M200 LD M20 OUT B100	LD=,AND=,OUT LD,AND,SET,ANI RST
DINT	b_result := EQ_E(b_select, di_data1, di_data2, di_data3, b_data1);	LDD= di_datal di_data2 ANDD= di_data2 di_data3 OUT M8191 LD b_select AND M8191 SET b_datal LD b_select ANI M8191 RST b_datal LD b_select OUT b_result	LDD=,ANDD=,OUT, LD,AND,SET,ANI, RST

自变量型	ST程序	转换结果		使用指令
BOOL	b_result := EQ_E(b_select, X10, X11, X12, M20);	LD AND LDI ANI ORB LD AND LDI ANI ORB ANB OUT LD AND SET LD ANI RST LD OUT	X10 X11 X10 X11 X11 X12 X11 X12 M8191 b_select M8191 M20 b_select M8191 M20 b_select b_result	LD,AND,LDI,ANI, ORB,ANB,SET,RST
STRING	b_result := EQ_E(b_select, s_ary1, s_ary2, b_data1);	LD$= OUT LD AND SET LD ANI RST LD OUT	s_ary1 s_ary2 M8191 b_select M8191 b_data1 b_select M8191 b_data1 b_select b_result	LD$=,OUT,LD,AND, SET,ANI,RST

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 62

6.9.4 小于等于号(<=) LE_E

在指定的所有数据中，对≦(小于等于号)关系是否成立进行获取。

■函数定义

BOOL LE_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,….,ANY_SIMPLE Sn,BOOL D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	比较对象数据
D1	OUT	比较结果

备注)D1 = (S1≦S2) &(S2≦S3)&・・・&(Sn-1≦Sn)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := LE_E(b_select, r_data1, r_data2, r_data3, b_data1);	LDE<= r_data1 r_data2 ANDE<= r_data2 r_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDE<=,ANDE<=, OUT,LD,AND,SET, ANI,RST
INT	B100 := LE_E(M20, D10, D20, D30, M200);	LD<= D10 D20 AND<= D20 D30 OUT M8191 LD M20 AND M8191 SET M200 LD M20 ANI M8191 RST M200 LD M20 OUT B100	LD<=,AND<=,OUT LD,AND,SET,ANI, RST

自变量型	ST程序	转换结果	使用指令
DINT	b_result := LE_E(b_select, di_data1, di_data2, di_data3, b_data1);	LDD<= di_data1 di_data2 ANDD<= di_data2 di_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDD<=,ANDD<=, OUT,LD,AND,SET, ANI,RST
BOOL	b_result := LE_E(b_select, X10, X11, X12, M20);	LDI X10 OR X11 LDI X11 OR X12 ANB OUT M8191 LD b_select AND M8191 SET M20 LD b_select ANI M8191 RST M20 LD b_select OUT b_result	LDI,OR,ANB,OUT, AND,SET,ANI,RST
STRING	b_result := LE_E(b_select, s_ary1, s_ary2, b_data1);	LD$<= s_ary1 s_ary2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LD$<=,OUT,LD, AND,SET,ANI,RST

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 64

6.9.5 小于号(<)　LT_E

在指定的所有数据中，对＜(小于号)的关系是否成立进行获取。

■函数定义

BOOL LT_E(BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2,….,ANY_SIMPLE Sn,BOOL D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	比较对象数据
D1	OUT	比较结果

备注)D1 = (S1＜S2) &(S2＜S3)&・・・&(Sn-1＜Sn)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := LT_E(b_select, r_data1, r_data2, r_data3, b_data1);	LDE< r_data1 r_data2 ANDE< r_data2 r_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDE<,ANDE<,OUT, LD,AND,SET,ANI, RST
INT	B100 := LT_E(M20, D10, D20, D30, M200);	LD< D10 D20 AND< D20 D30 OUT M8191 LD M20 AND M8191 SET M200 LD M20 ANI M8191 RST M200 LD M20 OUT B100	LD<,AND<,OUT, LD,SET,ANI,RST

自变量型	ST程序	转换结果	使用指令
DINT	b_result := LT_E(b_select, di_data1, di_data2, di_data3, b_data1);	LDD< di_data1 di_data2 ANDD< di_data2 di_data3 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDD<,ANDD<,OUT, LD,AND,SET,ANI, RST
BOOL	b_result := LT_E(b_select, X10, X11, X12, M20);	LDI X10 AND X11 LDI X11 AND X12 ANB OUT M8191 LD b_select AND M8191 SET M20 LD b_select ANI M8191 RST M20 LD b_select OUT b_result	LDI,AND,ANB,OUT, LD,SET,ANI,RST
STRING	b_result := LT_E(b_select, s_ary1, s_ary2, b_data1);	LD$< s_ary1 s_ary2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LD$<,OUT,LD,AND, SET,ANI,RST

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 66

6.9.6 不等号(<>) NE_E

在指定的2个数据中，对≠(不等号)关系是否成立进行获取。

■函数定义

```
BOOL NE_E( BOOL EN, ANY_SIMPLE S1, ANY_SIMPLE S2, BOOL D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	比较对象数据
S2	IN	比较对象数据
D1	OUT	比较结果

备注) D1=(S1≠S2)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

自变量型	ST程序	转换结果	使用指令
REAL	b_result := NE_E(b_select, r_data1, r_data2, b_data1);	LDE<> r_data1 r_data2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDE<>,OUT,LD, AND,SET,ANI,RST
INT	B100 := NE_E(M20, D10, D20, M200);	LD<> D10 D20 OUT M8191 LD M20 AND M8191 SET M200 LD M20 ANI M8191 RST M200 LD M20 OUT B100	LD<>,OUT,LD,AND, SET,ANI,RST

自变量型	ST程序	转换结果	使用指令
DINT	b_result := NE_E(b_select, di_data1, di_data2, b_data1);	LDD<> di_data1 di_data2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LDD<>,OUT,LD,AND, SET,ANI,RST
BOOL	b_result := NE_E(b_select, X10, X11, M20);	LD X10 ANI X11 LDI X10 AND X11 ORB OUT M8191 LD b_select AND M8191 SET M20 LD b_select ANI M8191 RST M20 LD b_select OUT b_result	LD,ANI,LDI,AND, ORB,OUT,SET,RST
STRING	b_result := NE_E(b_select, s_ary1, s_ary2, b_data1);	LD$<> s_ary1 s_ary2 OUT M8191 LD b_select AND M8191 SET b_data1 LD b_select ANI M8191 RST b_data1 LD b_select OUT b_result	LD$<>,OUT,LD,AND, SET,ANI,RST

关于可用数据类型请参阅“3.2.2 关于ANY类型”。

6 - 68

6.10 字符串功能

6.10.1 字符串长度截取 LEN
LEN_E

对指定的字符串数据的字符串长进行获取。

■函数定义

```
INT LEN( STRING S1 );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	获取字符串长的数据(字符串数据)

●返回值

返回值名	内容
INT	字符串长结果(BIN16位数据)

备注) 本函数不能在基本型QCPU中使用。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	i_data1 := LEN(s_ary1);	LD SM400 LEN s_ary1 i_data1	LD,LEN

■函数定义

```
BOOL LEN_E( BOOL EN, STRING S1, INT D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	获取字符串长的数据(字符串数据)
D1	OUT	字符串长结果(BIN16位数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则对sData中存储的字符串的长度进行获取          *)
(*并将结果存储到Result中。                                          *)
     M0 := LEN_E( X0, sData, Result ) ;
```

6.10.2 从字符串的开始位置截取 LEFT
LEFT_E

从指定的字符串的左侧(字符串的起始)开始对指定的n字符的字符串进行截取。

■函数定义

```
STRING LEFT( STRING S1, INT n );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	截取的数据(字符串数据)
n	IN	截取的字符数(BIN16位数据)

●返回值

返回值名	内容
STRING	截取结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。
对于存储截取的字符串数据的数据区域，应预留n+1字符的容量。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	s_ary1 := LEFT(s_ary2, i_data1);	LD SM400 LEFT s_ary2 s_ary1 i_data1	LD,LEFT

■函数定义

```
BOOL LEFT _E( BOOL EN, STRING S1, INT n, STRING D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	截取的数据(字符串数据)
n	IN	截取的字符数(BIN16位数据)
D1	OUT	截取结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData中存储的字符串数据的左侧开始，          *)
(*对iData中指定的字符数的字符串进行截取并将其存储到Result中。          *)
    M0 := LEFT_E( X0, sData, iData, Result ) ;
```

6.10.3 从字符串的终端截取 RIGHT RIGHT_E

从指定的字符串的右侧(字符串的最终)开始对指定的n字符的字符串进行截取。

■函数定义

```
STRING RIGHT( STRING S1, INT n );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	截取的字符串数据(字符串数据)
n	IN	截取的字符数(BIN16位数据)

●返回值

返回值名	内容
STRING	获取结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。
对于存储截取的字符串数据的数据区域，应预留n+1字符的容量。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	s_ary1 := RIGHT(s_ary2, i_data1);	LD SM400 RIGHT s_ary2 s_ary1 i_data1	LD,RIGHT

■函数定义

```
BOOL RIGHT _E( BOOL EN, STRING S1, INT n, STRING D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	截取的字符串数据(字符串数据)
n	IN	截取的字符数(BIN16位数据)
D1	OUT	截取结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData中存储的字符串的右侧开始对iData中      *)
(*指定的字符数的字符串进行截取并将其存储到Result中。                    *)
    M0 := RIGHT_E( X0, sData, iData, Result ) ;
```

6.10.4 从字符串的指定位置截取 MID
MID_E

从指定的字符串数据的左侧(字符串的起始)开始，按照指定的位置对指定的n字符的字符串数据进行截取。

■函数定义

```
STRING MID( STRING S1, INT n, INT POS );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	截取的字符串数据(字符串数据)
n	IN	截取的字符数(BIN16位数据)
POS	IN	截取的数据的起始位置(BIN16位数据)

●返回值

返回值名	内容
STRING	截取结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。
对于存储截取的字符串数据的数据区域，应预留n+1字符的容量。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	s_ary1 := MID(s_ary2, i_data1, i_data2);	LD SM400 MOV i_data1 D10239 MOV i_data2 D10238 MIDR s_ary2 s_ary1 D10238	LD,MOV,MIDR

■函数定义

```
BOOL MID_E( BOOL EN, STRING S1, INT n, INT POS, STRING D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	截取的字符串数据(字符串数据)
n	IN	截取的字符数(BIN16位数据)
POS	IN	截取的数据的起始位置(BIN16位数据)
D1	OUT	截取结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData中存储的字符串的起始算起第iData2个字符*)
(*位置开始对iData1中存储的字符数的字符串进行截取并存储到Result中。      *)
    M0 := MID_E( X0, sData, iData1, iData2, Result ) ;
```

6.10.5 字符串的连接 CONCAT
CONCAT_E

将指定的字符串数据全部进行连接。

■函数定义

```
STRING CONCAT( STRING S1, STRING S2,….,STRING Sn );
```

●自变量

自变量名	IN/OUT	内容
S1～Sn	IN	连接的数据(字符串数据)

●返回值

返回值名	内容
STRING	连接结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。
对于存储连接的字符串数据的数据区域，应预留连接的字符数+1字符的容量。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	s_result := CONCAT(s_ary1, s_ary2, s_ary3);	LD SM400 $MOV s_ary1 s_result $+ s_ary2 s_result $+ s_ary3 s_result	LD,$MOV,$+

■函数定义

```
BOOL CONCAT_E( BOOL EN, STRING S1, STRING S2,….,STRING Sn, STRING D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1～Sn	IN	连接的数据(字符串数据)
D1	OUT	连接结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则将sData1、sData2中存储的字符串数据进行      *)
(*连接并将结果存储到Result中。                                        *)
    M0 := CONCAT_E( X0, sData1, sData2, Result ) ;
```

6.10.6 至指定位置的字符串插入 INSERT
INSERT_E

在指定的字符串数据的指定位置后面插入字符串数据。

■函数定义

```
STRING INSERT( STRING S1, STRING S2, INT POS );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	被插入的数据(字符串数据)
S2	IN	插入数据(字符串数据)
POS	IN	插入位置(BIN16位数据)

●返回值

返回值名	内容
STRING	插入结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。
对于存储插入后的字符串数据的数据区域，应预留出插入后的字符数+1字符的容量。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	w_Str3 := INSERT(w_Str1,w_Str2,w_Word1);	LD SM400 $+ w_Str2 w_Str1 w_Str3 AND<> w_Word1 K1 MOV K1 D10238 - w_Word1 K1 D10239 MIDW w_Str1 w_Str3 D10238 MOV w_Word1 D10238 LEN w_Str2 D10239 MIDW w_Str2 w_Str3 D10238	LD,$+,AND<>,MOV,-,MIDW,LEN

■函数定义

```
BOOL INSERT_E( BOOL EN, STRING S1, STRING S2, INT POS, STRING D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被插入的数据(字符串数据)
S2	IN	插入数据(字符串数据)
POS	IN	插入位置(BIN16位数据)
D1	OUT	插入结果(字符串数据)

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则在sData1的字符串数据的起始算起第iData个          *)
(*字符位置处插入sData2的字符串数据并将结果存储到Result中。              *)
     M0 := INSERT_E( X0, sData1, sData2, iData, Result ) ;
```

6.10.7 从字符串的指定位置删除 DELETE
DELETE_E

从指定的字符串的指定位置开始删除n字符的字符串。

■函数定义

```
STRING DELETE( STRING S1, INT n, INT POS );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	被删除的数据(字符串数据)
n	IN	删除字符数(BIN16位数据)
POS	IN	删除位置(BIN16位数据)

●返回值

返回值名	内容
STRING	删除结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。
对于存储删除后的字符串数据的数据区域,应预留出删除后的字符数+1字符的容量。
在删除位置POS为0的情况下，被删除的数据S1的后面(右侧)算起n字符的字符串将被删除。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	w_Str2 := DELETE(w_Str1,w_Word1, w_Word2);	LD SM400 LEN w_Str1 D10238 - w_Word1 D10238 RIGHT w_Str1 w_Str2 D10238 - w_Word2 K1 D10239 AND<> 10239 K0 MOV K1 D10238 MIDW w_Str1 w_Str2 D10238	LD,LEN,-,RIGHT, AND<>,MOV,MIDW

■函数定义

```
BOOL DELETE_E( BOOL EN, STRING S1, INT n, INT POS, STRING D1 );
```

●自变量

变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被删除的数据(字符串数据)
n	IN	删除字符数(BIN16位数据)
POS	IN	删除位置(BIN16位数据)
D1	OUT	删除结果(字符串数据)

备注) 对于存储删除后的字符串数据的数据区域,应预留出删除后的字符数+1字符的容量。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*果执行条件X0变为ON，则从sData的字符串数据的起始算起第iData2个字符开  *)
(*始删除iData1中指定的字符数的字符串，并将结果存储到Result中。          *)
    M0 := DELETE_E( X0, sData, iData1, iData2, Result ) ;
```

6.10.8 从字符串的指定位置替换 REPLACE
REPLACE_E

将指定的字符串数据的指定位置算起n字符的字符串数据替换为指定的字符串。

■函数定义

```
STRING REPLACE( STRING S1, STRING S2, INT n, INT POS );
```

●自变量

自变量名	IN/OUT	内容
S1	IN	被替换的数据(字符串数据)
S2	IN	替换的数据(字符串数据)
n	IN	替换字符数(BIN16位数据)
POS	IN	替换开始位置(BIN16位数据)

●返回值

返回值名	内容
STRING	替换结果(字符串数据)

备注) 本函数不能在基本型QCPU中使用。
对于存储替换后的字符串数据的数据区域,应预留出替换后的字符数+1字符的容量。

●使用示例

自变量型	ST程序	转换结果		使用指令
STRING	w_Str3 := REPLACE(w_Str1,w_Str2, w_Word1, w_Word2);	LD $MOV MOV MOV MIDW 	SM400 w_Str1 w_Str3 w_Word1 D10239 w_Word2 D10238 w_Str2 w_Str3 D10238	LD,$MOV,MOV, MIDW

■函数定义

```
BOOL REPLACE_E( BOOL EN, STRING S1, STRING S2, INT n, INT POS, STRING D1 );
```

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被替换的数据(字符串数据)
S2	IN	替换的数据(字符串数据)
n	IN	替换字符数(BIN16位数据)
POS	IN	替换开始位置(BIN16位数据)
D1	OUT	替换结果(字符串数据)

备注) 对于存储替换后的字符串数据的数据区域,应预留出替换后的字符数+1字符的容量。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData1的字符串数据的起始算起第iData2个      *)
(*字符开始，将Data1中指定的字符数的字符串数据用sData2的字符串数据进行  *)
(*替换，并将结果存储到Result中。                                        *)
    M0 := REPLACE_E( X0, sData1, sData2, iData1, iData2, Result ) ;
```

6.10.9 从字符串的指定位置查找 FIND
FIND_E

从指定的字符串中查找指定的字符串。

■函数定义

INT FIND(STRING S1, STRING S2);

●自变量

自变量名	IN/OUT	内容
S1	IN	被进行查找的数据(字符串数据)
S2	IN	进行查找的数据(字符串数据

●返回值

返回值名	内容
INT	最先被查找到的查找位置(BIN16位数据)

备注) 本函数不能在基本型QCPU中使用。
查找不到的情况下，返回值将变为0。

●使用示例

自变量型	ST程序	转换结果	使用指令
STRING	w_Word1:= FIND(w_Str1,w_Str2);	LD SM400 INSTR w_Str2 w_Str1 w_Word1 K1	LD,INSTR

■函数定义

BOOL FIND_E(BOOL EN, STRING S1, STRING S2, INT D1);

●自变量

自变量名	IN/OUT	内容
EN	IN	执行条件(仅TRUE时执行函数)
S1	IN	被进行查找的数据(字符串数据)
S2	IN	进行查找的数据(字符串数据)
D1	OUT	最先被查找到的查找位置(BIN16位数据)

备注) 查找不到的情况下，返回值将变为0。

●返回值

返回值名	内容
BOOL	执行条件

●使用示例

```
(*如果执行条件X0变为ON，则从sData1的字符串数据中查找sData2的字符串      *)
(*数据，并将最先查找到的查找位置存储到Result中。                        *)
    M0 := FIND_E( X0, sData1, sData2, Result ) ;
```

备忘录

6 - 78

7 出错列表

以下介绍对创建的ST程序进行转换时发生的出错有关内容。
关于将ST程序写入到CPU模块中时发生的执行出错，请参阅“MELSEC-Q/L编程手册(公共指令篇)”、“QCPU用户手册(硬件设计・维护点检篇)”、“MELSEC-L CPU用户手册(硬件设计・维护点检篇)”。

● 发生了转换出错时
对于程序中的出错将显示出错对话框。
1个程序发生的出错的最多个数为1000个。超出了1000个时，将无法显示在出错列表中。

● 关于转换出错显示
有时会发生1个程序语句中显示多个出错，或1个出错显示多个信息的情况。

● 转换出错列表(出错信息・原因・处理)

No.	出错信息	原因	处理
1	存在无法解析的字符串。(C1009)	存在无法解析的字符串。无法解析的字符串示例如下所示。 例1: 2## 格式错误。 例2: STRING型: 对STRV1进行定义 STRV1 := $"abc"; 使用了$符号。 例3: DO := !10; 使用了!符号。 例4: J25|\K4X0 := 5; 使用了|符号。	变更为正确的字符串。
2	存在无法解析的运算符。(C1010)	存在无法解析的运算符。 例1: Y0 : = M0 => M1;	变更为正确的运算符。
3	实数常数错误。(C1013)	实数常数的记述非法。非法的记述示例如下所示。 例1: REAL型: 对RealV1进行定义 RealV1 := 1.; 实数常数的格式错误。 例2: RealV1 := 0.1E; 实数常数的格式错误。	变更为正确的实数常数记述。
4	软元件的记述错误。(C1014)	软元件的记述非法。 非法的记述示例如下所示。 例1: D0.10 := TRUE; 字软元件的位No. 指定错误。 例2: D0@ := 0;	变更为正确的软元件记述。

No.	出错信息	原因	处理
5	软元件的记述错误。(C1017)	软元件的记述非法。 非法的记述示例如下所示。 例1: DO := %MMW0.10; 用不能使用的格式进行了记述。	变更为正确的软元件记述。
6	注释的记述错误。(C1018)	注释的记述非法。 不是"(*" "*)"的格式。 非法的记述示例如下所示。 例1: (* * 括弧不完整。 例2: (*(* 括弧与*的格式错误。 例3: (* *) "*"与")"之间存在空格。 例4: (*aaaaa) *不足。	变更为正确的注释记述。
7	字符串常数的记述错误。 (C1019)	字符串型常数的记述非法。非法的记述示例如下所示。 例1: STRING型: 对STRV1进行定义 STRV1 := """; 例2: STRV1 := "; "不完整。 例3: STRV1 := "'字符"; 字符串""内存在'。 例4: STRV1 := "$"; 换码顺序的使用方法有错误。	变更为正确的字符串型常数的记述。
8	常数的记述错误。(C1020～C1023)	使用了不支持的数据类型。 或者进行了错误的常数记述。 非法的记述示例如下所示。 例1: W_TMP := TIME#1100_0101; 例2: W_TMP := T#0; 例3: W_TMP := ２#0; "２"为全角。 例4: W_TMP := DT#1900-01-01, 00:00:00; 例5: W_TMP := D#1994-06; 例6: W_TMP := TOD#09:30:61;	使用的数据类型不支持。应变更为正确的数据类型。将常数变更为正确的记述。
9	是未定义的自变量。 (*1)(C1028) (*1表示自变量名。)	使用了未定义的自变量。 使用未定义自变量的示例如下所示。 例1: I_TEST :=1 ; 未进行标签设置的状况下使用了标签。 例2: DO := HAAH; 在16进制数中使用了除A～F以外的字母。 例3: DO := 1234; 表达式中使用了全角空格。	对要使用的自变量进行定义。

No.	出错信息	原因	处理
10	数组的要素指定方法有错误。 (C1033)	数组的要素指定方法错误。 例1: 字型数组标签: W_ARY W_ARY[0,1] := 1; 以与定义的数组不同的格式进行了记述。	变更为正确的数组记述。
11	是未定义的函数。 (*1)(C1049) (*1表示函数名。)	使用了未定义的函数。 使用了未定义函数的示例如下所示。 例1: 实数型标签: RE_1 M0 := OS_E_MD(TRUE,E1.0,RE_1);	变更为正确的函数名记述。
12	自变量名或软元件名过长。 (C1077)	自变量名超出了16个字符。 或者软元件名过长。 出错的程序示例如下所示。 例1: abcde678901234567 := D10; 例2: D0 := D000000··· 00000000000001; 软元件名过长。	应使用已定义的自变量名。 变更为正确的软元件记述。
13	第*1个的自变量中使用了除常数以外的值。(C2021) (*1表示自变量出错位置。)	应指定为常数的自变量中使用了除常数以外的值。 出错的程序示例如下示。 例1: M1 := ROL(M0,X0); 第2个自变量中使用了除常数以外的值。 例2: D100 := SHL(D0,D1); 第2个自变量中使用了除常数以外的值。	应在指定的自变量中使用常数。
14	语法错误。(C2054)	记述了错误的语法。 语法非法的示例如下所示。 例1: D0 : 0; 代入语句中未记述"="。 例2: FOR ARY[0] := 0 TO D10 BY D20 DO D100 := D100+1; END_FOR; 循环自变量中指定了数组要素。 例3: FOR STR.W_TMP := 0 TO D10 BY D20 DO D100 := D100+1; END_FOR; 循环自变量中指定了结构体要素。 例4: D0 := 1++++++++2; +运算符的使用方法有错误。 例5: 字型数组: IntAry1 D0 := IntAry1[[0; 数组的记述方法有错误。 例6: CASE D0 OF 1 :D0 := K5; ELSE D1 := K5; END_CASE; "1"为全角字符。	变更为正确的语法。

No.	出错信息	原因	处理

15	“*1”不完整。(C8006)
	(*1表示以下内容)
	END FOR
	;
	END WHILE
	END FOR
	END_REPEAT
	END_CASE
	END_IF
		声明的最末尾处未记述“；”。	应在声明的最末尾处记述“；”。

		FOR语句中未记述“END FOR”。	应在FOR语句中记述“END FOR”。
		WHILE语句中未记述“END WHILE”。	应在WHILE语句中记述“END WHILE”。
		REPEAT语句中未记述“END_REPEAT”。	应在REPEAT语句中记述“END_REPEAT”。
		在CASE条件语句中未记述“END_CASE”。	CASE条件语句中未记述“END_CASE”。
		IF条件语句中未记述“END_IF”。	在IF条件语句中记述“END_IF”。
16	循环语句的外面使用了EXIT语句。(C8009)	循环语句的外面记述了“EXIT”语句。	在循环语句中记述“EXIT”语句。
17	常数记述错误。(C8010)	使用了不支持的数据类型。 例1: 类型标签: wTime wTime := T#1111111111111111 1s;	使用的数据类型不支持。 应变更为正确的数据类型。
18	调用了未定义的FB。(C8011)	调用了未定义的FB。 使用未定义FB的示例如下所示。 例1: FB_1(); 对未定义的FB进行了调用。 例2: 字型标签: W_TMP W_TMP(); 记述了除FB以外的自变量。	应对使用的FB进行定义。
19	输入/输入输出自变量“*1”中未进行值指定。(C8012) (*1表示输入・输入输出自变量名。)	FB的输入・输入输出自变量中未进行值指定。 上述出错的示例如下所示。 例1: 输入输出自变量: IO_TEST1 引用FB名: FB1 FB1(); 例2: 输入输出自变量: IO_TEST1 引用FB名: FB1 FB1(IO_TEST); 输入自变量未代入值。	在FB的输入・输入输出自变量中进行值的指定。
20	自变量“*1”的类型不一致。(C8013) (*1表示自变量名。)	FB调用的自变量与指定的值或自变量的类型不一致。 上述出错的示例如下所示。 例1: 输入自变量(字型): IN1 引用FB名: FB1 FB1(IN1 := TRUE); 字型输入自变量中指定了位型常数。 例2: 输入自变量(字型): IN1 输出自变量(字型): OUT1 引用FB名: FB1 双字型: DIN1 FB1(IN1 : =DIN1); 字型输入自变量中指定了双字型自变量。	变更为与FB调用的自变量一致的类型。

No.	出错信息	原因	处理
21	输入输出自变量、输出自变量中不能指定无法代入值的自变量。 “*1”(C8014) (*1表示输入输出自变量名、输出自变量名。)	调用的FB的输入输出自变量、输出自变量中指定了无法代入值的自变量。 例1: 输入输出自变量: IO_TEST1 IO_TEST1 : = TRUE; 引用FB名: FB1 FB1(IO_TEST1 := TRUE); 向输入输出自变量传递了常数。 例2: 输入自变量: IN1 输出自变量: OUT1 引用FB名: FB1 字型常数标签: wCon FB1(IN1 : =1,OUT1 := wCon); 向字型输出自变量传递了常数标签。	调用的FB的输入输出自变量、输出自变量中应指定可以进行值的代入的自变量。
22	使用了不能作为FB自变量使用的自变量“*1”。(C8015) (*1表示自变量名。)	向调用的FB的输入・输出・输入输出自变量以外的自变量传递了值。 上述出错的示例如下所示。 例1: 输入自变量IN1 输出自变量OUT1 自变量TEST1 引用FB名: FB1 FB1(TEST1 := X10);	在调用FB时不要使用除FB的输入・输出・输入输出自变量以外的自变量。
23	自变量“*1”的分配重复。 (C8016) (*1表示自变量名。)	在FB调用中同一个自变量使用了2个以上。 例1: 输入输出自变量(位型): INOUT1 引用FB名: FB1 FB1(INOUT1 := TRUE,INOUT1 := FALSE);	FB调用中不要使用同一个自变量。
24	自变量“*1”未定义。(C8017) (*1表示自变量名。)	进行调用的FB的自变量未定义。 例1: 输入输出自变量: INOUT1 引用FB名: FB1 FB1(TMP_INOUT1 := TRUE);	应对进行调用的FB自变量进行定义。
25	整数值“*1”有错误。 (C8018) (*1表示整数值。)	整数值非法。 例1: D1 := 9999999999 ; 整数值超出了允许使用范围。	变更为允许范围内的整数值。
26	常数“*1”有错误。(C8019) (*1表示常数。)	BOOL常数非法。 例1: D1 := 2##0011_0101; 记述了错误的BOOL常数。 例2: M0 :=2 #F; 记述了错误的BOOL常数。	将记述变更为可用的BOOL常数。
27	数组自变量的要素编号中使用了除字型以外的值。 (C8021)	要素指定中使用了除字型以外的值 例1: 位型数组: BoolAry1 实数型标签: RealVal BoolAry1[RealVal] := x0; 例2: 位型数组: BoolAry1 BoolAry1[D0<D1] 记述了错误的要素指定。	将要素的数据类型变更为字型。

No.	出错信息	原因	处理
28	数组自变量的要素编号超出允许的定义要素数。(C8022)	指定的要素编号超出了数组定义的要素范围。 例1: 字型数组标签(要素数2): Kosu Kosu[5] := D0;	将要素编号变更为在数组定义的要素范围内。
29	对于不是数组自变量的自变量，以数组格式进行了记述。(C8023)	对于不是数组自变量的自变量以数组格式语句进行了记述。 例1: 字型标签: W_TMP1 W_TMP1[2] := 100; 对不是数组的自变量以数组格式进行了记述。 例2: aaa[1] := D0; 将未定义标签以数组格式进行了记述。	变更为正确的自变量记述。
30	“*1”不是“*2”的要素。(C8024) (*1表示结构体要素名或FB自变量名，*2表示结构体名或FB名。)	结构体的要素名有错误。或FB的自变量名有错误。 例1: 结构体要素名: mem1 引用结构体名: InsSDT1 InsSDT1.mem2 := 100; 记述了错误的结构体要素名。 例2: 输入自变量: IN1 引用FB名: FB1 FB1(IN1 := 10); d0 := FB1.aaa; 记述了未定义的FB输出自变量。	变更为正确的结构体要素名。或变更为正确的FB自变量名。
31	将不能作为FB输出使用的自变量“*1”用于FB“*2”中。(C8025) (*1表示FB自变量名，*2表示FB名。)	使用了不能作为FB输出使用的FB自变量。 例1: 内部自变量(字型): TEMP1 引用FB名: FB1 D100 := FB1.TEMP1; 将内部自变量作为FB输出使用。	使用正确的FB自变量，作为FB输出进行记述。
32	自变量“*1”(FB:*2)不能用作自变量以外。(C8026) (*1表示FB自变量名，*2表示FB名。)	FB自变量的使用方法有错误。 例1: [FB定义] 输入自变量: IN1 输出自变量: OUT1 引用FB名: FB1 X1 := FB1.IN1; 将输入自变量作为FB输出使用。	FB自变量中应使用正确的FB自变量。
33	是未定义的结构体。(C8027)	结构体名非法。 例1: 结构体: 无 字型标签: W_TMP2 W_TMP2.mem1 := 100; 进行了错误的结构体记述。	变更为正确的结构体名。

No.	出错信息	原因	处理
34	在第*1个位置处不能指定无法进行值的代入的自变量。“＊2”(C8028) (*1表示出错位置，*2表示函数名、″:=″。)	在进行值的代入的位置处，指定了无法进行常数及输入自变量等的值的代入的自变量。 例1: 标签(常数型): cnt cnt := D10; 代入到标签常数中。 例2: ABS_E(TRUE,d0,K10); 在函数的输出自变量中记述了常数。 例3: FB输入自变量(字型): IN1 BPLUS_3_M(M0,K1,D0,FB1.IN1); 在进行值的输出的自变量中指定了输入自变量。	变更为可进行值的代入的自变量。
35	第“*1”个的自变量类型不一致。“＊2”(C8029) (*1表示自变量出错位置，*2表示函数名。)	自变量的类型不一致。 例1: 字型数组: IntAry1[0..1] M1 := BACOS_MD(TRUE, IntAry1,D1);	对函数自变量指定的出错位置的类型进行修改。或者对自变量的类型进行修改。
36	“*1”中使用了非法的类型。(C8030) (*1表示“:=”及“*”等的运算符。)	自变量・软元件的左边与右边的数据类型不相同。 例1: D0 : =TRUE; 在字软元件中代入了位型。 例2: D1 := D2*M1; 对字型与位型进行运算。 例3: M0 := d1 > M1; 将字型与位型进行比较。	将自变量・软元件的左边与右边的数据类型置为相同。
37	函数“*1”的*2个变量与定义不一致。(C8031) (*1表示函数名，*2表示与定义不一致的自变量个数。)	函数调用的自变量个数与定义不一致。 例1: ABS(); 记述少于定义的自变量个数。 例2: d0 := ABS(10,10); 记述多于定义的自变量个数。	变更为正确的函数自变量个数。

38	格式类型非法。(C8032)	控制语句中格式的类型不一致。
		例1: 双字型: DwLBL
		FOR DwLBL := W1 TO W2 BY W3 DO
		W5 := W6;
		END_FOR;
		循环自变量与最终值的表达式・增加表达式的数据类型不一致。
		例2: CASE W1 OF
		1: D0 := 1;
		2147483648 : D0 := 2;
		ELSE
		D0 := 10;
		END_CASE;
		整数表达式与选择值的数据类型不一致。
		例3: IF W1 THEN
		D100 := 1;
		END_IF;
		布尔表达式中指定了字型。
			变更为合适的格式类型。

No.	出错信息	原因	处理
39	常数自变量(FOR语句内)中不能进行代入。 (C8033)	试图对常数自变量进行写入。 上述出错程序示例如下所示。 例1: 常数标签: tei FOR tei := W10 TO W20 BY W30 DO R10 := R20; END_FOR;	不能对常数自变量(FOR语句内)进行写入。
40	FOR语句中使用了除字/双字型以外的自变量。(C8034)	FOR语句中使用了除字・双字以外类型的自变量。 (循环自变量中指定了字符串・数组・结构体自变量名等情况下) 例1: 字符串标签: Str1, Str2, Str3, Str4 FOR Str1 := Str2 TO Str3 BY Str4 DO D0 := D100; END_FOR; 对循环自变量指定了字符串自变量名。	在FOR语句中使用正确的类型。
41	语句中“*1”不完整。 (C8039) (*1表示 DO UNTIL OF THEN DO。)	FOR语句未记述“DO”。 例1: FOR D1 := D2 TO D3 BY D4	FOR语句中应记述“DO”。
		REPEAT语句中未记述“UNTIL”。	REPEAT语句中应记述“UNTIL”。
		CASE条件语句中未记述“OF”。	CASE条件语句应记述“OF”。
		IF条件语句未记述“THEN”。	IF条件语句应记述“THEN”。
		ELSIF条件语句未记述“THEN”。	ELSIF条件语句应记述“THEN”。
		WHILE语句中未记述“DO”。	WHILE语句中应记述“DO”。
42	调用的FB“*1”的自变量有错误。(C8040) (*1表示FB名。)	调用的FB的输入・输出・输入输出自变量的记述非法。 例1: 引用FB名: FB1 FB1(X10); 例2: 输入自变量: IN1 引用FB名: FB1 FB1(FB1.IN1);	变更未正确的FB的调用记述。
43	未指定存储函数返回值的自变量。(C8041)	在没有EN/ENO的函数中，不存在返回值・存储返回值的自变量。 未指定存储返回值的自变量的示例如下所示。 例1: INT_TO_DINT(D0);	对函数的返回值进行记述。

No.	出错信息	原因	处理

44	控制语句的嵌套及条件过多或控制语句的程序过长。(C9017)	控制语句的嵌套及条件过多或控制语句的程序过长。
		例1: IF D0 = 0 THEN
		IF D1 = 0 THEN
		・・・
		END_IF;
		END_IF;
		语句内设置了598次以上的嵌套。
		例2: FOR D0 := 0 TO 100 BY 1 DO
		FOR D1 := 0 TO 100 BY 1 DO
		・・・
		END_FOR;
		END_FOR;
		FOR语句内设置了299次以上的嵌套。
		例3: WHILE D0 < 10 DO
		WHILE D1 < 10 DO
		・・・
		END_WHILE;
		END_WHILE;
		WHILE语句内设置了598次以上的嵌套。
		例4: CASE W0 OF
		0: D0 := 0;
		1: D0 := 1;
		・・・
		1491: D0 := 1491;
		END_CASE;
		CASE语句中使用了1492个以上的整数选择值。
			控制语句的程序过长。应减少嵌套的个数，减少条件等，使控制语句的程序变短。

45	函数“*1”的执行条件EN的值不正常。(C9019) (*1表示函数名。)	特定的函数中，执行条件EN中未放入常时TRUE，却放入了FALSE 例1: EI_M(FALSE); 例2: DI_M(0); 例3: COM_M (FALSE);	在执行条件EN中指定正确的值。
46	系统文件的读取失败。(C9020)	系统文件已损坏。	应重新进行系统文件安装。
47	Z0、Z1为系统所使用，因此不能使用。(C9035)	使用了Z0、Z1。 例1: INC_M(M10, D0Z1); 例2: Z0 := 10;	变更为不使用Z0、Z1。
48	常数*1超出了要素编号(*2..*3)范围。(C9039) (*1表示要素编号，*2、*3表示要素个数。)	数组的要素编号非法(超出范围)。 例1: 字型数组标签: IntAry1[255] IntAry1[K255] := 0;	应在要素个数的范围内进行常数指定。
49	用0进行除法运算。(C9065)	用0进行除法运算。 例1: D0 := 10/0; 例2: D1 := W1/K0;	对用0进行除法运算之处进行变更。

No.	出错信息	原因	处理
50	函数“*1”的返回值不能直接参照。(C9066) (*1表示函数名。)	运算时无法对字符串函数(***_STR()、LEFT()、RIGHT()函数等)的返回值进行直接参照的情况下。 例1: M0 := INT_TO_STR(D0) < "AAA";	将出错的字符串函数设置为另外的程序，将程序修改为使用该字符串函数的返回值。
51	系统文件的读取失败。 (C9072)	系统文件已损坏。	系统文件已损坏。应重新安装。
52	函数“*1”转换时发生了出错。(C9076) (*1表示函数名。)	转换结果有错误。 例1: TIMER_H_M(X0,TC0,-1); 第3自变量中使用了负值。	函数的自变量中应使用允许指定的数据类型或允许指定的范围的数据。
53	输入自变量中使用了表达式。 (C9118)	指定MELSEC函数的起始软元件的输入自变量中指定了运算公式或函数。 在指定MELSEC函数的输入自变量中指定了要素编号可变的位型数组要素。 指定MELSEC函数的输入自变量中指定了除要素编号可变的位型以外的数组要素。 例1: BMOV_M(X0,MAX(D0,D1,D2), D100,D200); 输入自变量中使用了函数。 例2: TO_M(X0,D0+1,D1,D2,D3); 输入自变量中使用了运算公式。 例3: DTO_M(X0,Dint1+K8X0,D1,D2,D3); 输入自变量中使用了运算公式。 例4: BKRST_M(X0,ARY[D0],D1); 指定起始软元件的输入自变量S1中指定了要素编号可变的位型数组要素。 例5: BKPLUS_M(M0,ARY[D1], ARY[D2], ARY[D3], ARY[D4]); 指定起始软元件的输入自变量S1、S2中进行了要素编号可变的数组要素指定。	•指定了运算公式或函数的情况下 在指定起始软元件的输入自变量中不能指定运算公式或函数。应指定标签名或软元件。 •指定了要素编号可变的位型数组要素的情况下 指定软元件的起始的自变量中不能指定要素编号可变的位型数组要素。 将要素编号修改为常数，或者指定标签名或位软元件。 •指定了要素编号可变的位型以外的数组要素的情况下，如果进行了要素编号可变的数组要素指定，则仅限于在编译器内部使用的变址寄存器中使用，因此应将要素编号修改为常数，或者进行指定标签名或位软元件，减少1个函数内使用的数组要素指定数等的修改。
54	转换结果有错误。(F0028) “*1” (*1表示不正确的转换结果。)	ST的语法正确，但软元件的规格等导致出错。 例1: TS0 := TRUE;	对出错信息中显示的列表的内容进行确认，对程序进行修改。

No.	出错信息	原因	处理
55	字符串的允许使用最大字符数为32个字符。(F0102)	使用的字符数超出了允许设置的最大值。 使用了超出最大值以上的字符数的出错示例如下所示。 例1: 字符串标签: Str1 Str1 := "123456789012345 678901234567890 123"; 字符串数为33个字符的情况	将字符串变更为32个字符以内。
56	使用了不正确的软元件或超出范围的数值。(F0137)“*1”(*1表示不正确的转换结果。)	使用了不正确的软元件或超出范围的数值。 例1: D0 := K-32769; 例2: D0 := H10001; 例3: M0 := COUNTER_M(TRUE,CC2,-1);	变更为正确的软元件或范围内的数值。
57	TIMER_M的自变量中使用了除定时器以外的软元件。(F0177)	TIMER_M的自变量中使用了定时器以外的软元件。 例1:TIMER_M(X0, CC0, 2);	函数TIMER_M的自变量中应使用定时器软元件。
58	COUNTER_M的自变量中使用了除计数器以外的软元件。(F0178)	COUNTER_M的自变量中使用了计数器以外的软元件。 例1: COUNTER_M(X0, TC0, 2);	函数COUNTER_M的自变量中应使用计数器软元件。
59	CONCAT(_E)函数的自变量与返回值使用了相同的自变量。(F0196)	CONCAT(_E)函数的自变量与返回值使用了相同的自变量。 例1: 字符串标签: Str1・Str2 Str1 := CONCAT(Str2,Str1);	CONCAT(_E)函数的自变量与返回值应使用不同的自变量。
60	INSERT(_E)函数的自变量与返回值使用了相同的自变量。(F0206)	INSERT(_E)函数的自变量与返回值使用了相同的自变量。 例1: 字符串标签: Str1 Str1 := INSERT(Str1,Str2, D0); 自变量与返回值使用了相同的自变量。	INSERT(_E)函数的自变量与返回值应使用不同的自变量。
61	使用了不正确的软元件类别。(C10000)	使用了不正确的软元件类别(定时器・累计定时器・计数器・指针)。 例1: Timer1 := 0; 使用了软元件类别定时器。	不能使用不正确的软元件类别(定时器・累计定时器・计数器・指针)。 应变更为可以使用的软元件类别。
62	指定的软元件、数值超出了范围，或不能使用。(C10001)	软元件编号超出了允许范围。 或指定了不能使用的软元件。或数值超出了允许范围。 例1: M0 := X2000; X的软元件编号中指定了超出1FFF的软元件编号。 例2: D0 := A0; QCPU/LCPU中使用了累加器。 例3: 双字型标签: DW1 DW1 := K2147483648;	将软元件编号修改为在允许范围内。或者变更为可以使用的软元件。或者将数值修改为在允许的范围内。

No.	出错信息	原因	处理
63	函数或运算符的个数过多。(C10002)	1个语句内函数或运算符合计使用了1025个以上。 例1: D0 := 1+1+1+1+・・・+1+1; 1个语句内使用了1025个以上的运算符"+"。 例2: d0 := ABS(ABS(ABS(ABS(wlabel) ・・・・)); 1个语句内使用了1025个以上的函数ABS。	1个语句内使用函数或运算符的情况下 ，合计应不超过1025个。
64	数组要素指定的嵌套过多。(C10003)	数组要素指定中设置了17个以上的嵌套。 例1: Array1[Array1[Array1[Array1 [Array1[Array1[.....]]]]]; 数组要素指定中设置了17个以上的嵌套。	将数组要素指定的嵌套变更为5个。不支持6个以上。17个以上时将变为出错状态。
65	指定的FB名已被使用。(C10004)	程序中对同一个FB名进行了2次以上的FB调用。 例1: 引用FB名: FB1 输入输出自变量名: INOUT1 FB1(INOUT1 := D100); FB1(INOUT1 := D101);	将同一个FB名的FB调用设置为1次。
66	输出自变量在FB调用前已被使用。(C10005)	在FB调用之前使用了FB输出。 例1: 引用FB名: FB1 输入输出自变量名: INOUT1 输出自变量名: OUT1 D0 := FB1.OUT1; FB1(INOUT1 := D100);	变更为在FB调用后使用FB输出。
67	“*1”中使用了非法的类型。(C10006) (*1表示运算符。)	双字、实数型的代入语句及运算中使用了超出数据类型范围的值。 例1: 双字型标签: w_Dword w_Dword := -2147483649;	应指定正确的范围。
68	函数“*1”中使用了非法的类型。(C10007) (*1表示函数名。)	对MELSEC函数的自变量使用了非法的数据类型。 例1: RST_M(M0,ddev1); 函数RST_M的第二自变量中指定了双字型。 例2: DECO_M(M0,Real1,K8,Real2); 函数DECO_M的第二•四自变量中指定了实数型。 例3: COMRD_S_MD(M0,ddev1, Str32); 函数COMRD_S_MD的第二自变量中指定了双字型。	自变量中应使用正确的数据类型的自变量。

附录

附录1 标签・FB名中不能使用的字符串

ST编程时，不能作为标签・FB名使用的字符串如下所示。
软元件名、指令名、函数名中使用的字符串不能作为标签・FB名使用。
如果使用了下表中所示的字符串，在执行登录／编译时将变为出错状态。

	标签・FB名中不能使用的字符串
A	A, ACALL, ACJ, ACTION, ANB, ANY, ANY_BIT, ANY_DATE, ANY_DERIVED, ANY_ELEMENTARY, ANY_INT, ANY_MAGNITUDE, ANY_NUM, ANY_REAL, ANY_SIMPLE, ANY_STRING, ARRAY, AT
B	B, BEND, BL, BLOCK, BOOL, BOOL_TO_BYTE(DINT, DWORD, INT, REAL, SINT, UDINT, UINT, USINT, WORD), BY, BYTE, BYTE_TO_BOOL(DINT, DWORD, INT, REAL, SINT, STRING, UDINT, UINT, USINT, WORD), BWORKR, BWORKRP, BWORKW, BWORKWP, B_BCD_TO_DINT(INT, SINT)
C	C, CASE, CAL, CALC, CALCN, CONFIGURATION, CONSTANT, CTD, CTU, CTUD
D	D, DATE, DATE_AND_TIME, DINT, DINT_TO_BCD(BOOL, BYTE, DWORD, INT, REAL, SINT, STRING, TIME, UDINT, UINT, USINT, WORD), DO, DT, DWORD, DWORD_TO_BOOL(BYTE, DINT, INT, REAL, SINT, STRING, UDINT, UINT, USINT, WORD), DX, DY, D_BCD_TO_DINT(INT, SINT)
E	E, ELSE, ELSIF, EN, END, END_ACTION, END_CASE, END_FOR, END_FUNCTION, END_PROGRAM, END_IF, END_REPEAT, END_RESOURCE, END_STEP, END_STRUCT, END_TRANSITION, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EQ_STRING, EXIT
F	F, FALSE, FD, FOR, FROM, FUNCTION, FUNCTION_BLOCK, FX, FY, F_EDGE, F_TRIG
G	G, GE, GE_STRING, GT, GT_STRING
H	H
I	I, IF, INITIAL_STEP, INT, INT_TO_BOOL(BYTE, DINT, DWORD, REAL, SINT, STRING, UDINT, UINT, USINT, WORD)
J	J, JMPC, JMPCN
K	K
L	L, LDN, LE, LE_STRING, LIMIT_STRING, LINT, LREAL, LT, LT_STRING, LWORD
M	M, MAX_STRING, MIN_STRING, MOD, MPP, MPS, MRD
N	N, NE, NE_STRING, NOP, NOT
O	OF, ON, ORB, ORN
P	P, PROGRAM
Q	Q
R	R, R1, RCALL, RCJ, READ, READ_ONLY, READ_WRITE, REAL, REAL_TO_BOOL(BYTE, DINT, DWORD, INT, SINT, STRING, UDINT, UINT, USINT, WORD), RECV, REPEAT, RESOURCE, RETAIN, RETC, RETCN, RETURN, REQ, RS, R_EDGE, R_TRIG
S	S, SB, SD, SEND, SEL_STRING, SFCP, SFCPEND, SG, SINT, SINT_TO_BOOL(BYTE, DINT, DWORD, INT, REAL, STRING, UDINT, UINT, USINT, WORD), SM, SR, SREAD, ST, STEP, STEPC, STEPD, STEPG, STEPI, STEPID, STEPIR, STEPISC, STEPISE, STEPIST, STEPN, STEPR, STEPSC, STEPSE, STEPST, STN, STRING, STRING_TO_BYTE(DINT, DWORD, INT, REAL, SINT, TIME, UDINT, UINT, USINT, WORD), STRUCT, SW, SWRITE, SZ
T	T, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_STRING, TO, TOD, TOF, TON, TP, TR, TRAN, TRANA, TRANC, TRANCA, TRANCO, TRANCOC, TRANCOCJ, TRUNC_DINT(INT, SINT), TRANJ, TRANL, TRANO, TRANOA, TRANOC, TRANOCA, TRANOCJ, TRANOJ, TRANSITION, TRUE, TYPE

附

	标签・FB名中不能使用的字符串
U	U,UDINT,UDINT_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UINT,UNTIL,USINT,WORD), UINT,UINT_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UDINT,USINT,WORD), ULINT,UNTIL,USINT,USINT_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UDINT,UINT,WORD)
V	V,VAR,VAR_CONSTANT,VAR_EXT,VAR_EXTERNAL,VAR_EXTERNAL_FB,VAR_EXTERNAL_PG, VAR_GLOBAL,VAR_GLOBAL_FB,VAR_GLOBAL_PG,VAR_IN_OUT,VAR_INPUT,VAR_OUTPUT,VAR_TEMP,VD,VOID
W	W,WHILE,WITH,WORD,WORD_TO_BOOL(BYTE,DINT,DWORD,INT,REAL,SINT,STRING,UDINT,UINT,USINT), WORKR,WORKRP,WORKW,WORKWP,WRITE,WSTRING,W_BCD_TO_DINT(INT,SINT)
X	X,XOR,XORN
Y	Y
Z	Z,ZNRF,ZR

标签定义时的注意事项

1. 不能使用空格。
2. 起始字符不能使用数字。
3. 不能使用下述字符。

   (，)，＊，/，＋，－，<，>，＝，&，

   !，”，#，$，%，’，~，^，|，@，`，[，]，{，}，;，:，，，.，?，\，_

   下划线仅在位于字符串的最后时，或者连续使用了2个以上时将会变为出错状态。
4. 不能使用软元件名。

   软元件名的后面附加了0～F的字符的情况下将变为出错状态。

   例)XFFF,M100
5. 标签名中不要使用“EnDm”(例: E001D9)。

   (n、m表示任意的数值。)

   有可能被识别为实数值而无法作为标签名使用。
6. 不能使用指令名(顺控程序指令、基本指令、应用指令、SFC指令等)、函数名(MELSEC函数、IEC函数)。

附

附 - 2

附录2 GX Developer与GX Works2中ST指令对应表

ST程序中可以使用的指令根据GX Developer与GX Works2而有所不同。
因此将包含有GX Works2中创建的ST程序的工程以GX Developer格式保存并通过GX Developer进行了读取时，如果直接进行编译则有可能变为出错状态。在这种情况下应根据下表对ST程序进行修改。

GX Works2	GX Developer
BACOS	BACOS_MD
BAND	BAND_MD
BASIN	BASIN_MD
BATAN	BATAN_MD
BCD	BCD_M
BCOS	BCOS_MD
BDSQR	BDSQR_MD
BIN	BIN_M
BKAND	BKAND_M
BKBCD	BKBCD_M
BKBIN	BKBIN_M
BKOR	BKOR_M
BKRST	BKRST_M
BKXNR	BKXNR_M
BKXOR	BKXOR_M
BMOV	BMOV_M
BRST	BRST_M
BSET	BSET_M
BSFL	BSFL_M
BSFR	BSFR_M
BSIN	BSIN_MD
BSQR	BSQR_MD
BTAN	BTAN_MD
BTOW	BTOW_MD
BXCH	BXCH_M
CML	CML_M
COM	COM_M
DATERD	DATERD_MD
DATEWR	DATEWR_MD
DBAND	DBAND_MD
DBCD	DBCD_M
DBIN	DBIN_M
DBL	DBL_M
DCML	DCML_M
DDEC	DDEC_M

GX Works2	GX Developer
DEC	DEC_M
DECO	DECO_M
DELTA	DELTA_M
DFLT	DFLT_M
DFRO	DFRO_M
DGRY	DGRY_M
DI	DI_M
DINC	DINC_M
DIS	DIS_M
DLIMIT	DLIMIT_MD
DMAX	DMAX_M
DMIN	DMIN_M
DNEG	DNEG_M
DOR	DOR_M
DRCL	DRCL_M
DRCR	DRCR_M
DROL	DROL_M
DROR	DROR_M
DSER	DSER_M
DSFL	DSFL_M
DSFR	DSFR_M
DSORT	DSORT_M
DSUM	DSUM_M
DTEST	DTEST_MD
DTO	DTO_M
DWSUM	DWSUM_M
DXCH	DXCH_M
DXNR	DXNR_M
DXOR	DXOR_M
DZONE	DZONE_MD
EI	EI_M
EMOD	EMOD_M
ENCO	ENCO_M
ENEG	ENEG_M
EREXP	EREXP_M

GX Works2	GX Developer
ESTR	ESTR_M
EVAL	EVAL_M
FLT	FLT_M
FMOV	FMOV_M
FROM	FROM_M
GBIN	GBIN_M
GRY	GRY_M
HOUR	HOUR_M
INC	INC_M
MIDR	MIDR_M
NDIS	NDIS_M
NEG	NEG_M
NUNI	NUNI_M
OUT	OUT_M
PLOW	PLOW_M
POFF	POFF_M
PSCAN	PSCAN_M
PSTOP	PSTOP_M
QCDSET	QCDSET_M
QDRSET	QDRSET_M
RCL	RCL_M
RCR	RCR_M
RFS	RFS_M
RND	RND_M
RSET	RSET_MD
RST	RST_M
SECOND	SECOND_M
SEG	SEG_M
SER	SER_M
SET	SET_M
SFL	SFL_M
SFR	SFR_M
SFT	SFT_M
SORT	SORT_M
SRND	SRND_M

(转下页)

附 - 3

GX Works2	GX Developer
STOP	STOP_M
SUM	SUM_M
SWAP	SWAP_MD
TEST	TEST_MD
UNI	UNI_M

GX Works2	GX Developer
WAND	WAND_M
WDT	WDT_M
WOR	WOR_M
WSUM	WSUM_M
WTOB	WTOB_MD

GX Works2	GX Developer
WXNR	WXNR_M
WXOR	WXOR_M
XCH	XCH_M
ZONE	ZONE_MD

附 - 4

索引

[数字]

10进制ASCII→32位BIN转换(DDABIN_S_MD) ·····5-76
10进制ASCII→BCD4位转换(DABCD_S_MD) ·····5-78
10进制ASCII→BCD8位转换(DDABCD_S_MD) ·····5-78
10进制ASCII→BIN转换(DABIN_S_MD) ·····5-76
16进制ASCII→32位BIN转换(DHABIN_S_MD) ·····5-77
16进制ASCII→BIN转换(HABIN_S_MD) ·····5-77
16位BIN→32位BIN转换(DBL_M) ·····5-27
16位BIN的2的补数(NEG_M) ·····5-30
16位数据的4位分离(DIS_M) ·····5-62
16位数据的4位合并(UNI_M) ·····5-63
16位数据否定传送(CML_M) ····· 5-33
16位数据替换(XCH_M) ····· 5-35
1字右移(DSFR_M) ·····5-55
1字左移(DSFL_M) ·····5-55
32位BCD→BIN转换(DBIN_M) ·····5-24
32位BIN→10进制ASCII转换(DBINDA_S_MD) ·····5-73
32位BIN→16进制ASCII转换(DBINHA_S_MD) ·····5-74
32位BIN→16位BIN转换(WORD_M) ···· 5-27
32位BIN→BCD转换(DBCD_M) ····· 5-23
32位BIN→浮动小数点转换(DFLT_M) ·· 5-26
32位BIN→格雷码转换(DGRY_M) ···· 5-28
32位BIN→字符串转换(DSTR_S_MD) ·····5-80
32位BIN的2的补数(DNEG_M) ·····5-30
32位BIN递减(DDEC_M) ·····5-22
32位BIN递增(DINC_M) ·····5-22
32位浮动小数点→BIN转换(DINT_E_MD) ····· 5-25
32位格雷码→BIN转换(DGBIN_M) ···· 5-29
32位数据查找(DSER_M) ····· 5-59
32位数据的位测试(DTEST_MD) ····· 5-57
32位数据否定传送(DCML_M) ····· 5-33
32位数据否定排他逻辑和(2软元件)(DXNR_M) ····· 5-47
32位数据否定排他逻辑和(3软元件)(DXNR_3_M) ····· 5-48
32位数据合计值计算(DWSUM_M) ···· 5-69
32位数据逻辑和(2软元件)(DOR_M) ··· 5-42
32位数据逻辑和(3软元件)(DOR_3_M) ·· 5-43
32位数据逻辑积(2软元件)(DAND_M) ·· 5-40
32位数据逻辑积(3软元件)(DAND_3_M) · 5-40
32位数据排他逻辑和(2软元件)(DXOR_M) ····· 5-45
32位数据排他逻辑和(3软元件)(DXOR_3_M) ····· 5-45
32位数据排序(DSORT_M) ·····5-68
32位数据上下限极限控制(DLIMIT_MD) ····5-99
32位数据死区控制(DBAND_MD) ·····5-100
32位数据替换(DXCH_M) ·····5-35
32位数据位校验(DSUM_M) ·····5-60
32位数据位域控制(DZONE_MD) ·····5-101
32位数据右旋转(包含进位标志)(DRCR_M) ·····5-51
32位数据右旋转(不包含进位标志)(DROR_M) ····· 5-51
32位数据最大值查找(DMAX_M) ····· 5-66
32位数据最小值查找(DMIN_M) ····· 5-67
32位数据左旋转(包含进位标志)(DRCL_M) ····· 5-52
32位数据左旋转(不包含进位标志)(DROL_M) ····· 5-52
7段编译(SEG_M) ····· 5-62

[A]

ABS(_E)(绝对值) ·····6-21
ACOS(_E)(浮动小数点COS-1运算) ·····6-29
ACOS_E_MD(浮动小数点COS-1运算) ·····5-90
ADD_E(加法) ·····6-31
AND_E(逻辑和) ·····6-43
ANY ·····3-4
ARRAY ·····3-3
ASC_S_MD(BIN→ASCII转换) ·····5-83
ASCII→BIN转换(HEX_S_MD) ·····5-83
ASIN(_E)(浮动小数点SIN-1运算) ·····6-28
ASIN_E_MD(浮动小数点SIN-1运算) ·····5-89
ATAN(_E)(浮动小数点TAN-1运算) ·····6-30
ATAN_E_MD(浮动小数点TAN-1运算) ·····5-90

[B]

BACOS_MD(BCD型COS-1运算) ·····5-97
BAND_MD(死区控制) ·····5-100
BASIN_MD(BCD型SIN-1运算) ·····5-97
BATAN_MD(BCD型TAN-1运算) ·····5-98

索引 - 1

BCD_M(BIN→BCD转换) ·····5-23
BCD→BIN转换(BIN_M) ·····5-24
BCD4位→10进制ASCII转换
(BCDDA_S_MD) ·····5-75
BCD4位的乘法(BMULTI_M) ·····5-17
BCD4位的除法(BDIVID_M) ·····5-17
BCD4位的加法(2软元件)(BPLUS_M) ·····5-13
BCD4位的加法(3软元件)(BPLUS_3_M) ·····5-13
BCD4位的减法(2软元件)(BMINUS_M) ·····5-14
BCD4位的减法(3软元件)(BMINUS_3_M) ····5-14
BCD4位平方根(BSQR_MD) ·····5-94
BCD8位→10进制ASCII转换
(DBCDDA_S_MD) ·····5-75
BCD8位的乘法(DBMULTI_M) ·····5-18
BCD8位的除法(DBDIVID_M) ·····5-18
BCD8位的加法(2软元件)(DBPLUS_M) ·····5-15
BCD8位的加法(3软元件)(DBPLUS_3_M) ····5-15
BCD8位的减法(2软元件)(DBMINUS_M) ·····5-16
BCD8位的减法(3软元件)(DBMINUS_3_M) ···5-16
BCD8位平方根(BDSQR_MD) ·····5-95
BCDDA_S_MD(BCD4位→10进制ASCII转换) ··5-75
BCD格式数据→浮动小数点(EREXP_M) ·····5-87
BCD型COS-1运算(BACOS_MD) ·····5-97
BCD型COS运算(BCOS_MD) ·····5-96
BCD型SIN-1运算(BASIN_MD) ·····5-97
BCD型SIN运算(BSIN_MD) ·····5-95
BCD型TAN-1运算(BATAN_MD) ·····5-98
BCD型TAN运算(BTAN_MD) ·····5-96
BCOS_MD(BCD型COS运算) ·····5-96
BDIVID_M(BCD4位的除法) ·····5-17
BDSQR_MD(BCD8位平方根) ·····5-95
BIN_M(BCD→BIN转换) ·····5-24
BIN→10进制ASCII转换(BINDA_S_MD) ·····5-73
BIN→16进制ASCII转换(BINHA_S_MD) ·····5-74
BIN→ASCII转换(ASC_S_MD) ·····5-83
BIN→BCD转换(BCD_M) ·····5-23
BIN→浮动小数点转换(FLT_M) ·····5-26
BIN→格雷码转换(GRY_M) ·····5-28
BIN→字符串转换(STR_S_MD) ·····5-80
BINDA_S_MD(BIN→10进制ASCII转换) ·····5-73
BINHA_S_MD(BIN→16进制ASCII转换) ·····5-74
BIN块加法(BKPLUS_M) ····· 5-20
BIN块减法(BKMINUS_M) ····· 5-20
BKAND_M(块数据逻辑积) ····· 5-41
BKBCD_M(块转换BIN→BCD转换) ···· 5-31
BKBIN_M(块转换BCD→BIN转换) ···· 5-32
BKCMP_EQ_M(块数据比较(=)) ····· 5-10
BKCMP_GE_M(块数据比较(>=)) ····· 5-12
BKCMP_GT_M(块数据比较(>)) ····· 5-11
BKCMP_LE_M(块数据比较(<=) ····· 5-11
BKCMP_LT_M(块数据比较(<) ····· 5-12
BKCMP_NE_M(块数据比较(<>) ····· 5-10
BKMINUS_M(BIN块减法) ····· 5-20
BKOR_M(块数据逻辑和) ····· 5-43
BKPLUS_M(BIN块加法) ····· 5-20
BKRST_M(位软元件批量复位) ····· 5-58
BKXNR_M(块数据否定排他逻辑和) ··· 5-48
BKXOR_M(块数据排他逻辑和) ····· 5-46
BMINUS_3_M(BCD4位的减法(3软元件)) ····5-14
BMINUS_M(BCD4位的减法(2软元件)) ·····5-14
BMOV_M(块传送) ·····5-34
BMULTI_M(BCD4位的乘法) ·····5-17
BOOL ·····3-3
BOOL_TO_DINT(_E)(布尔型(BOOL)→
双精度整数型(DINT)转换) ·····6-3
BOOL_TO_INT(_E)(布尔型(BOOL)→
整数型(INT)转换) ·····6-4
BOOL_TO_STR(_E)(布尔型(BOOL)→
字符串型(STRING)转换) ·····6-5
BPLUS_3_M(BCD4位的加法(3软元件)) ·····5-13
BPLUS_M(BCD4位的加法(2软元件)) ·····5-13
BRST_M(字软元件的位复位) ·····5-56
BSET_M(字软元件的位设置) ·····5-56
BSFL_M(n位数据1位左移) ·····5-54
BSFR_M(n位数据1位右移) ·····5-54
BSIN_MD(BCD型SIN运算) ·····5-95
BSQR_MD(BCD4位平方根) ·····5-94
BTAN_MD(BCD型TAN运算) ·····5-96
BTOW_MD(字节单位数据合并) ····· 5-65
BXCH_M(块数据替换) ····· 5-36
编码(ENCO_M) ····· 5-61
编译(DECO_M) ····· 5-61
变址修饰 ····· 3-16
标签 ····· 3-11
不等号(<>)(NE_E) ····· 6-67
布尔型(BOOL)→双精度整数型(DINT)
转换(BOOL_TO_DINT(_E)) ·····6-3
布尔型(BOOL)→整数型(INT)
转换(BOOL_TO_INT(_E)) ·····6-4
布尔型(BOOL)→字符串型(STRING)
转换(BOOL_TO_STR(_E)) ·····6-5

[C]
CASE条件语句 ·····4-12

索引 - 2

CML_M(16位数据否定传送) ·····5-33
COM_M(刷新) ·····5-70
COMRD_S_MD(软元件的注释数据读取) ·····5-79
CONCAT(_E)(字符串的连接) ·····6-73
COS(_E)(浮动小数点COS运算) ·····6-26
COS_E_MD(浮动小数点COS运算) ·····5-88
COUNTER_M(计数器) ·····5-5
乘法(MUL_E) ·····6-32
程序待机(PSTOP_M) ·····5-107
程序低速执行登录(PLOW_M) ·····5-108
程序扫描执行登录(PSCAN_M) ·····5-108
程序输出OFF待机(POFF_M) ·····5-107
除法(DIV_E) ·····6-34
从字符串的开始位置截取(LEFT(_E)) ·····6-70
从字符串的指定位置查找(FIND(_E)) ·····6-77
从字符串的指定位置截取(MID(_E)) ·····6-72
从字符串的指定位置删除(DELETE(_E)) ···6-75
从字符串的指定位置替换(REPLACE(_E)) ··6-76
从字符串的终端截取(RIGHT(_E)) ·····6-71
从字符串右侧截取(RIGHT_M) ·····5-84
从字符串左侧截取(LEFT_M) ·····5-84

[D]
DABCD_S_MD(10进制ASCII→BCD4位转换) ··5-78
DABIN_S_MD(10进制ASCII→BIN转换) ·· 5-76
DAND_3_M(32位数据逻辑积(3软元件)) · 5-40
DAND_M(32位数据逻辑积(2软元件)) ·· 5-40
DATEMINUS_M(时钟数据的减法) ·····5-105
DATEPLUS_M(时钟数据的加法) ·····5-105
DATERD_MD(时钟数据的读取) ·····5-104
DATEWR_MD(时钟数据的写入) ·····5-104
DBAND_MD(32位数据死区控制) ·····5-100
DBCD_M(32位BIN→BCD转换) ·····5-23
DBCDDA_S_MD(BCD8位→10进制ASCII转换)
·····5-75
DBDIVID_M(BCD8位的除法) ·····5-18
DBIN_M(32位BCD→BIN转换) ·····5-24
DBINDA_S_MD(32位BIN→10进制ASCII转换)
·····5-73
DBINHA_S_MD(32位BIN→16进制ASCII转换)
·····5-74
DBL_M(16位BIN→32位BIN转换) ·····5-27
DBMINUS_3_M(BCD8位的减法(3软元件)) ···5-16
DBMINUS_M(BCD8位的减法(2软元件)) ·····5-16
DBMULTI_M(BCD8位的乘法) ·····5-18
DBPLUS_3_M(BCD8位的加法(3软元件)) ····5-15
DBPLUS_M(BCD8位的加法(2软元件)) ·····5-15
DCML_M(32位数据否定传送) ····· 5-33
DDABCD_S_MD(10进制ASCII→
BCD8位转换) ·····5-78
DDABIN_S_MD(10进制ASCII→
32位BIN转换) ·····5-76
DDEC_M(32位BIN递减) ·····5-22
DEC_M(递减) ····· 5-21
DECO_M(编译) ····· 5-61
DEG_E_MD(浮动小数点弧度→
角度转换) ····· 5-91
DELETE(_E)(从字符串的指定位置删除) · 6-75
DELTA_M(直接输出的脉冲化) ·····5-7
DFLT_M(32位BIN→浮动小数点转换) ·· 5-26
DFRO_M(特殊功能模块2字数据读取) ·· 5-71
DGBIN_M(32位格雷码→BIN转换) ···· 5-29
DGRY_M(32位BIN→格雷码转换) ···· 5-28
DHABIN_S_MD(16进制ASCII→
32位BIN转换) ·····5-77
DI_M(中断禁止) ·····5-37
DINC_M(32位BIN递增) ·····5-22
DINT ·····3-3
DINT ·····
DINT_E_MD(32位浮动小数点→BIN转换)
·····5-25
DINT_TO_BOOL(_E)(双精度整数型(DINT)→
布尔型(BOOL)转换) ·····6-6
DINT_TO_INT(_E)(双精度整数型(DINT)→
整数型(INT)转换) ·····6-7
DINT_TO_REAL(_E)(双精度整数型(DINT)→
实数型(REAL)转换) ·····6-8
DINT_TO_STR(_E)(双精度整数型(DINT)→
字符串型(STRING)转换) ·····6-9
DIS_M(16位数据的4位分离) ·····5-62
DIV_E(除法) ·····6-34
DLIMIT_MD(32位数据上下限极限控制) ····5-99
DMAX_M(32位数据最大值查找) ·····5-66
DMIN_M(32位数据最小值查找) ·····5-67
DNEG_M(32位BIN的2的补数) ·····5-30
DOR_3_M(32位数据逻辑和(3软元件)) ·· 5-43
DOR_M(32位数据逻辑和(2软元件)) ··· 5-42
DRCL_M(32位数据左旋转(包含进位标志))
····· 5-52
DRCR_M(32位数据右旋转(包含进位标志))
····· 5-51
DROL_M(32位数据左旋转(不包含进位标志))
····· 5-52

索引 - 3

DROR_M(32位数据右旋转(不包含进位标志)) ・・・・・・・・・・・・・・・・・・・・・ 5-51
DSER_M(32位数据查找) ··············5-59
DSFL_M(1字左移) ··················5-55
DSFR_M(1字右移) ··················5-55
DSORT_M(32位数据排序) ··············5-68
DSTR_S_MD(32位BIN→字符串转换) ·······5-80
DSUM_M(32位数据位校验) ·············5-60
DTEST_MD(32位数据的位测试) ・・・・・ 5-57
DTO_M(特殊功能模块2字数据写入) ・・・ 5-72
DVAL_S_MD(字符串→32位BIN转换) ・・・ 5-81
DWSUM_M(32位数据合计值计算) ・・・・ 5-69
DXCH_M(32位数据替换) ・・・・・・・・ 5-35
DXNR_3_M(32位数据否定排他逻辑和(3软元件)) ・・・・・・・・・・・・・・・・・・・・ 5-48
DXNR_M(32位数据否定排他逻辑和(2软元件)) ・・・・・・・・・・・・・・・・・・・・ 5-47
DXOR_3_M(32位数据排他逻辑和(3软元件)) ・・・・・・・・・・・・・・・・・・・・ 5-45
DXOR_M(32位数据排他逻辑和(2软元件)) ・・・・・・・・・・・・・・・・・・・・・ 5-45
DZONE_MD(32位数据位域控制) ·········5-101
大于等于号(>=)(GE_E) ···············6-59
大于号(>)(GT_E) ···················6-57
代入(MOVE(_E)) ····················6-38
等号(=)(EQ_E) ·····················6-61
低速型(TIMER_M) ····················5-4
递减(DEC_M) ·······················5-21
递增(INC_M) ・・・・・・・・・・・・・ 5-21
多路调制器(MUX(_E)) ・・・・・・・・・ 6-55

[E]
EI_M(中断允许) ・・・・・・・・・・・・ 5-37
EMOD_M(浮动小数点→BCD分解) ・・・・ 5-86
ENCO_M(编码) ・・・・・・・・・・・・ 5-61
ENEG_M(浮动小数点的2的补数) ・・・・ 5-31
EQ_E(等号(=)) ・・・・・・・・・・・ 6-61
EREXP_M(BCD格式数据→浮动小数点) ・ 5-87
ESTR_M(浮动小数点→字符串转换) ・・・ 5-82
EVAL_M(字符串→浮动小数点转换) ・・・ 5-82
EXIT语句 ・・・・・・・・・・・・・・・ 4-21
EXP(_E)(自然指数) ・・・・・・・・・ 6-24
EXP_E_MD(浮动小数点指数运算) ・・・ 5-92
EXPT(_E)(指数) ····················6-36
二进制的选择(SEL(_E)) ··············6-47

[F]
FIND(_E)(从字符串的指定位置查找) ·····6-77
FLT_M(BIN→浮动小数点转换) ··········5-26
FMOV_M(同一数据块传送) ·············5-34
FOR・・・DO语句 ·····················4-15
FROM_M(特殊功能模块1字数据读取) ・・ 5-71
否定排他逻辑和(2软元件)(WXNR_M) ・・ 5-46
否定排他逻辑和(3软元件)(WXNR_3_M) ・ 5-47
浮动小数点→BCD分解(EMOD_M) ・・・・ 5-86
浮动小数点→BIN转换(INT_E_MD) ・・・ 5-25
浮动小数点→字符串转换(ESTR_M) ・・・ 5-82
浮动小数点COS-1运算(ACOS(_E)) ・・・ 6-29
浮动小数点COS-1运算(ACOS_E_MD) ・・・ 5-90
浮动小数点COS运算(COS(_E)) ・・・・・ 6-26
浮动小数点SIN-1运算(ASIN(_E)) ・・・ 6-28
浮动小数点SIN-1运算(ASIN_E_MD) ・・・ 5-83
浮动小数点SIN运算(SIN(_E)) ・・・・・ 6-25
浮动小数点SIN运算(SIN_E_MD) ・・・・ 5-88
浮动小数点TAN-1运算(ATAN(_E)) ・・・ 6-30
浮动小数点TAN-1运算(ATAN_E_MD) ・・・ 5-90
浮动小数点TAN运算(TAN(_E)) ・・・・・ 6-27
浮动小数点TAN运算(TAN_E_MD) ・・・・ 5-89
浮动小数点的2的补数(ENEG_M) ・・・・ 5-31
浮动小数点弧度→角度转换(DEG_E_MD) ・ 5-91
浮动小数点角度→弧度(RAD_E_MD) ・・・ 5-91
浮动小数点平方根(SQR_E_MD) ・・・・・ 5-92
浮动小数点指数运算(EXP_E_MD) ・・・・ 5-92
浮动小数点自然对数运算(LOG_E_MD) ・・ 5-93

[G]
GBIN_M(格雷码→BIN转换) ・・・・・・ 5-29
GE_E(大于等于号(>=)) ···············6-59
GRY_M(BIN→格雷码转换) ·············5-28
GT_E(大于号(>)) ····················6-57
高低字节替换(SWAP_MD) ··············5-36
高速型(TIMER_H_M) ··················5-5
格雷码→BIN转换(GBIN_M) ············5-29
功能块的调用 ・・・・・・・・・・・・・ 4-29

[H]
HABIN_S_MD(16进制ASCII→BIN转换) ····5-77
HEX_S_MD(ASCII→BIN转换) ···········5-83
HOUR_M(时钟数据格式转换(秒→时、分、秒)) ································5-106
合计值计算(WSUM_M) ・・・・・・・・・ 5-68

索引 - 4

[I]
I/O刷新(RFS_M) ·····5-38
IF・・・ELSE条件语句 ·····4-9
IF・・・ELSIF条件语句 ·····4-10
IF・・・THEN条件语句 ·····4-7
INC_M(递增) ·····5-21
INSERT(_E)(至指定位置的字符串插入) ···6-74
INSTR_M(字符串查找) ·····5-86
INT ·····3-3
INT_E_MD(浮动小数点→BIN转换) ·····5-25
INT_TO_BOOL(_E)(整数型(INT)→
布尔型(BOOL)转换) ·····6-10
INT_TO_DINT(_E)(整数型(INT)→
双精度整数型(DINT)转换) ・・・・・・ 6-11
INT_TO_REAL(_E)(整数型(INT)→
实数型(REAL)转换) ·····6-12
INT_TO_STR(_E)(整数型(INT)→
字符串型(STRING)转换) ·····6-13

[J
计数器(COUNTER_M) ·····5-5
加法(ADD_E) ·····6-31
减法(SUB_E) ·····6-33
结构化数据类型 ·····3-3
绝对值(ABS(_E)) ·····6-21

[K]
块传送(BMOV_M) ·····5-34
块数据比较(<)(BKCMP_LT_M) ·····5-12
块数据比较(<=)(BKCMP_LE_M) ・・・・・ 5-11
块数据比较(<>)(BKCMP_NE_M) ・・・・・ 5-10
块数据比较(=)(BKCMP_EQ_M) ・・・・・ 5-10
块数据比较(>)(BKCMP_GT_M) ・・・・・ 5-11
块数据比较(>=)(BKCMP_GE_M) ・・・・・ 5-12
块数据否定排他逻辑和(BKXNR_M) ・・・ 5-48
块数据逻辑和(BKOR_M) ・・・・・・・・ 5-43
块数据逻辑积(BKAND_M) ・・・・・・・ 5-41
块数据排他逻辑和(BKXOR_M) ・・・・・ 5-46
块数据替换(BXCH_M) ・・・・・・・・・ 5-36
块转换BCD→BIN转换(BKBIN_M) ・・・・ 5-32
块转换BIN→BCD转换(BKBCD_M ・・・・・ 5-31

[L]
LE_E(小于等于号(<=)) ····· 6-63
LEFT(_E)(从字符串的开始位置截取) ····· 6-70
LEFT_M(从字符串左侧截取) ····· 5-84
LEN(_E)(字符串长度截取) ····· 6-69
LEN_S_MD(字符串的长度检测) ·····5-79
LIMIT(_E)(限制器) ·····6-53
LIMIT_MD(上下限极限控制) ·····5-99
LN(_E)(自然对数) ・・・・・・・・・・・・ 6-23
LOG_E_MD(浮动小数点自然对数运算) ・・・・ 5-93
LT_E(小于号(<)) ·····6-65
逻辑否定(NOT(_E)) ·····6-46
逻辑和(2软元件)(WOR_M) ·····5-41
逻辑和(3软元件)(WOR_3_M) ·····5-42
逻辑和(AND_E) ·····6-43
逻辑积(2软元件)(WAND_M) ·····5-39
逻辑积(3软元件)(WAND_3_M) ・・・・・ 5-39
逻辑积(OR_E) ·····6-44

[M]
MAX(_E)(最大值) ·····6-49
MAX_M(数据最大值查找) ·····5-65
MID(_E)(从字符串的指定位置截取) ·····6-72
MIDR_M(字符串中的任意截取) ·····5-85
MIDW_M(字符串中的任意替换) ·····5-85
MIN(_E)(最小值) ·····6-51
MIN_M(数据最小值查找) ·····5-66
MOD(_E)(余数) ·····6-35
MOVE(_E)(代入) ·····6-38
MUL_E(乘法) ·····6-32
MUX(_E)(多路调制器) ·····6-55

[N]
NDIS_M(任意数据的位分离) ·····5-63
NE_E(不等号(<>)) ·····6-67
NEG_M(16位BIN的2的补数) ·····5-30
NOT(_E)(逻辑否定) ·····6-46
NUNI_M(任意数据的位合并) ·····5-64
n位数据1位右移(BSFR_M) ·····5-54
n位数据1位左移(BSFL_M) ·····5-54
n位右移(SFR_M) ·····5-53
n位左移(SFL_M) ·····5-53

[O]
OR_E(逻辑积) ·····6-44
OUT_M(软元件的输出) ·····5-4

[P]
PLOW_M(程序低速执行登录) ·····5-108
POFF_M(程序输出OFF待机) ·····5-107
PSCAN_M(程序扫描执行登录) ·····5-108
PSTOP_M(程序待机) ·····5-107

索引 - 5

排他逻辑和(2软元件)(WXOR_M) ・・・・ 5-44
排他逻辑和(3软元件)(WXOR_3_M) ・・・ 5-44
排他逻辑和(XOR_E) ・・・・・・・・・ 6-45
平方根(SQRT(_E)) ・・・・・・・・・・・・・・・・・・・・6-22

[Q]
QCDSET_M(注释用文件的设置) ・・・・・5-103
QDRSET_M(文件寄存器用文件的设置) ・・5-102

[R]
RAD_E_MD(浮动小数点角度→弧度) ・・・ 5-91
RCL_M(左旋转(包含进位标志)) ・・・・ 5-50
RCR_M(右旋转(包含进位标志)) ・・・・ 5-49
REAL ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・3-3
REAL_TO_DINT(_E)(实数型(REAL)→
双精度整数型(DINT)转换) ・・・・・・・・・・・・・・6-14
REAL_TO_INT(_E)(实数型(REAL)→
整数型(INT)转换) ・・・・・・・・・・・・・・・・・・・・6-15
REAL_TO_STR(_E)(实数型(REAL)→
字符串型(STRING)转换) ・・・・・・・・・・・・・・・6-16
REPEAT・・・UNTIL语句 ・・・・・・・・・・・・・・・・4-18
REPLACE(_E)(从字符串的指定位置替换) ・・6-76
RETURN语句 ・・・・・・・・・・・・・・・・・・・・・・・・4-20
RFS_M(I/O刷新) ・・・・・・・・・・・・・・・・・・・・・・5-38
RIGHT(_E)(从字符串的终端截取) ・・・・・・・・6-71
RIGHT_M(从字符串右侧截取) ・・・・・・・・・・・・5-84
RND_M(随机数发生) ・・・・・・・・・・・・・・・・・・・5-93
ROL(_E)(左旋转) ・・・・・・・・・・・・・・・・・・・・・6-42
ROL_M(左旋转(不包含进位标志)) ・・・・・・・・5-50
ROR(_E)(右旋转) ・・・・・・・・・・・・・・・・・・・・・6-41
ROR_M(右旋转(不包含进位标志)) ・・・・・・・・5-49
RSET_MD(文件寄存器的块No.切换) ・・・・・・5-102
RST_M(软元件的复位) ・・・・・・・・・・・・・・・・・・5-6
任意数据的位分离(NDIS_M) ・・・・・・・・・・・・5-63
任意数据的位合并(NUNI_M) ・・・・・・・・・・・・5-64
软元件的1位移(SFT_M) ・・・・・・・・・・・・・・・・・5-8
软元件的复位(RST_M) ・・・・・・・・・・・・・・・・・・5-6
软元件的设置(SET_M) ・・・・・・・・・・・・・・・・・・5-6
软元件的输出(OUT_M) ・・・・・・・・・・・・・・・・・5-4
软元件的注释数据读取(COMRD_S_MD) ・・・・・5-79

[S]
SECOND_M(时钟数据格式转换
(时、分、秒→秒)) ・・・・・・・・・・・・・・・・・・・・5-106
SEG_M(7段编译) ・・・・・・・・・・・・・ 5-62
SEL(_E)(二进制的选择) ・・・・・・・ 6-47
SER_M(数据查找) ・・・・・・・・・・・・・・・・・・・・・5-59
SET_M(软元件的设置) ・・・・・・・・・・・・・・・・・・5-6
SFL_M(n位左移) ・・・・・・・・・・・・・・・・・・・・・・5-53
SFR_M(n位右移) ・・・・・・・・・・・・・・・・・・・・・・5-53
SFT_M(软元件的1位移) ・・・・・・・・・・・・・・・・・5-8
SHL(_E)(位左移) ・・・・・・・・・・・・・・・・・・・・・6-39
SHR(_E)(位右移) ・・・・・・・・・・・・・・・・・・・・・6-40
SIN(_E)(浮动小数点SIN运算) ・・・・・ 6-25
SIN_E_MD(浮动小数点SIN运算) ・・・・ 5-88
SORT_M(数据排序) ・・・・・・・・・・・・・・・・・・・・5-67
SQR_E_MD(浮动小数点平方根) ・・・・・・・・・・・5-92
SQRT(_E)(平方根) ・・・・・・・・・・・・・・・・・・・・6-22
SRND_M(系列变更) ・・・・・・・・・・・・・・・・・・・・5-94
STOP_M(停止) ・・・・・・・・・・・・・・・・・・・・・・・5-9
STR_S_MD(BIN→字符串转换) ・・・・・・・・・・・・5-80
STR_TO_BOOL(_E)(字符串型(STRING)→
布尔型(BOOL)转换) ・・・・・・・・・・・・・・・・・・・6-17
STR_TO_DINT(_E)(字符串型(STRING)→
双精度整数型(DINT)转换) ・・・・・・・・・・・・・・6-18
STR_TO_INT(_E)(字符串型(STRING)→
整数型(INT)转换) ・・・・・・・・・・ 6-19
STR_TO_REAL(_E)(字符串型(STRING)→
实数型(REAL)转换) ・・・・・・・・・ 6-20
STRING ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・3-3
STRING ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・3-3
STRING_PLUS_3_M(字符串数据合并(3软元件))
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・5-19
STRING_PLUS_M(字符串数据合并(2软元件))
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・5-19
STRUCT ・・・・・・・・・・・・・・・・・・・・・・・・・・・・3-3
SUB_E(减法) ・・・・・・・・・・・・・・・・・・・・・・・・6-33
SUM_M(位校验) ・・・・・・・・・・・・・・・・・・・・・・5-60
SWAP_MD(高低字节替换) ・・・・・・・・・・・・・・・・5-36
上下限极限控制(LIMIT_MD) ・・・・・・・・・・・・5-99
时钟数据的读取(DATERD_MD) ・・・・・・・・・・・5-104
时钟数据的加法(DATEPLUS_M) ・・・・・・・・・・5-105
时钟数据的减法(DATEMINUS_M) ・・・・・・・・・5-105
时钟数据的写入(DATEWR_MD) ・・・・・・・・・・・5-104
时钟数据格式转换(秒→时、分、秒)(HOUR_M)
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・5-106
时钟数据格式转换(时、分、秒→秒)(SECOND_M)
・・・・・・・・・・・・・・・・・・・・・・5-106
实数型(REAL)→双精度整数型(DINT)
转换(REAL_TO_DINT(_E)) ・・・・・・・・・・・・・・6-14
实数型(REAL)→整数型(INT)
转换(REAL_TO_INT(_E)) ・・・・・・・・・・・・・・・6-15
实数型(REAL)→字符串型(STRING)
转换(REAL_TO_STR(_E)) ・・・・・・・・・・・・・・・6-16

索引 - 6

数据查找(SER_M) ·····5-59
数据排序(SORT_M) ·····5-67
数据最大值查找(MAX_M) ·····5-65
数据最小值查找(MIN_M) ·····5-66
刷新(COM_M) ·····5-70
双精度整数型(DINT)→布尔型(BOOL)
转换(DINT_TO_BOOL(_E)) ·····6-6
双精度整数型(DINT)→实数型(REAL)
转换 DINT_TO_REAL(_E) ·····6-8
双精度整数型(DINT)→整数型(INT)
转换DINT_TO_INT(_E) ·····6-7
双精度整数型(DINT)→字符串型(STRING)
转换DINT_TO_STR(_E) ·····6-9
死区控制(BAND_MD) ·····5-100
随机数发生(RND_M) ·····5-93

[T]
TAN(_E)(浮动小数点TAN运算) ·····6-27
TAN_E_MD(浮动小数点TAN运算) ·····5-89
TEST_MD(字软元件的位测试) ·····5-57
TIMER_H_M(高速型) ·····5-5
TIMER_M(低速型) ·····5-4
TO_M(特殊功能模块1字数据写入) ·····5-72
停止(STOP_M) ·····5-9
同一数据块传送(FMOV_M) ·····5-34

[U]
UNI_M(16位数据的4位合并) ·····5-63

[V]
VAL_S_MD(字符串→BIN转换) ·····5-81

[W]
WAND_3_M(逻辑积(3软元件)) ·····5-39
WAND_M(逻辑积(2软元件)) ·····5-39
WDT_M(WDT复位) ·····5-109
WDT复位(WDT_M) ·····5-109
WHILE・・・DO语句 ·····4-17
WOR_3_M(逻辑和(3软元件)) ·····5-42
WOR_M(逻辑和(2软元件)) ·····5-41
WORD_M(32位BIN→16位BIN转换) ·····5-27
WSUM_M(合计值计算) ·····5-68
WTOB_MD(字节单位数据分离) ·····5-64
WXNR_3_M(否定排他逻辑和(3软元件)) ····5-47
WXNR_M(否定排他逻辑和(2软元件)) ·····5-46
WXOR_3_M(排他逻辑和(3软元件)) ·····5-44
WXOR_M(排他逻辑和(2软元件)) ·····5-44
位软元件批量复位(BKRST_M) ·····5-58
位数指定 ·····3-16
位校验(SUM_M) ·····5-60
位右移(SHR(_E)) ·····6-40
位域控制(ZONE_MD) ·····5-101
位指定 ·····3-16
位左移(SHL(_E)) ·····6-39
文件寄存器的块No.切换(RSET_MD) ·····5-102
文件寄存器用文件的设置(QDRSET_M) ····5-102

[X]
XCH_M(16位数据替换) ·····5-35
XOR_E(排他逻辑和) ·····6-45
系列变更(SRND_M) ·····5-94
限制器(LIMIT(_E)) ·····6-53
小于等于号(<=)(LE_E) ·····6-63
小于号(<)(LT_E) ·····6-65

[Y]
右旋转(ROR(_E)) ·····6-41
右旋转(包含进位标志)(RCR_M) ·····5-49
右旋转(不包含进位标志)(ROR_M) ·····5-49
余数(MOD(_E)) ·····6-35
运算符 ·····4-2

[Z]
ZONE_MD(位域控制) ·····5-101
整数型(INT)→布尔型(BOOL)
转换(INT_TO_BOOL(_E)) ·····6-10
整数型(INT)→实数型(REAL)
转换(INT_TO_REAL(_E)) ·····6-12
整数型(INT)→双精度整数型(DINT)
转换(INT_TO_DINT(_E)) ·····6-11
整数型(INT)→字符串型(STRING)
转换(INT_TO_STR(_E)) ·····6-13
直接输出的脉冲化(DELTA_M) ·····5-7
指数(EXPT(_E)) ·····6-36
至指定位置的字符串插入(INSERT(_E)) ···6-74
智能功能模块1字数据读取(FROM_M) ·····5-71
智能功能模块1字数据写入(TO_M) ·····5-72
智能功能模块2字数据读取(DFRO_M) ·····5-71
智能功能模块2字数据写入(DTO_M) ·····5-72
中断禁止(DI_M) ·····5-37
中断允许(EI_M) ·····5-37
注释 ·····4-32
注释用文件的设置(QCDSET_M) ·····5-103
字符串→32位BIN转换(DVAL_S_MD) ·····5-81

索引 - 7

字符串→BIN转换(VAL_S_MD) ·····5-81
字符串→浮动小数点转换(EVAL_M) ·····5-82
字符串查找(INSTR_M) ·····5-86
字符串长度截取(LEN(_E)) ·····6-69
字符串的长度检测(LEN_S_MD) ·····5-79
字符串的连接(CONCAT(_E)) ·····6-73
字符串数据合并(2软元件)(STRING_PLUS_M)
·····5-19
字符串数据合并(3软元件)(STRING_PLUS_3_M)
·····5-19
字符串型(STRING)→布尔型(BOOL)
转换(STR_TO_BOOL(_E)) ·····6-17
字符串型(STRING)→实数型(REAL)
转换(STR_TO_REAL(_E)) ·····6-20
字符串型(STRING)→双精度整数型(DINT)
转换(STR_TO_DINT(_E)) ·····6-18
字符串型(STRING)→整数型(INT)
转换(STR_TO_INT(_E)) ·····6-19
字符串中的任意截取(MIDR_M) ·····5-85
字符串中的任意替换(MIDW_M) ····· 5-85
字节单位数据分离(WTOB_MD) ····· 5-64
字节单位数据合并(BTOW_MD) ····· 5-65
字软元件的位测试(TEST_MD) ·····5-57
字软元件的位复位(BRST_M) ·····5-56
字软元件的位设置(BSET_M) ·····5-56
自然对数(LN(_E)) ·····6-23
自然指数(EXP(_E)) ·····6-24
最大值(MAX(_E)) ·····6-49
最小值(MIN(_E)) ·····6-51
左旋转(ROL(_E)) ····· 6-42
左旋转(包含进位标志)(RCL_M) ····· 5-50
左旋转(不包含进位标志)(ROL_M) ····· 5-50

索引 - 8

质保

使用之前请确认以下产品质保的详细说明。

1. 免费质保期限和免费质保范围

在免费质保期内使用本产品时如果出现任何属于三菱责任的故障或缺陷(以下称“故障”),则经销商或三菱服务公司将负责免费维修。

注意如果需要在国内现场或海外维修时，则要收取派遣工程师的费用。对于涉及到更换故障模块后的任何再试运转、维护或现场测试，三菱将不负任何责任。

[免费质保期限]

免费质保期限为自购买日或货到目的地日的一年内。

注意产品从三菱生产并出货之后，最长分销时间为6个月，生产后最长的免费质保期为18个月。维修零部件的免费质保期不得超过修理前的免费质保期。

[免费质保范围]

(1) 范围局限于按照使用手册、用户手册及产品上的警示标签规定的使用状态、使用方法和使用环境正常使用的情况下。

(2) 以下情况下，即使在免费质保期内，也要收取维修费用。

1. 因不适当存储或搬运、用户粗心或疏忽而引起的故障。因用户的硬件或软件设计而导致的故障。
2. 因用户未经批准对产品进行改造而导致的故障等。
3. 对于装有三菱产品的用户设备，如果根据现有的法定安全措施或工业标准要求配备必需的功能或结构后本可以避免的故障。
4. 如果正确维护或更换了使用手册中指定的耗材(电池、背光灯、保险丝等)后本可以避免的故障。
5. 因火灾或异常电压等外部因素以及因地震、雷电、大风和水灾等不可抗力而导致的故障。
6. 根据从三菱出货时的科技标准还无法预知的原因而导致的故障。
7. 任何非三菱或用户责任而导致的故障。

2. 产品停产后的有偿维修期限

(1) 三菱在本产品停产后的7年内受理该产品的有偿维修。
停产的消息将以三菱技术公告等方式予以通告。

(2) 产品停产后，将不再提供产品(包括维修零件)。

3. 海外服务

在海外，维修由三菱在当地的海外FA中心受理。注意各个FA中心的维修条件可能会不同。

4. 意外损失和间接损失不在质保责任范围内

无论是否在免费质保期内，对于任何非三菱责任的原因而导致的损失、机会损失、因三菱产品故障而引起的用户利润损失、无论能否预测的特殊损失和间接损失、事故赔偿、除三菱以外产品的损失赔偿、用户更换设备、现场机械设备的再调试、运行测试及其它作业等，三菱将不承担责任。

5. 产品规格的改变

目录、手册或技术文档中的规格如有改变，恕不另行通知。

6. 产品应用

(1) 在使用三菱MELSEC通用可编程控制器时，应该符合以下条件:即使在可编程控制器设备出现问题或故障时也不会导致重大事故，并且应在设备外部系统地配备能应付任何问题或故障的备用设备及失效保险功能。

(2) 三菱通用可编程控制器是以一般工业用途等为对象设计和制造的。因此，可编程控制器的应用不包括那些会影响公共利益的应用，如核电厂和其它由独立供电公司经营的电厂以及需要特殊质量保证的应用如铁路公司或用于公用设施目的的应用。

另外，可编程控制器的应用不包括航空、医疗应用、焚化和燃烧设备、载人设备、娱乐及休闲设施、安全装置等与人的生命财产密切相关以及在安全和控制系统方面需要特别高的可靠性时的应用。

然而，对于这些应用，假如用户咨询当地三菱代表机构，提供有特殊要求方案的大纲并提供满足特殊环境的所有细节及用户自主要求，则可以进行一些应用。

MELSEC-Q/L结构体编程手册

结构化文本篇

三菱电机自动化(中国)有限公司

地址：上海市黄浦区南京西路288号创兴金融中心17楼
邮编：200003
电话：021-23223030　　传真：021-23223000
网址：www.meas.cn

书号	SH(NA)-080907CHN-A(1004)STC
印号	STC-MELSEC-Q/L(ST)-PM(1004)