Foreword

This manual contains text, diagrams and explanations which will guide the reader through the safe and correct installation, use, and operation of the FX Series programmable controller function for structured programs. It should be read and understood before attempting to install or use the unit.

Store this manual in a safe place so that you can take it out and read it whenever necessary. Always forward it to the end user.

This manual confers no industrial property rights or any rights of any other kind, nor does it confer any patent licenses. Mitsubishi Electric Corporation cannot be held responsible for any problems involving industrial property rights which may occur as a result of using the contents noted in this manual.

© 2009 MITSUBISHI ELECTRIC CORPORATION
Outline Precautions

This manual provides information for the use of the FX Series Programmable Controllers. The manual has been written to be used by trained and competent personnel. The definition of such a person or persons is as follows;

a) Any engineer who is responsible for the planning, design and construction of automatic equipment using the product associated with this manual should be of a competent nature, trained and qualified to the local and national standards required to fulfill that role. These engineers should be fully aware of all aspects of safety with regards to automated equipment.

b) Any commissioning or service engineer must be of a competent nature, trained and qualified to the local and national standards required to fulfill that job. These engineers should also be trained in the use and maintenance of the completed product. This includes being completely familiar with all associated documentation for the said product. All maintenance should be carried out in accordance with established safety practices.

c) All operators of the completed equipment should be trained to use that product in a safe and coordinated manner in compliance to established safety practices. The operators should also be familiar with documentation which is connected with the actual operation of the completed equipment.

Note: the term 'completed equipment' refers to a third party constructed device which contains or uses the product associated with this manual

• This product has been manufactured as a general-purpose part for general industries, and has not been designed or manufactured to be incorporated in a device or system used in purposes related to human life.

• Before using the product for special purposes such as nuclear power, electric power, aerospace, medicine or passenger movement vehicles, consult with Mitsubishi Electric.

• This product has been manufactured under strict quality control. However when installing the product where major accidents or losses could occur if the product fails, install appropriate backup or failsafe functions in the system.

• When combining this product with other products, please confirm the standard and the code, or regulations with which the user should follow. Moreover, please confirm the compatibility of this product to the system, machine, and apparatus with which a user is using.

• If in doubt at any stage during the installation of the product, always consult a professional electrical engineer who is qualified and trained to the local and national standards. If in doubt about the operation or use, please consult the nearest Mitsubishi Electric representative.

• Since the examples indicated by this manual, technical bulletin, catalog, etc. are used as a reference, please use it after confirming the function and safety of the equipment and system. Mitsubishi Electric will accept no responsibility for actual use of the product based on these illustrative examples.

• This manual content, specification etc. may be changed without a notice for improvement.

• The information in this manual has been carefully checked and is believed to be accurate; however, you have noticed a doubtful point, a doubtful error, etc., please contact the nearest Mitsubishi Electric representative.

Registration

• Microsoft®, Windows® and Excel® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

• CompactFlash is a trademark of SanDisk Corporation in the United States and other countries.

• Ethernet is a trademark of Xerox Corporation.

• MODBUS® is a registered trademark of Schneider Electric SA.

• The company name and the product name to be described in this manual are the registered trademarks or trademarks of each company.
# Table of Contents

Positioning of This Manual....................................................................................................................... 8
Related Manuals........................................................................................................................................ 11
Generic Names and Abbreviations Used in Manuals ........................................................................ 14

## 1. Outline

1.1 Outline of Structured Programs and Programming Languages.................................................. 15
   1.1.1 Outline of structured programs .................................................................................................. 15
   1.1.2 Programming languages .......................................................................................................... 16
1.2 PLC Series and Programming Software Version........................................................................... 17
1.3 Cautions on Creation of Fundamental Programs ....................................................................... 17
   1.3.1 I/O processing and response delay .......................................................................................... 17
   1.3.2 Double output (double coil) operation and countermeasures ................................................. 18
   1.3.3 Circuits not available in structured ladder programs and countermeasures ........................... 19
   1.3.4 Handling of general flags ...................................................................................................... 19
   1.3.5 Handling of operation error flag ........................................................................................... 22

## 2. Function/Operator List

2.1 Type Conversion Functions............................................................................................................. 23
2.2 Standard Functions Of One Numeric Variable ........................................................................... 25
2.3 Standard Arithmetic Functions...................................................................................................... 25
2.4 Standard Bit Shift Functions......................................................................................................... 25
2.5 Standard Bitwise Boolean Functions............................................................................................ 25
2.6 Standard Selection Functions........................................................................................................ 26
2.7 Standard Comparison Functions.................................................................................................. 26
2.8 Standard Character String Functions........................................................................................... 26
2.9 Functions Of Time Data Types ..................................................................................................... 27
2.10 Standard Function Blocks............................................................................................................ 27
2.11 Operator..................................................................................................................................... 28
   2.11.1 Arithmetic operations ............................................................................................................ 28
   2.11.2 Logical operations ................................................................................................................ 28
   2.11.3 Comparison operations ....................................................................................................... 28

## 3. Function Construction

3.1 Applied Function Expression and Execution Type ....................................................................... 29
3.2 Labels......................................................................................................................................... 30
3.3 Device and Address....................................................................................................................... 33
3.4 EN and ENO............................................................................................................................... 34

## 4. How to Read Explanation of Functions

........................................................................................................................................................................ 36
5. Applied Functions (Type Conversion Functions) 38

5.1 BOOL_TO_INT(_E) / Bit data → word [signed] data conversion .................................................. 40
5.2 BOOL_TO_DINT(_E) / Bit data → double word [signed] data conversion .......................... 42
5.3 BOOL_TO_STR(_E) / Bit data → string data conversion .............................................................. 44
5.4 BOOL_TO_WORD(_E) / Bit data → word [unsigned]/bit string [16-bit] data conversion ........ 46
5.5 BOOL_TO_DWORD(_E) / Bit data → double word [unsigned]/bit string [32-bit] data conversion . 48
5.6 BOOL_TO_TIME(_E) / Bit data → time data conversion .............................................................. 50
5.7 INT_TO_DINT(_E) / Word [signed] data → double word [signed] data conversion .......... 52
5.8 DINT_TO_INT(_E) / Double word [signed] data → word [signed] data conversion ............ 54
5.9 INT_TO_BOOL(_E) / Word [signed] data → bit data conversion .............................................. 56
5.10 DINT_TO_BOOL(_E) / Double word [signed] data → bit data conversion ....................... 58
5.11 INT_TO_REAL(_E) / Word [signed] data → float (single precision) data conversion ....... 60
5.12 DINT_TO_REAL(_E) / Double word [signed] data → float (single precision) data conversion .............................................. 62
5.13 INT_TO_STR(_E) / Word [signed] data → string data conversion ......................................... 64
5.14 DINT_TO_STR(_E) / Double word [signed] data → string data conversion .................. 66
5.15 INT_TO_WORD(_E) / Word [signed] data → word [unsigned]/bit string [16-bit] data conversion .............................................. 68
5.16 DINT_TO_WORD(_E) / Double word [signed] data → word [unsigned]/bit string [16-bit] data conversion .............................................. 70
5.17 INT_TO_DWORD(_E) / Word [signed] data → double word [unsigned]/bit string [32-bit] data conversion .............................................. 72
5.18 DINT_TO_DWORD(_E) / Double word [signed] data → double word [unsigned]/bit string [32-bit] data conversion .............................................. 74
5.19 INT_TO_BCD(_E) / Word [signed] data → BCD data conversion ........................................... 76
5.20 DINT_TO_BCD(_E) / Double word [signed] data → BCD data conversion ...................... 78
5.21 INT_TO_TIME(_E) / Word [signed] data → time data conversion ................................... 80
5.22 DINT_TO_TIME(_E) / Double word [signed] data → time data conversion .................... 82
5.23 REAL_TO_INT(_E) / Float (single precision) data → word [signed] data conversion ....... 84
5.24 REAL_TO_DINT(_E) / Float (single precision) data → double word [signed] data conversion .............................................. 86
5.25 REAL_TO_WORD(_E) / Float (single precision) data → word [unsigned]/bit string [16-bit] data conversion .............................................. 88
5.26 WORD_TO_BOOL(_E) / Word [unsigned] / bit string [16-bit] data → bit data conversion ............ 91
5.27 DWORD_TO_BOOL(_E) / Double word [unsigned]/bit string [32-bit] data → bit data conversion .............................................. 93
5.28 WORD_TO_INT(_E) / Word [unsigned]/bit string [16-bit] data → word [signed] data conversion .............................................. 95
5.29 WORD_TO_DINT(_E) / Word [unsigned]/bit string [16-bit] data → double word [signed] data conversion .............................................. 97
5.30 DWORD_TO_INT(_E) / Double word [unsigned]/bit string [32-bit] data → Word [signed] data conversion .............................................. 99
5.31 DWORD_TO_DINT(_E) / Double word [unsigned]/bit string [32-bit] data → double word [signed] data conversion .............................................. 101
5.32 WORD_TO_DWORD(_E) / Word [unsigned]/bit string [16-bit] data → double word [unsigned]/bit string [32-bit] data conversion .............................................. 103
5.33 DWORD_TO_WORD(_E) / Double word [unsigned]/bit string [32-bit] data → word [unsigned]/bit string [16-bit] data conversion .............................................. 105
5.34 WORD_TO_TIME(_E) / Word [unsigned]/bit string [16-bit] data → time data conversion ............ 107
5.35 DWORD_TO_TIME(_E) / Double word [unsigned]/bit string [32-bit] data → time data conversion .............................................. 109
5.36 STR_TO_BOOL(_E) / String data → bit data conversion .................................................... 111
5.37 STR_TO_INT(_E) / String data → word [signed] data conversion ........................................ 113
5.38 STR_TO_DINT(_E) / String data → double word [signed] data conversion ......................... 115
5.39 STR_TO_REAL(_E) / String data → float (single precision) data conversion ..................... 117
5.40 STR_TO_TIME(_E) / String data → time data conversion .................................................. 120
5.41 BCD_TO_INT(_E) / BCD data → word [signed] data conversion ........................................ 122
5.42 BCD_TO_DINT(_E) / BCD data → double word [signed] data conversion ......................... 124
5.43 TIME_TO_BOOL(_E) / Time data → bit data conversion ..................................................... 126
5.44 TIME_TO_INT(_E) / Time data → word [signed] data conversion ...................................... 128
5.45 TIME_TO_DINT(_E) / Time data → double word [signed] data conversion ....................... 130
5.46 TIME_TO_STR(_E) / Time data → string data conversion .................................................. 132
5.47 TIME_TO_WORD(_E) / Time data → word [unsigned]/bit string [16-bit] data conversion .... 134
5.48 TIME_TO_DWORD(_E) / Time data → double word [unsigned]/bit string [32-bit] data conversion .......................................................... 136
5.49 BITARR_TO_INT(_E) / Bit array → Word [signed] type, word [unsigned]/bit string [16-bit] data conversion .......................................................... 138
5.50 BITARR_TO_DINT(_E) / Bit array → Double word [signed] type, double word [unsigned]/bit string [32-bit] data conversion ........................................ 140
5.51 INT_TO_BITARR(_E) / Word [signed] data, word [unsigned]/bit string [16-bit] data → bit array conversion ....................................................................... 142
5.52 DINT_TO_BITARR(_E) / Double word [signed] data, double word [unsigned]/bit string [32-bit] data → bit array conversion .................................................. 144
5.53 CPY_BITARR(_E) / Bit array copy ......................................................................................... 146
5.54 GET_BIT_OF_INT(_E) / Specified bit read of word [signed] data ....................................... 148
5.55 SET_BIT_OF_INT(_E) / Specified bit write of word [signed] data .................................... 150
5.56 CPY_BIT_OF_INT(_E) / Specified bit copy of word [signed] data ....................................... 152
5.57 GET_BOOL_ADDR / Acquisition of start data ...................................................................... 154
5.58 GET_INT_ADDR / Acquisition of start data .......................................................................... 155
5.59 GET_WORD_ADDR / Acquisition of start data ..................................................................... 156

6. Applied Functions (Standard Functions Of One Numeric Variable) 157

6.1 ABS(_E) / Absolute value ........................................................................................................ 158

7. Applied Functions (Standard Arithmetic Functions) 160

7.1 ADD_E / Addition ...................................................................................................................... 161
7.2 SUB_E / Subtraction ................................................................................................................. 163
7.3 MUL_E / Multiplication ............................................................................................................. 165
7.4 DIV_E / Division ......................................................................................................................... 167
7.5 MOD(_E) / Modulus operation .................................................................................................. 169
7.6 EXPT(_E) / Exponentiation ....................................................................................................... 171
7.7 MOVE(_E) / Move operation ..................................................................................................... 173

8. Applied Functions (Standard Bit Shift Functions) 175

8.1 SHL(_E) / Left shift .................................................................................................................. 176
8.2 SHR(_E) / Right shift ................................................................................................................ 178
# Table of Contents

## 9. Applied Functions (Standard Bitwise Boolean Functions)
180

9.1 AND_E / Logical product ................................................................. 181
9.2 OR_E / Logical sum ...................................................................... 183
9.3 XOR_E / exclusive logical sum ......................................................... 185
9.4 NOT(_E) / logical negation ............................................................. 187

## 10. Applied Functions (Standard Selection Functions)
189

10.1 SEL(_E) / Selection ......................................................................... 190
10.2 MAXIMUM(_E) / Maximum selection ............................................. 192
10.3 MINIMUM(_E) / Minimum selection .............................................. 194
10.4 LIMITATION(_E) / Upper/Lower limit control .............................. 196
10.5 MUX(_E) / Multiplexer ................................................................ 198

## 11. Applied Functions (Standard Comparison Functions)
200

11.1 GT_E / Comparison ...................................................................... 201
11.2 GE_E / Comparison ...................................................................... 203
11.3 EQ_E / Comparison ...................................................................... 205
11.4 LE_E / Comparison ...................................................................... 207
11.5 LT_E / Comparison ...................................................................... 209
11.6 NE_E / Comparison ...................................................................... 211

## 12. Applied Functions (Standard Character String Functions)
213

12.1 MID(_E) / Extract mid string .......................................................... 214
12.2 CONCAT(_E) / String concatenation .............................................. 217
12.3 INSERT(_E) / String insertion ....................................................... 219
12.4 DELETE(_E) / String deletion ...................................................... 222
12.5 REPLACE(_E) / String replacement .............................................. 224
12.6 FIND(_E) / Searches a character string ....................................... 227

## 13. Applied Functions (Functions Of Time Data Types)
230

13.1 ADD_TIME(_E) / Addition ............................................................ 231
13.2 SUB_TIME(_E) / Subtraction ....................................................... 233
13.3 MUL_TIME(_E) / Multiplication ................................................. 235
13.4 DIV_TIME(_E) / Division ............................................................. 237

## 14. Standard Function Blocks
239

14.1 R_TRIG(_E) / Rising edge detector ............................................. 240
14.2 F_TRIG(_E) / Falling edge detector ............................................. 242
14.3 CTU(_E) / Up counter ................................................................ 244
14.4 CTD(_E) / Down counter ............................................................ 246
14.5 CTUD(_E) / Up/Down counter ...................................................... 248
14.6 TP(_E), TP_10(_E) / Pulse timer .................................................. 250
14.7 TON(_E), TON_10(_E) / On delay timer ........................................ 252
14.8 TOF(_E), TOF_10(_E) / Off delay timer ........................................ 254
## 14. Application Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.9</td>
<td>COUNTER_FB_M / Counter function blocks</td>
<td>256</td>
</tr>
<tr>
<td>14.10</td>
<td>TIMER_10_FB_M / Timer function blocks</td>
<td>258</td>
</tr>
<tr>
<td>14.11</td>
<td>TIMER_CONT_FB_M / Timer function blocks</td>
<td>259</td>
</tr>
<tr>
<td>14.12</td>
<td>TIMER_100_FB_M / Timer function blocks</td>
<td>261</td>
</tr>
</tbody>
</table>

## 15. Operator

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1</td>
<td>ADD / Addition</td>
<td>263</td>
</tr>
<tr>
<td>15.2</td>
<td>SUB / Subtraction</td>
<td>264</td>
</tr>
<tr>
<td>15.3</td>
<td>MUL / Multiplication</td>
<td>265</td>
</tr>
<tr>
<td>15.4</td>
<td>DIV / Division</td>
<td>266</td>
</tr>
<tr>
<td>15.5</td>
<td>MOD / Modulus operation</td>
<td>267</td>
</tr>
<tr>
<td>15.6</td>
<td>** / Exponentiation</td>
<td>268</td>
</tr>
<tr>
<td>15.7</td>
<td>AND / Logical product</td>
<td>269</td>
</tr>
<tr>
<td>15.8</td>
<td>OR / Logical sum</td>
<td>270</td>
</tr>
<tr>
<td>15.9</td>
<td>XOR / Exclusive logical sum</td>
<td>271</td>
</tr>
<tr>
<td>15.10</td>
<td>NOT / Logical negation</td>
<td>273</td>
</tr>
<tr>
<td>15.11</td>
<td>GT / Comparison</td>
<td>274</td>
</tr>
<tr>
<td>15.12</td>
<td>GE / Comparison</td>
<td>275</td>
</tr>
<tr>
<td>15.13</td>
<td>EQ / Comparison</td>
<td>276</td>
</tr>
<tr>
<td>15.14</td>
<td>LE / Comparison</td>
<td>277</td>
</tr>
<tr>
<td>15.15</td>
<td>LT / Comparison</td>
<td>278</td>
</tr>
<tr>
<td>15.16</td>
<td>NE / Comparison</td>
<td>279</td>
</tr>
</tbody>
</table>

## Appendix A: Correspondence between Devices and Addresses

Appendix A: Correspondence between Devices and Addresses 280

## Appendix B: Function/Operator List [by Type/in Alphabetic Order]

Appendix B: Function/Operator List [by Type/in Alphabetic Order] 282

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix B-1 [By type]</td>
<td>282</td>
<td></td>
</tr>
<tr>
<td>Appendix B-2 [In alphabetic order]</td>
<td>285</td>
<td></td>
</tr>
</tbody>
</table>

## Warranty

Warranty ........................................................................................................................................ 289

## Revision History

Revision History .......................................................................................................................... 290
Positioning of This Manual

This manual explains application functions for structured programs provided by GX Works2. Refer to other manuals for devices, parameters and sequence instructions. Refer to each corresponding manual for analog, communication, positioning control and special units and blocks.

1. When using FX3S/FX3G/FX3GC/FX3U/FX3UC PLCs

  This manual explains programming methods, specifications, functions, etc. required to create structured programs.

  This manual explains devices and parameters for structured programs provided by GX Works2.

  This manual explains sequence instructions for structured programs provided by GX Works2.

  This manual explains application functions for structured programs provided by GX Works2.

  This manual explains details of analog special function blocks and analog special adapters for FX3S/FX3G/FX3GC/FX3U/FX3UC PLCs and PID instruction. Explanation of instructions and instructions used in program examples are expressed for GX Developer.

  This manual explains details of simple N:N link, parallel link, computer link, no-protocol communication (RS and RS2 instructions), programming communication and inverter communication for FX PLCs. Explanation of instructions and instructions used in program examples are expressed for GX Developer.

  This manual explains details of wiring, instructions and operations of the positioning function built in FX3S/FX3G/FX3GC/FX3U/FX3UC PLC main units. Explanation of instructions and instructions used in program examples are expressed for GX Developer.

- Individual manuals (Manual supplied with product or additional Manual¹)
  This manual explains details of each special unit/block. Explanation of instructions and instructions used in program examples are expressed for GX Developer.

  *1. Detailed explanation may be provided by a separate manual in some products.
2. When using FX1S/FX1N/FX1NC/FX2N/FX2NC PLCs

MELSEC-Q/L/F Structured Programming Manual (Fundamentals)  (Additional Manual)
This manual explains programming methods, specifications, functions, etc. required to create
structured programs.

This manual explains devices and parameters for structured programs provided
by GX Works2.

This manual explains sequence instructions for structured programs provided
by GX Works2.

This manual explains application functions for structured programs provided
by GX Works2.

This manual explains details of simple N:N link, parallel link, computer link, no-protocol
communication (RS instruction), programming communication and inverter communication
for FX PLCs.
Explanation of instructions and instructions used in program examples are expressed for
GX Developer and FX-PCS/WIN.

Individual manuals  (Manual supplied with product or additional Manual*1)
This manual explains details of each special unit/block.
Explanation of instructions and instructions used in program examples are expressed
for GX Developer and FX-PCS/WIN.

*1. Detailed explanation may be provided by a separate manual in some products.
3. When using FX0S/FX0/FX0N/FXU/FX2C PLCs

MELSEC-Q/L/F Structured Programming Manual (Fundamentals)  (Additional Manual)
This manual explains programming methods, specifications, functions, etc. required to create structured programs.

This manual explains devices and parameters for structured programs provided by GX Works2.

This manual explains sequence instructions for structured programs provided by GX Works2.

(This manual)

This manual explains application functions for structured programs provided by GX Works2.

This manual explains details of parallel link, computer link, no-protocol communication (RS instruction) and programming communication for FX PLCs. Explanation of instructions and instructions used in program examples are expressed for GX Developer and FX-PCS/WIN.

Individual manuals  (Manual supplied with product or additional Manual *)
This manual explains details of each special unit/block. Explanation of instructions and instructions used in program examples are expressed for GX Developer and FX-PCS/WIN.

* Detailed explanation may be provided by a separate manual in some products.
Related Manuals

This manual explains devices and parameters for structured programs provided by GX Works2. Refer to other manuals for sequence instructions and applied functions.

This chapter introduces only reference manuals for this manual and manuals which describe the hardware information of PLC main units.

Manuals not introduced here may be required in some applications.
Refer to the manual of the used PLC main unit and manuals supplied together with used products.
Contact the representative for acquiring required manuals.

### Common among FX PLCs [structured]

<table>
<thead>
<tr>
<th>Manual name</th>
<th>Manual number</th>
<th>Supplied with product or Additional Manual</th>
<th>Contents</th>
<th>Model name code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MELSEC-Q/L/F Structured Programming Manual (Fundamentals)</td>
<td>SH-080782</td>
<td>Additional Manual</td>
<td>Programming methods, specifications, functions, etc. required to create structured programs</td>
<td>13JW06</td>
</tr>
<tr>
<td>FXCPU Structured Programming Manual [Device &amp; Common]</td>
<td>JY997D26001</td>
<td>Additional Manual</td>
<td>Devices, parameters, etc. provided in structured projects of GX Works2</td>
<td>09R925</td>
</tr>
</tbody>
</table>

### FX3s/FX3g/FX3gc/FX3u/FX3uc PLCs

<table>
<thead>
<tr>
<th>PLC main unit</th>
<th>Manual name</th>
<th>Manual number</th>
<th>Supplied with product or Additional Manual</th>
<th>Contents</th>
<th>Model name code</th>
</tr>
</thead>
<tbody>
<tr>
<td>FX3u Series User's Manual- Hardware Edition</td>
<td>JY997D16501</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3u PLC main unit.</td>
<td>09R516</td>
<td></td>
</tr>
<tr>
<td>FX3uc Series User's Manual - Hardware Edition</td>
<td>JY997D28701</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3uc PLC main unit.</td>
<td>09R519</td>
<td></td>
</tr>
<tr>
<td>FX3s Series Hardware Manual</td>
<td>JY997D33401</td>
<td>Supplied with product</td>
<td>I/O specifications, wiring and installation of the PLC main unit FX3s extracted from the FX3s Series User’s Manual - Hardware Edition. For detailed explanation, refer to the FX3s Series User’s Manual - Hardware Edition.</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>FX3s Series User's Manual- Hardware Edition</td>
<td>JY997D31301</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3s PLC main unit.</td>
<td>09R521</td>
<td></td>
</tr>
<tr>
<td>FX3cc Series User's Manual- Hardware Edition</td>
<td>JY997D45401</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3cc PLC main unit.</td>
<td>09R533</td>
<td></td>
</tr>
</tbody>
</table>
## FXCPU Structured Programming Manual

### [Application Functions]

<table>
<thead>
<tr>
<th>PLC main unit</th>
<th>Manual name</th>
<th>Manual number</th>
<th>Supplied with product or Additional Manual</th>
<th>Contents</th>
<th>Model name code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FX3S Series User's Manual - Hardware Edition</td>
<td>JY997D48601</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3S PLC main unit.</td>
<td>09R535</td>
</tr>
</tbody>
</table>

### Programming

<table>
<thead>
<tr>
<th>Programming</th>
<th>Manual name</th>
<th>Manual number</th>
<th>Supplied with product or Additional Manual</th>
<th>Contents</th>
<th>Model name code</th>
</tr>
</thead>
</table>

### FX1S/FX1N/FX1NC PLCs

#### FX2N/FX2NC PLCs [whose production is finished]

<table>
<thead>
<tr>
<th>PLC main unit</th>
<th>Manual name</th>
<th>Manual number</th>
<th>Supplied with product or Additional Manual</th>
<th>Contents</th>
<th>Model name code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FX1S HARDWARE MANUAL</td>
<td>JY992D83901</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1S PLC main unit.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FX1N HARDWARE MANUAL</td>
<td>JY992D89301</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1N PLC main unit.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>FX2N HARDWARE MANUAL</td>
<td>JY992D66301</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX2N PLC main unit.</td>
<td>09R508</td>
</tr>
<tr>
<td></td>
<td>FX1NC HARDWARE MANUAL</td>
<td>JY992D92101</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1NC PLC main unit. (Japanese only)</td>
<td>09R505</td>
</tr>
<tr>
<td></td>
<td>FX2NC HARDWARE MANUAL</td>
<td>JY992D76401</td>
<td>Additional Manual</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX2NC PLC main unit.</td>
<td>09R509</td>
</tr>
</tbody>
</table>

### Programming

<table>
<thead>
<tr>
<th>Programming</th>
<th>Manual name</th>
<th>Manual number</th>
<th>Supplied with product or Additional Manual</th>
<th>Contents</th>
<th>Model name code</th>
</tr>
</thead>
</table>
## FX0S/FX0/FX0N/FXU/FX2C PLCs [whose production is finished]

<table>
<thead>
<tr>
<th>Manual name</th>
<th>Manual number</th>
<th>Supplied with product or Additional Manual</th>
<th>Contents</th>
<th>Model name code</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLC main unit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX0/FX0N HARDWARE MANUAL</td>
<td>JY992D47501</td>
<td>Supplied with product</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX0/FX0N PLC main unit.</td>
<td>-</td>
</tr>
<tr>
<td>FX0S HARDWARE MANUAL</td>
<td>JY992D55301</td>
<td>Supplied with product</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX0S PLC main unit.</td>
<td>-</td>
</tr>
<tr>
<td>FX/FX2C HARDWARE MANUAL</td>
<td>JY992D47401</td>
<td>Supplied with product</td>
<td>Details about the hardware including I/O specifications, wiring, installation and maintenance of the FXU/FX2C PLC main unit.</td>
<td>-</td>
</tr>
<tr>
<td>Programming</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Manuals of models whose production is finished

Production is finished for FX0S/FX0/FX0N/FXU/FX2C/FX2N/FX2NC PLCs.
## Generic Names and Abbreviations Used in Manuals

<table>
<thead>
<tr>
<th>Abbreviation/generic name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PLCs</strong></td>
<td></td>
</tr>
<tr>
<td>FX3U Series or FX3U PLC</td>
<td>Generic name of FX3U Series PLCs</td>
</tr>
<tr>
<td>FX3UC Series or FX3UC PLC</td>
<td>Generic name of FX3UC Series PLCs</td>
</tr>
<tr>
<td>FX3G Series or FX3G PLC</td>
<td>Generic name of FX3G Series PLCs</td>
</tr>
<tr>
<td>FX3GC Series or FX3GC PLC</td>
<td>Generic name of FX3GC Series PLCs</td>
</tr>
<tr>
<td>FX3S Series or FX3S PLC</td>
<td>Generic name of FX3S Series PLCs</td>
</tr>
<tr>
<td>FX2N Series or FX2N PLC</td>
<td>Generic name of FX2N Series PLCs</td>
</tr>
<tr>
<td>FX2NC Series or FX2NC PLC</td>
<td>Generic name of FX2NC Series PLCs</td>
</tr>
<tr>
<td>FX1N Series or FX1N PLC</td>
<td>Generic name of FX1N Series PLCs</td>
</tr>
<tr>
<td>FX1NC Series or FX1NC PLC</td>
<td>Generic name of FX1NC Series PLCs Only usable in Japan.</td>
</tr>
<tr>
<td>FX1S Series or FX1S PLC</td>
<td>Generic name of FX1S Series PLCs</td>
</tr>
<tr>
<td>FXU Series or FXU PLC</td>
<td>Generic name of FXu(FX, FX2) Series PLCs</td>
</tr>
<tr>
<td>FX2C Series or FX2C PLC</td>
<td>Generic name of FX2C Series PLCs</td>
</tr>
<tr>
<td>FX0N Series or FX0N PLC</td>
<td>Generic name of FX0N Series PLCs</td>
</tr>
<tr>
<td>FX0s Series or FX0s PLC</td>
<td>Generic name of FX0s Series PLCs</td>
</tr>
<tr>
<td>FX0 Series or FX0 PLC</td>
<td>Generic name of FX0 Series PLCs</td>
</tr>
<tr>
<td><strong>Special adapters</strong></td>
<td></td>
</tr>
<tr>
<td>CF card special adapter</td>
<td>Generic name of CF card special adapters</td>
</tr>
<tr>
<td>CF-ADP</td>
<td>FX3U-CF-ADP</td>
</tr>
<tr>
<td>Ethernet adapter</td>
<td>Abbreviated of FX3U-ENET-ADP</td>
</tr>
<tr>
<td><strong>Programming language</strong></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>Abbreviation of structured text language</td>
</tr>
<tr>
<td>Structured ladder</td>
<td>Abbreviation of ladder diagram language</td>
</tr>
<tr>
<td>FBD</td>
<td>Abbreviation of function block diagram language</td>
</tr>
<tr>
<td><strong>Manuals</strong></td>
<td></td>
</tr>
<tr>
<td>COMMUNICATION CONTROL EDITION</td>
<td>Abbreviation of FX Series User's Manual-DATA COMMUNICATION CONTROL EDITION</td>
</tr>
</tbody>
</table>
1. Outline

This manual explains applied functions for structured programs provided by GX Works2. Refer to a different manual for devices, parameters and sequence instructions. Refer to the following manual for labels, data types and programming languages for structured programs:

→ Q/L/F Structured Programming Manual (Fundamentals)

1.1 Outline of Structured Programs and Programming Languages

1.1.1 Outline of structured programs

You can construct two or more programs (program blocks) into one program. Because you can divide the entire machine processing into small sub processes and create a program for each sub process, you can efficiently create a program for a large system.

1. Structured program

Program structuring is a technique to divide the contents of control executed by the PLC CPU into hierarchical small units (blocks) of processing, and then construct a program. By using this technique, you can design a program while recognizing structuring of a sequence program.

Advantages of hierarchical program

• You can examine the outline of a program at first, and then design its details gradually.
• Program blocks located at the lowest level in the hierarchy are extremely simple and highly independent.

Advantages of program consisting of program blocks

• Because the processing of each block is clear, the entire program is easy to understand.
• The entire program can be divided into several blocks that are created by several people.
• The program reusability is improved, and the development efficiency is improved accordingly.

2. Improved reusability of programs

You can save program blocks in a library. Program resources in the library can be shared, and often used again.
1.1.2 Programming languages

The following programming languages can be used in each program block.

Graphic languages

1. Structured ladder language

This graphic language is created based on the relay circuit design technology. A circuit always starts from the bus line located on the left side. The structured ladder language consists of contacts, coils, functions and function blocks. These components are connected with vertical lines and horizontal lines.

2. FBD [Function Block Diagram language]

FBD is a graphic language easy to understand visually. You can easily create programs by connecting parts (functions and function blocks) for special processing, variables and constants along the flow of data and signals to improve the programming efficiency.

Text language

1. ST (Structured text) language

The ST language can describe control achieved by syntax using selective branches with conditional statements and repetition by repetitive statements in the same way as high-level languages such as C language.

By using the ST language, you can create simple programs easy to understand.

```
Y000=(X000 OR Y000) AND NOT X001;
IF X001 THEN
  D2:=D0; (*When X001 is ON, the contents of D0 are transferred to D2.*)
END_IF;
IF X002 THEN
  D4:=D4+1; (*When X002 is ON, the contents of D4 are added by "1".*
ELSE
  D6:=D6+1; (*When X002 is OFF, the contents of D6 are added by "1".*
END_IF;
```
1.2 PLC Series and Programming Software Version

<table>
<thead>
<tr>
<th>PLC Series</th>
<th>Software package name (model name)</th>
<th>GX Works2 version</th>
</tr>
</thead>
<tbody>
<tr>
<td>FX3U•FX3UC</td>
<td>GX Works2 (SW1DNC-GXW2-E)</td>
<td>Ver. 1.08J or later</td>
</tr>
<tr>
<td>FX3G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX2N•FX2NC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX1N•FX1NC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX1S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX0N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX0•FX0S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FX3GC</td>
<td></td>
<td>Ver. 1.77F or later</td>
</tr>
<tr>
<td>FX3S</td>
<td></td>
<td>Ver. 1.492N or later</td>
</tr>
</tbody>
</table>

1.3 Cautions on Creation of Fundamental Programs

This section explains cautions on programming. Refer to the following manual for cautions on structured programs and programming languages:

→ Q/L/F Structured Programming Manual (Fundamentals)

Refer to the following programming manual for detailed operations of and cautions on devices and parameters:

→ FX Structured Programming Manual [Device & Common]

1.3.1 I/O processing and response delay

1. Operation timing of I/O relays and response delay

FX PLCs execute the I/O processing by repeating the processing (1) to processing (3). Accordingly, the control executed by PLCs contains not only the drive time of input filters and output devices but also the response delay caused by the operation cycle.

Acquiring the latest I/O information

For acquiring the latest input information or immediately outputting the operation result in the middle of the operation cycle shown above, the I/O refresh instruction (REF) is available.

2. Short pulses cannot be received.

The ON duration and OFF duration of inputs in PLCs require longer time than "PLC cycle time + Input filter response delay".

When the response delay "10 ms" of the input filter is considered and the cycle time is supposed as "10 ms", the ON duration and OFF duration should be at least 20 ms respectively.

Accordingly, PLCs cannot handle input pulses at 25 Hz (= 1000 / (20 + 20)) or more. However, the situation can be improved by PLC special functions and applied instructions.

Convenient functions for improvement

By using the following functions, PLCs can receive pulses shorter than the operation cycle:

- High speed counter function
- Input interrupt function
- Pulse catch function
- Input filter value adjustment function
1.3.2 Double output (double coil) operation and countermeasures

This subsection explains the double output (double coil) operation and countermeasures.

1. Operation of double outputs

When a coil (output variable) is used twice (double coils) in another program block to be executed or in the same program block, the PLC gives priority to the latter coil.

Suppose that the same coil Y003 is used in two positions as shown in the right figure.
For example, suppose that X001 is ON and X002 is OFF.

In the first coil Y003, the image memory is set to ON and the output Y004 is also set to ON because the input X001 is ON.

In the second coil Y003, however, the image memory is set to OFF because the input X002 is OFF.

As a result, the actual output to the outside is "Y003: OFF, Y004: ON".

2. Countermeasures against double outputs

Double outputs (double coils) do not cause an illegal input error (program error), but the operation is complicated as described above.
Change the program as shown in the example below.

The SET and RST instructions or jump instruction can be used instead, or a same output coil can be programmed at each state using step ladder instructions STL and RET.
When you use the step ladder instructions STL and RET, note that the PLC regards it as double coils if you program, inside the state, an output coil located outside the RET instruction from another program block or the STL instruction.
1.3.3 Circuits not available in structured ladder programs and countermeasures

1. Bridge circuit
A circuit in which the current flows in both directions should be changed as shown in the right figure (so that a circuit without D and a circuit without B are connected in parallel).

2. Coil connection position
- You can program a contact on the right side of a coil. In this case, make sure to program a coil (including a function or function block) at the end of the circuit.

1.3.4 Handling of general flags
The following flags are valid in general sequence instructions:
(Examples)
M8020: Zero flag
M8021: Borrow flag
M8022: Carry flag
M8029: Instruction execution complete flag
M8090: Block comparison signal*1
M8032: Instruction non-execution flag*1
M8304: Zero flag*2
M8306: Carry flag*2

*1. Supported only in FX3U/FX3UC PLCs.
*2. Supported only in FX3S/FX3G/FX3GC/FX3U/FX3UC PLCs.

Each of these flags turns ON or OFF every time the PLC executes a corresponding instruction. These flags do not turn ON or OFF when the PLC does not execute a corresponding instruction or when an error occurs. Because these flags are related to many sequence instructions, their ON/OFF status changes every time the PLC executes each corresponding instruction. Refer to examples in the next page, and program a flag contact just under the target sequence instruction.
1. Program containing many flags (Example of instruction execution complete flag M8029)

If you program the instruction execution complete flag M8029 twice or more together for two or more sequence instructions which actuate the flag M8029, you cannot judge easily by which sequence instruction the flag M8029 is controlled. In addition, the flag M8029 does not turn ON or OFF correctly for each corresponding sequence instruction.

Refer to the next page when you would like to use the flag M8029 in any position other than the position just under the corresponding sequence instruction.
2. **Introduction of a method to use flags in any positions other than positions just under sequence instructions**

If two or more sequence instructions are programmed, general flags turn ON or OFF when each corresponding instruction is executed. Accordingly, when using a general flag in any position other than a position just under a sequence instruction, set to ON or OFF another device (variable) just under the sequence instruction, and then use the contact of such device (variable) as the command contact.
1.3.5 Handling of operation error flag

When there is an error in the instruction construction, target device or target device number range and an error occurs while operation is executed, the following flag turns ON and the error information is stored.

1. Operation error

<table>
<thead>
<tr>
<th>Error flag</th>
<th>Device which stores error code</th>
<th>Device which stores error occurrence step</th>
</tr>
</thead>
<tbody>
<tr>
<td>M8067</td>
<td>D8067</td>
<td>D8069&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D8315, D8314</td>
</tr>
</tbody>
</table>

*1. When an error occurs in a step up to the step No. 32767 in the FX3U/FX3UC PLC, you can check the error occurrence step also in D8069 (16 bits).

- When an operation error has occurred, M8067 turns ON, D8067 stores the operation error code, and the specified device (shown in the table above) stores the error occurrence step.
- When another error occurs in another step, the stored data is updated in turn to the error code and step number of the new error. (These devices are set to OFF when errors are cleared.)
- When the PLC mode changes from STOP to RUN, these devices are cleared instantaneously, and then turn ON again if errors have not been cleared.

2. Operation error latch

<table>
<thead>
<tr>
<th>Error flag</th>
<th>Device which stores error code</th>
<th>Device which stores error occurrence step</th>
</tr>
</thead>
<tbody>
<tr>
<td>M8068</td>
<td>-</td>
<td>D8068&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D8313, D8312</td>
</tr>
</tbody>
</table>

*2. When an error occurs in a step up to the step No. 32767 in the FX3U/FX3UC PLC, you can check the error occurrence step also in D8068 (16 bits).

- When an operation error has occurred, M8068 turns ON, and the device shown in the table above stores the error occurrence step.
- Even if another error occurs in another step, the stored data is not updated and remains held until these devices are forcibly set to OFF or until the power is turned OFF.
# 2. Function/Operator List

This chapter introduces a list of functions and operators available in programming.

## 2.1 Type Conversion Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_INT(E)</td>
<td>Converts bit data into word [signed] data.</td>
<td>FX3U(C)</td>
<td>Section 5.1</td>
</tr>
<tr>
<td>BOOL_TO_DINT(E)</td>
<td>Converts bit data into double word [signed] data.</td>
<td>FX3G(C)</td>
<td>Section 5.2</td>
</tr>
<tr>
<td>BOOL_TO_STR(E)</td>
<td>Converts bit data into string data.</td>
<td>FX3S</td>
<td>Section 5.3</td>
</tr>
<tr>
<td>BOOL_TO_WORD(E)</td>
<td>Converts bit data into word [unsigned]/bit string [16-bit] data.</td>
<td>FX2N(C)</td>
<td>Section 5.4</td>
</tr>
<tr>
<td>BOOL_TO_DWORD(E)</td>
<td>Converts bit data into double word [unsigned]/bit string [32-bit] data.</td>
<td>FX1N(C)</td>
<td>Section 5.5</td>
</tr>
<tr>
<td>BOOL_TO_TIME(E)</td>
<td>Converts bit data into time data.</td>
<td>FX1S</td>
<td>Section 5.6</td>
</tr>
<tr>
<td>INT_TO_DINT(E)</td>
<td>Converts word [signed] data into double word [signed] data</td>
<td>FXU/FX2C</td>
<td>Section 5.7</td>
</tr>
<tr>
<td>DINT_TO_INT(E)</td>
<td>Converts double word [signed] data into word [signed] data.</td>
<td>FX0N</td>
<td>Section 5.8</td>
</tr>
<tr>
<td>INT_TO_BOOL(E)</td>
<td>Converts word [signed] data into bit data.</td>
<td>FX0(S)</td>
<td>Section 5.9</td>
</tr>
<tr>
<td>DINT_TO_BOOL(E)</td>
<td>Converts double word [signed] data into bit data.</td>
<td></td>
<td>Section 5.10</td>
</tr>
<tr>
<td>INT_TO_REAL(E)</td>
<td>Converts word [signed] data into float (single precision) data.</td>
<td></td>
<td>Section 5.11</td>
</tr>
<tr>
<td>DINT_TO_REAL(E)</td>
<td>Converts double word [signed] data into float (single precision) data.</td>
<td></td>
<td>Section 5.12</td>
</tr>
<tr>
<td>INT_TO_STR(E)</td>
<td>Converts word [signed] data into string data.</td>
<td></td>
<td>Section 5.13</td>
</tr>
<tr>
<td>DINT_TO_STR(E)</td>
<td>Converts double word [signed] data into string data.</td>
<td></td>
<td>Section 5.14</td>
</tr>
<tr>
<td>INT_TO_WORD(E)</td>
<td>Converts word [signed] data into word [unsigned]/bit string [16-bit] data.</td>
<td></td>
<td>Section 5.15</td>
</tr>
<tr>
<td>DINT_TO_WORD(E)</td>
<td>Converts double word [signed] data into double word [unsigned]/bit string [16-bit] data.</td>
<td></td>
<td>Section 5.16</td>
</tr>
<tr>
<td>INT_TO_DWORD(E)</td>
<td>Converts word [signed] data into double word [unsigned]/bit string [16-bit] data.</td>
<td></td>
<td>Section 5.17</td>
</tr>
<tr>
<td>DINT_TO_DWORD(E)</td>
<td>Converts double word [signed] data into double word [unsigned]/bit string [32-bit] data.</td>
<td></td>
<td>Section 5.18</td>
</tr>
<tr>
<td>INT_TO_BCD(E)</td>
<td>Converts word [signed] data into BCD data.</td>
<td></td>
<td>Section 5.19</td>
</tr>
<tr>
<td>DINT_TO_BCD(E)</td>
<td>Converts double word [signed] data into BCD data.</td>
<td></td>
<td>Section 5.20</td>
</tr>
<tr>
<td>INT_TO_TIME(E)</td>
<td>Converts word [signed] data into time data.</td>
<td></td>
<td>Section 5.21</td>
</tr>
<tr>
<td>DINT_TO_TIME(E)</td>
<td>Converts double word [signed] data into time data.</td>
<td></td>
<td>Section 5.22</td>
</tr>
<tr>
<td>REAL_TO_INT(E)</td>
<td>Converts float (single precision) data into word [signed] data.</td>
<td></td>
<td>Section 5.23</td>
</tr>
<tr>
<td>REAL_TO_DINT(E)</td>
<td>Converts float (single precision) data into double word [signed] data.</td>
<td></td>
<td>Section 5.24</td>
</tr>
<tr>
<td>REAL_TO_STR(E)</td>
<td>Converts float (single precision) data into string data.</td>
<td></td>
<td>Section 5.25</td>
</tr>
<tr>
<td>WORD_TO_BOOL(E)</td>
<td>Converts word [unsigned]/bit string [16-bit] data into bit data.</td>
<td></td>
<td>Section 5.26</td>
</tr>
</tbody>
</table>

*1. The function is provided in the FX3G Series Ver.1.10 or later.
<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>FX3U(C)</th>
<th>FX3G(C)</th>
<th>FX3S</th>
<th>FX2N(C)</th>
<th>FX1N(C)</th>
<th>FX1S</th>
<th>FXU/FX2C</th>
<th>FX0N</th>
<th>FX0(S)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_TO_BOOL(_E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into bit data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.27</td>
</tr>
<tr>
<td>WORD_TO_INT(_E)</td>
<td>Converts word [unsigned]/bit string [16-bit] data into word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.28</td>
</tr>
<tr>
<td>WORD_TO_DINT(_E)</td>
<td>Converts word [unsigned]/bit string [16-bit] data into double word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.29</td>
</tr>
<tr>
<td>DWORD_TO_INT(_E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.30</td>
</tr>
<tr>
<td>DWORD_TO_DINT(_E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into double word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.31</td>
</tr>
<tr>
<td>WORD_TO_DWORD(_E)</td>
<td>Converts word [unsigned]/bit string [16-bit] data into double word [unsigned]/bit string [32-bit] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.32</td>
</tr>
<tr>
<td>DWORD_TOWORD(_E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into word [unsigned]/bit string [16-bit] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.33</td>
</tr>
<tr>
<td>WORD_TO_TIME(_E)</td>
<td>Converts word [unsigned]/bit string [16-bit] data into time data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.34</td>
</tr>
<tr>
<td>DWORD_TO_TIME(_E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into time data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.35</td>
</tr>
<tr>
<td>STR_TO_BOOL(_E)</td>
<td>Converts string data into bit data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.36</td>
</tr>
<tr>
<td>STR_TO_INT(_E)</td>
<td>Converts string data into word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.37</td>
</tr>
<tr>
<td>STR_TO_DINT(_E)</td>
<td>Converts string data into double word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.38</td>
</tr>
<tr>
<td>STR_TO_REAL(_E)</td>
<td>Converts string data into float (single precision) data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.39</td>
</tr>
<tr>
<td>STR_TO_TIME(_E)</td>
<td>Converts string data into time data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.40</td>
</tr>
<tr>
<td>BCD_TO_INT(_E)</td>
<td>Converts BCD data into word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.41</td>
</tr>
<tr>
<td>BCD_TO_DINT(_E)</td>
<td>Converts BCD data into double word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.42</td>
</tr>
<tr>
<td>TIME_TO_BOOL(_E)</td>
<td>Converts time data into bit data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.43</td>
</tr>
<tr>
<td>TIME_TO_INT(_E)</td>
<td>Converts time data into word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.44</td>
</tr>
<tr>
<td>TIME_TO_DINT(_E)</td>
<td>Converts time data into double word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.45</td>
</tr>
<tr>
<td>TIME_TO_STR(_E)</td>
<td>Converts time data into string data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.46</td>
</tr>
<tr>
<td>TIME_TO_WORD(_E)</td>
<td>Converts time data into word [unsigned]/bit string [16-bit] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.47</td>
</tr>
<tr>
<td>TIME_TO_DWORD(_E)</td>
<td>Converts time data into double word [unsigned]/bit string [32-bit] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.48</td>
</tr>
<tr>
<td>BITARR_TO_INT(_E)</td>
<td>Converts specified number of bits of a bit array into word [signed] data or word [unsigned]/bit string [16-bit] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.49</td>
</tr>
<tr>
<td>BITARR_TO_DINT(_E)</td>
<td>Converts specified number of bits of a bit array into double word [signed] data or double word [unsigned]/bit string [32-bit] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.50</td>
</tr>
<tr>
<td>INT_TO_BITARR(_E)</td>
<td>Outputs low-order &quot;n&quot; bits of word [signed] data or word [unsigned]/bit string [16-bit] data to a bit array.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.51</td>
</tr>
<tr>
<td>DINT_TO_BITARR(_E)</td>
<td>Outputs low-order &quot;n&quot; bits of double word [signed] data or double word [unsigned]/bit string [32-bit] data to a bit array.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.52</td>
</tr>
<tr>
<td>CPY_BITARR(_E)</td>
<td>Copies specified number of bits of a bit array.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.53</td>
</tr>
<tr>
<td>GET_BIT_OF_INT(_E)</td>
<td>Reads a value of a specified bit of word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.54</td>
</tr>
<tr>
<td>SET_BIT_OF_INT(_E)</td>
<td>Writes a value to a specified bit of word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.55</td>
</tr>
<tr>
<td>CPY_BIT_OF_INT(_E)</td>
<td>Copies a specified bit of word [signed] data to a specified bit of another word [signed] data.</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Section 5.56</td>
</tr>
</tbody>
</table>
## 2.2 Standard Functions Of One Numeric Variable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_BOOL_ADDR</td>
<td>Outputs start data as bit data.</td>
<td>FX2(C)</td>
<td>Section 5.57</td>
</tr>
<tr>
<td>GET_INT_ADDR</td>
<td>Outputs start data as word [signed] data.</td>
<td>FX3U(C)</td>
<td>Section 5.58</td>
</tr>
<tr>
<td>GET_WORD_ADDR</td>
<td>Outputs start data as word [unsigned]/bit string [16-bit] data.</td>
<td>FX3S</td>
<td>Section 5.59</td>
</tr>
</tbody>
</table>

### 2.3 Standard Arithmetic Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(_E)</td>
<td>Obtains the absolute value.</td>
<td>FX3U(C)</td>
<td>Section 6.1</td>
</tr>
<tr>
<td>ADD_E</td>
<td>Adds data. (Number of pins variable)</td>
<td>FX3G(C)</td>
<td>Section 7.1</td>
</tr>
<tr>
<td>SUB_E</td>
<td>Subtracts data.</td>
<td>FX3S</td>
<td>Section 7.2</td>
</tr>
<tr>
<td>MUL_E</td>
<td>Multiplies data. (Number of pins variable)</td>
<td>FX2N(C)</td>
<td>Section 7.3</td>
</tr>
<tr>
<td>DIV_E</td>
<td>Divides data and outputs the quotient.</td>
<td>FX1N(C)</td>
<td>Section 7.4</td>
</tr>
<tr>
<td>MOD(_E)</td>
<td>Divides data and outputs the remainder.</td>
<td>FX1S</td>
<td>Section 7.5</td>
</tr>
<tr>
<td>EXPT(_E)</td>
<td>Obtains the raised result.</td>
<td>FXU/FX2C</td>
<td>Section 7.6</td>
</tr>
<tr>
<td>MOVE(_E)</td>
<td>Transfers data.</td>
<td>FX0N</td>
<td>Section 7.7</td>
</tr>
</tbody>
</table>

### 2.4 Standard Bit Shift Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHL(_E)</td>
<td>Shifts bits leftward.</td>
<td>FX3U(C)</td>
<td>Section 8.1</td>
</tr>
<tr>
<td>SHR(_E)</td>
<td>Shifts bits rightward.</td>
<td>FX3G(C)</td>
<td>Section 8.2</td>
</tr>
</tbody>
</table>

### 2.5 Standard Bitwise Boolean Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND_E</td>
<td>Obtains the logical product. (Number of pins variable)</td>
<td>FX3S</td>
<td>Section 9.1</td>
</tr>
<tr>
<td>OR_E</td>
<td>Obtains the logical sum. (Number of pins variable)</td>
<td>FX2N(C)</td>
<td>Section 9.2</td>
</tr>
<tr>
<td>XOR_E</td>
<td>Obtains the exclusive logical sum. (Number of pins variable)</td>
<td>FX1N(C)</td>
<td>Section 9.3</td>
</tr>
<tr>
<td>NOT_E</td>
<td>Obtains the logical not.</td>
<td>FX1S</td>
<td>Section 9.3</td>
</tr>
</tbody>
</table>
### 2.6 Standard Selection Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL(_E)</td>
<td>Selects data in accordance with the input condition.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 10.1</td>
</tr>
<tr>
<td>MAXIMUM(_E)</td>
<td>Searches the maximum value. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 10.2</td>
</tr>
<tr>
<td>MINIMUM(_E)</td>
<td>Searches the minimum value. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 10.3</td>
</tr>
<tr>
<td>LIMITATION(_E)</td>
<td>Judges whether data is located within the range between the upper limit value and the lower limit value.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 10.4</td>
</tr>
<tr>
<td>MUX(_E)</td>
<td>Selects data, and outputs it. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 10.5</td>
</tr>
</tbody>
</table>

### 2.7 Standard Comparison Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT_E</td>
<td>Compares data with regard to &quot;&gt; (larger)&quot;. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 11.1</td>
</tr>
<tr>
<td>GE_E</td>
<td>Compares data with regard to &quot;≥ (larger or equal)&quot;. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 11.2</td>
</tr>
<tr>
<td>EQ_E</td>
<td>Compares data with regard to &quot;= (equal)&quot;. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 11.3</td>
</tr>
<tr>
<td>LE_E</td>
<td>Compares data with regard to &quot;≤ (smaller or equal)&quot;. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 11.4</td>
</tr>
<tr>
<td>LT_E</td>
<td>Compares data with regard to &quot;&lt; (smaller)&quot;. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 11.5</td>
</tr>
<tr>
<td>NE_E</td>
<td>Compares data with regard to &quot;≠ (unequal)&quot;.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 11.6</td>
</tr>
</tbody>
</table>

### 2.8 Standard Character String Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID(_E)</td>
<td>Obtains a character string from a specified position.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 12.1</td>
</tr>
<tr>
<td>CONCAT(_E)</td>
<td>Connects character strings. (Number of pins variable)</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 12.2</td>
</tr>
<tr>
<td>INSERT(_E)</td>
<td>Inserts a character string.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 12.3</td>
</tr>
<tr>
<td>DELETE(_E)</td>
<td>Deletes a character string.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 12.4</td>
</tr>
<tr>
<td>REPLACE(_E)</td>
<td>Replaces a character string.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 12.5</td>
</tr>
<tr>
<td>FIND(_E)</td>
<td>Searches a character string.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 12.6</td>
</tr>
</tbody>
</table>
### 2.9 Functions Of Time Data Types

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TIME(_E)</td>
<td>Adds time data.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 13.1</td>
</tr>
<tr>
<td>SUB_TIME(_E)</td>
<td>Subtracts time data.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 13.2</td>
</tr>
<tr>
<td>MUL_TIME(_E)</td>
<td>Multiplies time data.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 13.3</td>
</tr>
<tr>
<td>DIV_TIME(_E)</td>
<td>Divides time data.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 13.4</td>
</tr>
</tbody>
</table>

### 2.10 Standard Function Blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Applicable PLC</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TRIG(_E)</td>
<td>Detects the rising edge of a signal, and outputs pulse signal.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.1</td>
</tr>
<tr>
<td>F_TRIG(_E)</td>
<td>Detects the falling edge of a signal, and outputs pulse signal.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.2</td>
</tr>
<tr>
<td>CTU(_E)</td>
<td>Counts up the number of times of rising of a signal.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.3</td>
</tr>
<tr>
<td>CTD(_E)</td>
<td>Counts down the number of times of rising of a signal.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.4</td>
</tr>
<tr>
<td>CTUD(_E)</td>
<td>Counts up/down the number of times of rising of a signal.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.5</td>
</tr>
<tr>
<td>TP(_E)</td>
<td>Keeps ON a signal during specified time duration.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.6</td>
</tr>
<tr>
<td>TON(_E)</td>
<td>Keeps OFF a signal during specified time duration.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.7</td>
</tr>
<tr>
<td>TOF(_E)</td>
<td>Turns OFF the output signal at specified time after the input signal turned OFF.</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.8</td>
</tr>
<tr>
<td>COUNTER_FB_M</td>
<td>Counter drive</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.9</td>
</tr>
<tr>
<td>TIMER_10_FB_M</td>
<td>10ms timer drive</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.10</td>
</tr>
<tr>
<td>TIMER_CONT_FB_M</td>
<td>Retentive timer drive</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.11</td>
</tr>
<tr>
<td>TIMER_100_FB_M</td>
<td>100ms timer drive</td>
<td>✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔</td>
<td>Section 14.12</td>
</tr>
</tbody>
</table>
## 2.11 Operator

### 2.11.1 Arithmetic operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Function</th>
<th>Applicable PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structured ladder /FBD</strong></td>
<td><strong>ST</strong></td>
<td><strong>FI</strong></td>
</tr>
<tr>
<td>ADD</td>
<td>+</td>
<td>Adds data. (Number of pins variable)</td>
</tr>
<tr>
<td>SUB</td>
<td>-</td>
<td>Subtracts data.</td>
</tr>
<tr>
<td>MUL</td>
<td>*</td>
<td>Multiplies data. (Number of pins variable)</td>
</tr>
<tr>
<td>DIV</td>
<td>/</td>
<td>Divides data (, and outputs the quotient).</td>
</tr>
<tr>
<td>-</td>
<td>MOD</td>
<td>Divides data (, and outputs the remainder).</td>
</tr>
<tr>
<td>-</td>
<td>**</td>
<td>Obtains the raised result.</td>
</tr>
</tbody>
</table>

### 2.11.2 Logical operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Function</th>
<th>Applicable PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structured ladder /FBD</strong></td>
<td><strong>ST</strong></td>
<td><strong>FI</strong></td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>Obtains the logical sum. (Number of pins variable)</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>Obtains the exclusive logical sum. (Number of pins variable)</td>
</tr>
<tr>
<td>-</td>
<td>NOT</td>
<td>Obtains the logical not.</td>
</tr>
</tbody>
</table>

### 2.11.3 Comparison operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Function</th>
<th>Applicable PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structured ladder /FBD</strong></td>
<td><strong>ST</strong></td>
<td><strong>FI</strong></td>
</tr>
<tr>
<td>GT</td>
<td>&gt;</td>
<td>Compares data with regard to ”&gt;” (larger). (Number of pins variable)</td>
</tr>
<tr>
<td>GE</td>
<td>&gt;=</td>
<td>Compares data with regard to “≥” (larger or equal). (Number of pins variable)</td>
</tr>
<tr>
<td>EQ</td>
<td>=</td>
<td>Compares data with regard to “= (equal). (Number of pins variable)</td>
</tr>
<tr>
<td>LE</td>
<td>&lt;=</td>
<td>Compares data with regard to “≤” (smaller or equal). (Number of pins variable)</td>
</tr>
<tr>
<td>LT</td>
<td>&lt;</td>
<td>Compares data with regard to “&lt;” (smaller). (Number of pins variable)</td>
</tr>
<tr>
<td>NE</td>
<td>&lt;&gt;</td>
<td>Compares data with regard to “≠” (unequal).</td>
</tr>
</tbody>
</table>
3. Function Construction

This chapter explains the construction of applied functions.

3.1 Applied Function Expression and Execution Type

Applied function and argument

- The name expressing the contents is given to each function. For example, the function name "SHL (bit shift left)" is given.
- Each function consists of arguments which indicate I/O data used in the function.

\[\text{IN} (\square): \text{An argument whose contents do not change even if the function is executed is called "source", and expressed in this symbol.}\]

\[\star 1 (\square): \text{An argument whose contents change when the function is executed is called "destination", and expressed in this symbol.}\]

\[\text{K1 (\square)}: \text{Arguments not regarded as source or destination are expressed in "m", "n", etc.}\]

Argument target devices

- The input variable (label or device) specifies the target.
- Bit device themselves such as X, Y, M and S may be handled.
- Bit devices may be combined in a way "KnX", "KnY", "KnM" and "KnS" to express numeric data.

\[\text{FX Structured Programming Manual} \ [\text{Device & Common}]\]

- Current value registers of data registers (D), timers (T) and counters (C) may be handled.
- When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects.
  Use labels when handling 32-bit data.
  You can specify 32-bit counters directly, however, because they have 32-bit length. Use global labels when specifying devices.
  When 32-bit data is handled, two consecutive 16-bit data registers D are combined.
  For example, when data register D0 is defined as an argument of a 32-bit instruction by a label, 32-bit data stored in D1 and D0 is handled. (D1 offers high-order 16 bits, and D0 offers low-order 16-bits.)
  When the current value register of a timer or counter is used as a general data register, it is handled in the same way.

Changing the number of arguments (pins)

- With certain functions, the number of sources can be changed in the range from 2 to 28.
  The functions which can be changed is indicated by "(Number of pins variable)” in the function list table.

\[\rightarrow \text{Refer to Section 2. Function List}\]
3.2 Labels

Label types
Labels are classified into two types, global and local.
- Global labels can be used in program components and function blocks.
- Local labels can be used only in declared program blocks.

Label class
The label class indicates how each label can be used from which program component. The table below shows label classes.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Applicable program component</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR_GLOBAL</td>
<td>Common label available in all program components</td>
<td>Program block</td>
</tr>
<tr>
<td>VAR_GLOBAL_CONSTANT</td>
<td>Common constant available in all program components</td>
<td>✓</td>
</tr>
<tr>
<td>VAR</td>
<td>Label available within declared program components, and not available in any other program component</td>
<td>✓</td>
</tr>
<tr>
<td>VAR_CONSTANT</td>
<td>Constant available within declared program components, and not available in any other program component</td>
<td>✓</td>
</tr>
<tr>
<td>VAR_INPUT</td>
<td>Label which receives a value, and cannot be changed in program components</td>
<td>✓</td>
</tr>
<tr>
<td>VAR_OUTPUT</td>
<td>Label output from a function block</td>
<td></td>
</tr>
<tr>
<td>VAR_IN_OUT</td>
<td>Local label which receives a value, outputs it from a program component, and can be changed in program components</td>
<td></td>
</tr>
</tbody>
</table>

Label definition
It is necessary to define a label to use the label. An error will occur when a program in which labels are not defined is converted (compiled).
- When defining a global label, set the label name, class and data type, and assigns a device.
- When defining a local label, set the label name, class and data type. You do not have to specify devices for local labels. Assignment of devices is automatically executed during compiling.

In the example below, the label "Var_String1" is set for the function "BOOL_TO_STR_E".

```
 X000 BOOL_TO_STR_E
 EN ENO
 M0 BOOL
 Var_String1
```

- When using "Var_String1" as a global label
  Set the class, label name, data type and device (or address).

- When using "Var_String1" as a local label
  Set the class, label name and data type.
Constant description method

The table below describes the method required to set a constant to a label.

<table>
<thead>
<tr>
<th>Constant type</th>
<th>Description method</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Input &quot;TRUE&quot; or &quot;FALSE&quot;. Or input &quot;0&quot; or &quot;1&quot;.</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Binary number</td>
<td>Add &quot;2#&quot; before a binary number.</td>
<td>2#0010, 2#01101010</td>
</tr>
<tr>
<td>Octal number</td>
<td>Add &quot;8#&quot; before an octal number.</td>
<td>8#0, 8#337</td>
</tr>
<tr>
<td>Decimal number</td>
<td>Input a decimal number directly. Or add &quot;K&quot; before a decimal number.</td>
<td>123, K123</td>
</tr>
<tr>
<td>Hexadecimal number</td>
<td>Add &quot;16#&quot; or &quot;H&quot; before a hexadecimal number.</td>
<td>16#FF, HFF</td>
</tr>
<tr>
<td>Real number</td>
<td>Input a real number directly. Or add &quot;E&quot; before a real number.</td>
<td>2.34, E2.34</td>
</tr>
<tr>
<td>Character string</td>
<td>Surround a character string with single quotations (') or double quotations (&quot;').</td>
<td>'ABC', &quot;ABC&quot;</td>
</tr>
</tbody>
</table>

Data type

The label data type is basic or universal.

- The table below shows a list of basic data types.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Value range</th>
<th>Bit length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Boolean data</td>
<td>0(FALSE), 1(TRUE)</td>
<td>1 bit</td>
</tr>
<tr>
<td>Word [signed]</td>
<td>Integer</td>
<td>-32768 to 32767</td>
<td>16 bits</td>
</tr>
<tr>
<td>Double Word [signed]</td>
<td>Double precision integer</td>
<td>-2147483648 to 2147483647</td>
<td>32 bits</td>
</tr>
<tr>
<td>Word [unsigned]/Bit String [16-bit]</td>
<td>16-bit data</td>
<td>0 to 65535</td>
<td>16 bits</td>
</tr>
<tr>
<td>Double Word [unsigned]/Bit String [32-bit]</td>
<td>32-bit data</td>
<td>0 to 4294967295</td>
<td>32 bits</td>
</tr>
<tr>
<td>FLOAT (Single Precision)</td>
<td>Real number</td>
<td>E ±1.175495×10^-38 to E ±3.402823×10^38 (Number of significant figures: 6)</td>
<td>32 bits</td>
</tr>
<tr>
<td>String</td>
<td>Character string</td>
<td>(50 characters maximum)</td>
<td>Variable</td>
</tr>
<tr>
<td>Time</td>
<td>Time value</td>
<td>T#-24d-0h31m23s648.00ms to T#24dd20h31m23s647.00ms</td>
<td>32 bits</td>
</tr>
</tbody>
</table>
• The universal data type indicates data type of a label which combines several basic data types. The data type name begins with "ANY".

- The "ANY" type on a higher layer contains types on the lower layer.
- The "ANY" type on the top layer contains all types.

*1 Refer to the following manual for details.
→ Q/L/F Structured Programming Manual (Fundamentals)
### 3.3 Device and Address

Devices can be described in two methods, device method and address method.

#### Device method

In this method, a device is described using the device name and device number.

- **Device name**
- **Device number**

#### Address method

This method is defined in IEC61131-3, and used as shown in the table below.

<table>
<thead>
<tr>
<th>Head</th>
<th>1st character: Position</th>
<th>2nd character: Size</th>
<th>3rd and later characters: Classification</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>I Input (Omitted)</td>
<td>Bit</td>
<td>This number is provided for detailed classification. Period (.) is used to delimit the subsequent &quot;Number&quot;. The characters for classification may be omitted.</td>
<td>This decimal number corresponds to the device number.</td>
</tr>
<tr>
<td></td>
<td>Q Output</td>
<td>X Bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M Internal</td>
<td>W Word (16 bits)</td>
<td>D Double word (32 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>L Long Word (64 bits)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Memory area position**
  - The memory area position in which data is assigned is classified into "input", "output" or "internal".
  - X (X Device method) : I(Input)
  - Y (Y Device method) : Q(Output)
  - Any other device : M(Internal)

- **Size**
  - The principle of the description method corresponding to the device method (MELSEC description method) is as follows:
    - Bit device : X(Bit)
    - Word device: W(Word (16 bits)), D(Double word (32 bits))

- **Classification**
  - The 3rd and later characters indicate the device type which cannot be specified only by the position and size explained above.
  - The classification is not required for devices "X" and "Y".
  - Refer to the following for the device description method:

→ 7.3 Appendix A
3.4 EN and ENO

Execution of an instruction can be controlled when the instruction contains "EN" in its name.

- "EN" inputs the instruction execution condition.
- "ENO" outputs the instruction execution status.
- The table below shows the "ENO" status corresponding to the "EN" status and the operation result.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>Operation result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Executes operation.)</td>
<td>TRUE (Operation error did not occur.)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation error occurred.)</td>
<td>FALSE</td>
<td>Indefinite value</td>
</tr>
<tr>
<td>FALSE (Stops operation.)</td>
<td>FALSE</td>
<td>Indefinite value</td>
</tr>
</tbody>
</table>

In the above example, the function "BOOL_TO_STR_E" is executed only when X000 is "TRUE". When the function is executed normally, "TRUE" is output to M1.
<table>
<thead>
<tr>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>Outline</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>Function Operator List</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>Function Construction</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>How to Read Explanation of Functions</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>Applied Functions (Type Conversion Functions)</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>Applied Functions (Standard Functions of One Numeric Variable)</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>Applied Functions (Standard Arithmetic Functions)</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>Applied Functions (Standard Bit Shift Functions)</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>Applied Functions (Standard Bitwise Boolean Functions)</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>Applied Functions (Standard Selection Functions)</td>
</tr>
</tbody>
</table>

**MEMO**
4. How to Read Explanation of Functions

Function explanation pages have the following configuration.

1. **Outline**
   - This function compares data with regard to "\( \geq \) (larger or equal)".

2. **1. Format**
   - Function explanation pages have the following configuration.

3. **2. Set data**
   - In explanation of functions, I/O variables inside () are described.

4. **Explanation of function and operation**
   - a) This function outputs "TRUE" when all comparison results are \( \geq \).
   - b) This function outputs "FALSE" when any comparison result is \( < \).
   - The number of pins for \( \geq \) can be changed in the range of 2 to 26.
      - Referring to Section 3. Function Construction

5. **Cautions**
   - 1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   - You can specify 32-bit counters directly, however, because they are 32-bit devices.
   - Use global labels when specifying labels.
   - 2) When the FLOAT (Single Precision) data to \( \geq \) is set from the programming tool, a rounding error may be generated.
      - Refer to the MELSEC-Q/LF Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

6. **Program example**
   - In this program, the contents of devices specified in \( \geq \) and \( \leq \) are compared, and the operation result is output to a device specified in \( \geq \).

   ```
   [Structured ladder/FBD]
   q_int1 = "GE_E",
   "EN": EN, "IN": IN1, "IN2": IN2,
   "OUT": OUT1,
   q_int2 = "GE_E",
   "EN": EN, "IN": IN1, "IN2": IN2,
   "OUT": OUT2
   ```

   * The above page is prepared for explanation, and is different from the actual page.
4 How to Read Explanation of Functions

1) Indicates the chapter/section/subsection number and instruction name.
2) Indicates PLCs which support the function.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>○</td>
<td>The PLC Series supports the function from its first product.</td>
</tr>
<tr>
<td>△</td>
<td>The supporting status varies on the version. Applicable versions are explained in &quot;Cautions&quot;.</td>
</tr>
<tr>
<td>×</td>
<td>The PLC Series does not support the function.</td>
</tr>
</tbody>
</table>

3) Indicates the expression of each function.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured ladder /FBD</td>
<td>Indicates the instruction expression in the structured ladder language adopted as the representative.</td>
</tr>
<tr>
<td>ST</td>
<td>Indicates the instruction expression in the ST language.</td>
</tr>
</tbody>
</table>

4) Indicates the input variable name and output variable name of the function as well as the contents and data type of each variable. Refer to the following for detailed data types: → Q/L/F Structured Programming Manual (Fundamentals)

5) Explanation of function and operation
The function executed by this function is explained. In explanation, the structured ladder language is used as the representative.

6) Cautions
Cautions on using the function are described.

7) Program example
Program examples are explained in each language. In program examples of the structured ladder/FBD language, the structured ladder language is adopted as the representative.
### 5. Applied Functions (Type Conversion Functions)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_INT(_E)</td>
<td>Bit data → word [signed] data conversion</td>
<td>Section 5.1</td>
</tr>
<tr>
<td>BOOL_TO_DINT(_E)</td>
<td>Bit data → double word [signed] data conversion</td>
<td>Section 5.2</td>
</tr>
<tr>
<td>BOOL_TO_STR(_E)</td>
<td>Bit data → string data conversion</td>
<td>Section 5.3</td>
</tr>
<tr>
<td>BOOL_TO_WORD(_E)</td>
<td>Bit data → word [unsigned]/bit string [16-bit] data conversion</td>
<td>Section 5.4</td>
</tr>
<tr>
<td>BOOL_TO_DWORD(_E)</td>
<td>Bit data → double word [unsigned]/bit string [32-bit] data conversion</td>
<td>Section 5.5</td>
</tr>
<tr>
<td>BOOL_TO_TIME(_E)</td>
<td>Bit data → time data conversion</td>
<td>Section 5.6</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>Word [signed] data → double word [signed] data conversion</td>
<td>Section 5.7</td>
</tr>
<tr>
<td>DINT_TO_INT(_E)</td>
<td>Double word [signed] data → word [signed] data conversion</td>
<td>Section 5.8</td>
</tr>
<tr>
<td>INT_TO_BOOL(_E)</td>
<td>Word [signed] data → bit data conversion</td>
<td>Section 5.9</td>
</tr>
<tr>
<td>DINT_TO_BOOL(_E)</td>
<td>Double word [signed] data → bit data conversion</td>
<td>Section 5.10</td>
</tr>
<tr>
<td>INT_TO_REAL(_E)</td>
<td>Word [signed] data → float (single precision) data conversion</td>
<td>Section 5.11</td>
</tr>
<tr>
<td>DINT_TO_REAL(_E)</td>
<td>Double word [signed] data → float (single precision) data conversion</td>
<td>Section 5.12</td>
</tr>
<tr>
<td>INT_TO_STR(_E)</td>
<td>Word [signed] data → string data conversion</td>
<td>Section 5.13</td>
</tr>
<tr>
<td>DINT_TO_STR(_E)</td>
<td>Double word [signed] data → string data conversion</td>
<td>Section 5.14</td>
</tr>
<tr>
<td>INT_TO_WORD(_E)</td>
<td>Word [signed] data → word [unsigned]/bit string [16-bit] data conversion</td>
<td>Section 5.15</td>
</tr>
<tr>
<td>DINT_TO_WORD(_E)</td>
<td>Double word [signed] data → word [unsigned]/bit string [16-bit] data conversion</td>
<td>Section 5.16</td>
</tr>
<tr>
<td>INT_TO_DWORD(_E)</td>
<td>Word [signed] data → double word [unsigned]/bit string [32-bit] data conversion</td>
<td>Section 5.17</td>
</tr>
<tr>
<td>DINT_TO_DWORD(_E)</td>
<td>Double word [signed] data → double word [unsigned]/bit string [32-bit] data conversion</td>
<td>Section 5.18</td>
</tr>
<tr>
<td>INT_TO_BCD(_E)</td>
<td>Word [signed] data → BCD data conversion</td>
<td>Section 5.19</td>
</tr>
<tr>
<td>DINT_TO_BCD(_E)</td>
<td>Double word [signed] data → BCD data conversion</td>
<td>Section 5.20</td>
</tr>
<tr>
<td>INT_TO_TIME(_E)</td>
<td>Word [signed] data → time data conversion</td>
<td>Section 5.21</td>
</tr>
<tr>
<td>DINT_TO_TIME(_E)</td>
<td>Double word [signed] data → time data conversion</td>
<td>Section 5.22</td>
</tr>
<tr>
<td>REAL_TO_INT(_E)</td>
<td>Float (single precision) data → word [signed] data conversion</td>
<td>Section 5.23</td>
</tr>
<tr>
<td>REAL_TO_DINT(_E)</td>
<td>Float (single precision) data → double word [signed] data conversion</td>
<td>Section 5.24</td>
</tr>
<tr>
<td>REAL_TO_STR(_E)</td>
<td>Float (single precision) data → string data conversion</td>
<td>Section 5.25</td>
</tr>
<tr>
<td>WORD_TO_BOOL(_E)</td>
<td>Word [unsigned]/bit string [16-bit] data → bit data conversion</td>
<td>Section 5.26</td>
</tr>
<tr>
<td>DWORD_TO_BOOL(_E)</td>
<td>Double word [unsigned]/bit string [32-bit] data → bit data conversion</td>
<td>Section 5.27</td>
</tr>
<tr>
<td>WORD_TO_INT(_E)</td>
<td>Word [unsigned]/bit string [16-bit] data → word [signed] data conversion</td>
<td>Section 5.28</td>
</tr>
<tr>
<td>WORD_TO_DINT(_E)</td>
<td>Word [unsigned]/bit string [16-bit] data → double word [signed] data conversion</td>
<td>Section 5.29</td>
</tr>
<tr>
<td>DWORD_TO_INT(_E)</td>
<td>Double word [unsigned]/bit string [32-bit] data → Word [signed] data conversion</td>
<td>Section 5.30</td>
</tr>
<tr>
<td>DWORD_TO_DINT(_E)</td>
<td>Double word [unsigned]/bit string [32-bit] data → double word [signed] data conversion</td>
<td>Section 5.31</td>
</tr>
<tr>
<td>WORD_TO_DWORD(_E)</td>
<td>Word [unsigned]/bit string [16-bit] data → double word [unsigned]/bit string [32-bit] data conversion</td>
<td>Section 5.32</td>
</tr>
<tr>
<td>DWORD_TO_WORD(_E)</td>
<td>Double word [unsigned]/bit string [32-bit] data → word [unsigned]/bit string [16-bit] data conversion</td>
<td>Section 5.33</td>
</tr>
<tr>
<td>WORD_TO_TIME(_E)</td>
<td>Word [unsigned]/bit string [16-bit] data → time data conversion</td>
<td>Section 5.34</td>
</tr>
<tr>
<td>DWORD_TO_TIME(_E)</td>
<td>Double word [unsigned]/bit string [32-bit] data → time data conversion</td>
<td>Section 5.35</td>
</tr>
<tr>
<td>STR_TO_BOOL(_E)</td>
<td>String data → bit data conversion</td>
<td>Section 5.36</td>
</tr>
<tr>
<td>STR_TO_INT(_E)</td>
<td>String data → word [signed] data conversion</td>
<td>Section 5.37</td>
</tr>
<tr>
<td>STR_TO_DINT(_E)</td>
<td>String data → double word [signed] data conversion</td>
<td>Section 5.38</td>
</tr>
<tr>
<td>STR_TO_REAL(_E)</td>
<td>String data → float (single precision) data conversion</td>
<td>Section 5.39</td>
</tr>
<tr>
<td>STR_TO_TIME(_E)</td>
<td>String data → time data conversion</td>
<td>Section 5.40</td>
</tr>
<tr>
<td>BCD_TO_INT(_E)</td>
<td>BCD data → word [signed] data conversion</td>
<td>Section 5.41</td>
</tr>
<tr>
<td>BCD_TO_DINT(_E)</td>
<td>BCD data → double word [signed] data conversion</td>
<td>Section 5.42</td>
</tr>
<tr>
<td>TIME_TO_BOOL(_E)</td>
<td>Time data → bit data conversion</td>
<td>Section 5.43</td>
</tr>
<tr>
<td>TIME_TO_INT(_E)</td>
<td>Time data → word [signed] data conversion</td>
<td>Section 5.44</td>
</tr>
<tr>
<td>TIME_TO_DINT(_E)</td>
<td>Time data → double word [signed] data conversion</td>
<td>Section 5.45</td>
</tr>
<tr>
<td>Function name</td>
<td>Function</td>
<td>Reference</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>TIME_TO_STR(_E)</td>
<td>Time data → string data conversion</td>
<td>Section 5.46</td>
</tr>
<tr>
<td>TIME_TO_WORD(_E)</td>
<td>Time data → word [unsigned]/bit string [16-bit] data conversion</td>
<td>Section 5.47</td>
</tr>
<tr>
<td>TIME_TO_DWORD(_E)</td>
<td>Time data → double word [unsigned]/bit string [32-bit] data conversion</td>
<td>Section 5.48</td>
</tr>
<tr>
<td>BITARR_TO_INT(_E)</td>
<td>Bit array → Word [signed] type, word [unsigned]/bit String [16-bit] data conversion</td>
<td>Section 5.49</td>
</tr>
<tr>
<td>BITARR_TO_DINT(_E)</td>
<td>Bit array → Double word [signed] type, double word [unsigned]/bit string [32-bit] data conversion</td>
<td>Section 5.50</td>
</tr>
<tr>
<td>INT_TO_BITARR(_E)</td>
<td>Word [signed] data, word [unsigned]/bit string [16-bit] data → bit array conversion</td>
<td>Section 5.51</td>
</tr>
<tr>
<td>DINT_TO_BITARR(_E)</td>
<td>Double word [signed] data, double word [unsigned]/bit string [32-bit] data → bit array conversion</td>
<td>Section 5.52</td>
</tr>
<tr>
<td>CPY_BITARR(_E)</td>
<td>Bit array copy</td>
<td>Section 5.53</td>
</tr>
<tr>
<td>GET_BIT_OF_INT(_E)</td>
<td>Specified bit read of word [signed] data</td>
<td>Section 5.54</td>
</tr>
<tr>
<td>SET_BIT_OF_INT(_E)</td>
<td>Specified bit write of word [signed] data</td>
<td>Section 5.55</td>
</tr>
<tr>
<td>CPY_BIT_OF_INT(_E)</td>
<td>Specified bit copy of word [signed] data</td>
<td>Section 5.56</td>
</tr>
<tr>
<td>GET_BOOL_ADDR</td>
<td>Acquisition of start data</td>
<td>Section 5.57</td>
</tr>
<tr>
<td>GET_INT_ADDR</td>
<td>Acquisition of start data</td>
<td>Section 5.58</td>
</tr>
<tr>
<td>GET_WORD_ADDR</td>
<td>Acquisition of start data</td>
<td>Section 5.59</td>
</tr>
</tbody>
</table>
5.1 BOOL_TO_INT(_E) / Bit data → word [signed] data conversion

Outline
This function converts bit data into word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_INT</td>
<td>Structured ladder/FBD: M0 → BOOL_TO_INT(_BOOL) → D0</td>
</tr>
<tr>
<td>BOOL_TO_INT_E</td>
<td>ST: BOOL_TO_INT(EN,_BOOL); Example: D0:= BOOL_TO_INT(M0);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_BOOL</td>
<td>Conversion source bit data</td>
<td>Bit</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (D)</td>
<td>Word [signed] data after conversion</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts bit data stored in a device specified in ( ) into word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

When the input value is "FALSE", this function outputs "0" as the word [signed] data value.
When the input value is "TRUE", this function outputs "1" as the word [signed] data value.

!”

Cautions
Use the function having ".E" in its name to connect a bus.
Program example
In this program, bit data stored in a device specified in \( s \) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(BOOL_TO_INT)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool1}} \quad \text{BOOL_TO_INT} \quad g_{\text{int1}} \\
g_{\text{bool1}} \quad \text{BOOL} \quad g_{\text{int1}}
\end{array}
\]

[ST]

\[
g_{\text{int1}} := \text{BOOL_TO_INT}(g_{\text{bool1}});
\]

2) Function with EN/ENO(BOOL_TO_INT_E)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool1}} \quad \text{BOOL_TO_INT_E} \quad g_{\text{bool3}} \\
g_{\text{bool2}} \quad \text{BOOL} \quad g_{\text{int1}}
\end{array}
\]

[ST]

\[
g_{\text{bool3}} := \text{BOOL_TO_INT}_E(g_{\text{bool1}}, g_{\text{bool2}}, g_{\text{int1}});
\]
5.2 BOOL_TO_DINT(_E) / Bit data → double word [signed] data conversion

Outline
This function converts bit data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_DINT</td>
<td>M0  BOOL_TO_DINT(Bool) *1 Label</td>
<td>BOOL_TO_DINT(BOOL); Example: Label := BOOL_TO_DINT(M0);</td>
</tr>
<tr>
<td>BOOL_TO_DINT_E</td>
<td>X000 BOOL_TO_DINT_E(EN, _BOOL, Output_label); Example: BOOL_TO_DINT_E(X000,M0, Label);</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_BOOL ( )</td>
<td>Conversion source bit data</td>
<td>Bit</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Double word [signed] data after conversion</td>
<td>Double Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts bit data stored in a device specified in ( ) into double word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, bit data stored in a device specified in [\textcolor{red}{\textbf{g\_bool1}}] is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in [\textcolor{red}{\textbf{g\_dint1}}].

1) Function without EN/ENO(BOOL_TO_DINT)

\[ \text{[Structured ladder/FBD]} \quad [\text{ST}] \]

\[ g\_dint1 := \text{BOOL\_TO\_DINT}(g\_bool1); \]

2) Function with EN/ENO(BOOL_TO_DINT_E)

\[ \text{[Structured ladder/FBD]} \quad [\text{ST}] \]

\[ g\_bool3 := \text{BOOL\_TO\_DINT\_E}(g\_bool1, g\_bool2, g\_dint1); \]
5.3 BOOL_TO_STR(_E) / Bit data → string data conversion

Outline

This function converts bit data into string data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_STR</td>
<td>M0 → BOOL_TO_STR(_BOOL) → Label</td>
<td>BOOL_TO_STR(_BOOL); Example: Label = BOOL_TO_STR(M0);</td>
<td></td>
</tr>
<tr>
<td>BOOL_TO_STR_E</td>
<td>X000 → BOOL_TO_STR_E(EN, _BOOL, Output_label) → Label</td>
<td>BOOL_TO_STR_E(EN, _BOOL, Output_label); Example: BOOL_TO_STR_E(X000, M0, Label);</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td>_BOOL</td>
<td>Conversion source bit data</td>
<td>Bit</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td>*1</td>
<td>String data after conversion</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data input to a device specified in (■) into string data, and outputs the data obtained by conversion to a device specified in (□).

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data. Use global labels when specifying labels.
Program example
In this program, bit data stored in a device specified in (g) is converted into string data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BOOL_TO_STR)

[Structured ladder/FBD]  [ST]
  g_bool1  <--- BOOL_TO_STR  g_string1 := BOOL_TO_STR(g_bool1);

2) Function with EN/ENO(BOOL_TO_STR_E)

[Structured ladder/FBD]  [ST]
  g_bool1  <--- BOOL_TO_STR_E  EN  g_bool3 := BOOL_TO_STR_E(g_bool1, g_bool2, g_string1);
5.4 BOOLE_TO_WORD(_E) / Bit data → word [unsigned]/bit string [16-bit] data conversion

Outline
This function converts bit data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
</table>
| BOOLE_TO_WORD | M0  
| BOOLE_TO_WORD |  
| BOOLE_TO_WORD_E | X000  
| BOOLE_TO_WORD_E |  

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Input variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_BOOL ( )</td>
<td>Conversion source bit data</td>
<td>Bit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0</td>
<td>Word [unsigned]/bit string [16-bit] data after conversion</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts bit data stored in a device specified in ( ) into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in ( ).

When the input value is "FALSE", this function outputs "0H" as the word [unsigned]/bit string [16-bit] data value.

When the input value is "TRUE", this function outputs "1H" as the word [unsigned]/bit string [16-bit] data value.

Cautions
Use the function having "_E" in its name to connect a bus.
Program example

In this program, bit data stored in a device specified in (s) is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BOOL_TO_WORD)

   [Structured ladder/FBD] [ST]
   g_bool1 BOOL_TO_WORD g_word1 := BOOL_TO_WORD(g_bool1);

2) Function with EN/ENO(BOOL_TO_WORD_E)

   [Structured ladder/FBD] [ST]
   g_bool1 BOOL_TO_WORD_E _BOOL g_word1
   g_bool2 BOOL EN ENO g_bool3
   g_bool3 := BOOL_TO_WORD_E(g_bool1, g_bool2, g_word1);
5.5 **BOOL_TO_DWORD(_E)** / Bit data → double word [unsigned]/bit string [32-bit] data conversion

Outline

This function converts bit data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BOOL_TO_DWORD</strong></td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td><strong>M0</strong></td>
<td>BOOL_TO_DWORD(_E)</td>
</tr>
<tr>
<td><strong>Output variable</strong></td>
<td><strong>ENO</strong></td>
</tr>
<tr>
<td><strong>M0</strong></td>
<td>_BOOL ( ) Conversion source bit data</td>
</tr>
</tbody>
</table>
| **Label** | BOOL_TO_DWORD_E(EN, _BOOL, Output_label); | Example:
| **EN** | Label= BOOL_TO_DWORD(M0); |
| **ENO** | M0 | | |
| **X000** | Label | BOOL_TO_DWORD_E(X000, M0, Label); |

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input variable</strong></td>
<td><strong>EN</strong> Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>_BOOL ( )</strong></td>
<td>Conversion source bit data</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>Output variable</strong></td>
<td><strong>ENO</strong> Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*<strong>1 ( )</strong></td>
<td>Double word [unsigned]/bit string [32-bit] data after conversion</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data stored in a device specified in ( ) into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in ( ).

When the input value is "FALSE", this function outputs "0H" as the double word [unsigned]/bit string [32-bit] data value.

When the input value is "TRUE", this function outputs "1H" as the double word [unsigned]/bit string [32-bit] data value.

<table>
<thead>
<tr>
<th>Bit data</th>
<th>Double Word [unsigned]/Bit string [32-bit] data</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>0H</td>
</tr>
<tr>
<td>TRUE</td>
<td>1H</td>
</tr>
</tbody>
</table>

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.
Program example

In this program, bit data stored in a device specified in \( g \) is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in \( h \).

1) Function without EN/ENO(BOOL_TO_DWORD)

[Structured ladder/FBD]

\[
g_{\text{bool1}} \rightarrow \text{BOOL_TO_DWORD} \rightarrow g_{\text{dword1}}
\]

[ST]

\[
g_{\text{dword1}} := \text{BOOL_TO_DWORD}(g_{\text{bool1}});
\]

2) Function with EN/ENO(BOOL_TO_DWORD_E)

[Structured ladder/FBD]

\[
g_{\text{bool1}} \rightarrow g_{\text{bool2}} \rightarrow \text{BOOL_TO_DWORD_E} \rightarrow g_{\text{bool3}} \rightarrow g_{\text{dword1}}
\]

[ST]

\[
g_{\text{bool3}} := \text{BOOL_TO_DWORD_E}(g_{\text{bool1}}, g_{\text{bool2}}, g_{\text{dword1}});
\]
5.6 BOOL_TO_TIME(_E) / Bit data → time data conversion

Outline
This function converts bit data into time data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td>BOOL_TO_TIME</td>
<td>M0 BOOL_TO_TIME(_BOOL)</td>
</tr>
<tr>
<td></td>
<td>Label</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOL_TO_TIME(_E)</td>
<td>X000 BOOL_TO_TIME_E(EN,_BOOL, Output_label)</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>variable</td>
<td>Conversion source bit data</td>
<td>Bit</td>
</tr>
<tr>
<td>Output</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>variable</td>
<td>Time data after conversion</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts bit data stored in a device specified in ( ) into time data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
Program example

In this program, bit data stored in a device specified in \( g \) is converted into time data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(BOOL_TO_TIME)

\[
\begin{align*}
g\_bool1 & \quad \text{BOOL_TO_TIME} \\
\text{g\_time1} & \quad \text{BOOL}
\end{align*}
\]

[Structured ladder/FBD] [ST]

\[ g\_time1 := \text{BOOL\_TO\_TIME}(g\_bool1); \]

2) Function with EN/ENO(BOOL_TO_TIME_E)

\[
\begin{align*}
g\_bool1 & \quad \text{BOOL_TO_TIME\_E} \\
g\_bool2 & \quad \text{EN} \\
g\_bool3 & \quad \text{ENO} \\
\text{g\_time1} & \quad \text{BOOL}
\end{align*}
\]

[Structured ladder/FBD] [ST]

\[ g\_bool3 := \text{BOOL\_TO\_TIME\_E}(g\_bool1, g\_bool2, g\_time1); \]
5.7 INT_TO_DINT(_E) / Word [signed] data → double word [signed] data conversion

Outline
This function converts word [signed] data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_DINT</td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td></td>
<td>D0 [INT] [] []^1 Label []</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>ST</td>
</tr>
<tr>
<td></td>
<td>ENO [] [] []^1 Label []</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_INT [] ([])</td>
<td>Conversion source word [signed] data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>[]^1 ([])</td>
<td>Double word [signed] data after conversion</td>
<td>Double Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [signed] data stored in a device specified in \[\] into double word [signed] data, and outputs the data obtained by conversion to a device specified in \[\].

1234 → 1234

Word [signed] data Double word [signed] data

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, word [signed] data stored in a device specified in \( g_{\text{int1}} \) is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in \( g_{\text{dint1}} \).

1) Function without EN/ENO(INT_TO_DINT)

[Structured ladder/FBD]

\[ g_{\text{int1}} = 5923 \]

\[ g_{\text{dint1}} \]

\[ g_{\text{int1}} = 5923 \]

[ST]

\[ g_{\text{dint1}} := \text{INT\_TO\_DINT}(g_{\text{int1}}) \]

2) Function with EN/ENO(INT_TO_DINT_E)

[Structured ladder/FBD]

\[ g_{\text{bool1}} \]

\[ g_{\text{int1}} \]

\[ g_{\text{bool1}} \]

\[ g_{\text{bool3}} \]

\[ g_{\text{int1}} \]

\[ g_{\text{dint1}} \]

[ST]

\[ g_{\text{bool3}} := \text{INT\_TO\_DINT\_E}(g_{\text{bool1}}, g_{\text{int1}}, g_{\text{dint1}}) \]
5.8 DINT_TO_INT(_E) / Double word [signed] data → word [signed] data conversion

Outline

This function converts double word [signed] data into word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DINT_TO_INT</strong></td>
<td>Label → DINT_TO_INT(_DINT); DINT_TO_INT(_DINT); Example: D10 = DINT_TO_INT(Label);</td>
</tr>
<tr>
<td><strong>DINT_TO_INT_E</strong></td>
<td>X000 → DINT_TO_INT_E(EN,_DINT, Output_label); DINT_TO_INT_E(EN,_DINT, Output_label); Example: DINT_TO_INT_E(X000, Label, D10);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td>DINT</td>
<td>Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>*1</td>
<td>Word [signed] data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in into word [signed] data, and outputs the data obtained by conversion to a device specified in .

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, double word [signed] data stored in a device specified in (s) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(DINT_TO_INT)

[Structured ladder/FBD]

```
g_dint1 = 5923
```

```
DINT_TO_INT
```

```
g_int1 = DINT_TO_INT(g_dint1);
```

2) Function with EN/ENO(DINT_TO_INT_E)

[Structured ladder/FBD]

```
g_bool1
      g_dint1
DINT_TO_INT_E
      g Bool3
```

```
[ST]
```

```
g_bool3 := DINT_TO_INT_E(g_bool1, g_dint1, g_int1);
```

```
5.9 INT_TO_BOOL(_E) / Word [signed] data → bit data conversion

Outline
This function converts word [signed] data into bit data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_BOOL</td>
<td>D0</td>
</tr>
<tr>
<td>INT_TO_BOOL_E</td>
<td>X000</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td>_INT ( )</td>
<td>Conversion source word [signed] data</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>*1 ( )</td>
<td>Bit data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [signed] data stored in a device specified in ( ) into bit data, and outputs the data obtained by conversion to a device specified in ( ).
When the input value is "0", this function outputs "FALSE". When the input value is any value other than "0", this function outputs "TRUE".

<table>
<thead>
<tr>
<th>Word [signed] data</th>
<th>Bit data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>1567</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Cautions
Use the function having "_E" in its name to connect a bus.
Program example

In this program, word [signed] data stored in a device specified in (s) is converted into bit data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(INT_TO_BOOL)

[Structured ladder/FBD]

```
g_int1=5923
```

[ST]

```
g_bool1 := INT_TO_BOOL(g_int1);
```

2) Function with EN/ENO(INT_TO_BOOL_E)

[Structured ladder/FBD]

```
g_bool1
```

[ST]

```
g_bool3 := INT_TO_BOOL_E(g_bool1, g_int1, g_bool2);
```
5.10 DINT_TO_BOOL(_E) / Double word [signed] data → bit data conversion

Outline

This function converts double word [signed] data into bit data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT_TO_BOOL</td>
<td>Label DINT_TO_BOOL DINT *1 M0</td>
<td>DINT_TO_BOOL(DINT);</td>
</tr>
<tr>
<td>DINT_TO_BOOL E</td>
<td>X000 DINT_TO_BOOL_E EN ENO *1 M0</td>
<td>DINT_TO_BOOL_E(EN, DINT, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_DINT</td>
<td>Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Bit data after conversion</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in ( ) into bit data, and outputs the data obtained by conversion to a device specified in ( )

When the input value is "0", this function outputs "FALSE".

When the input value is any value other than "0", this function outputs "TRUE".

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.
Program example

In this program, double word [signed] data stored in a device specified in $\text{D}$ is converted into bit data, and the data obtained by conversion is output to a device specified in $\text{E}$.

1) Function without EN/ENO (DINT_TO_BOOL)

[Structured ladder/FBD]

```
g_dint1=0
```

[DINT]

```
g_bool1 := DINT_TO_BOOL(g_dint1);
```

2) Function with EN/ENO (DINT_TO_BOOL_E)

[Structured ladder/FBD]

```
g_bool1
```

[DINT]

```
g_dint1
```

[DINT]

```
g_bool2
```

[DINT]

```
g_bool3
```

[ST]

```
g_bool3 := DINT_TO_BOOL_E(g_bool1, g_dint1, g_bool2);
```
5.11 INT_TO_REAL(E) / Word [signed] data → float (single precision) data conversion

Outline
This function converts word [signed] data into float (single precision) data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td>INT_TO_REAL</td>
<td>D0 → INT_TO_REAL(a_Int) &quot;1&quot;</td>
</tr>
<tr>
<td>INT_TO_REAL_E</td>
<td>X000 → INT_TO_REAL_E(EN,a_Int,</td>
</tr>
<tr>
<td></td>
<td>ENO,D0) &quot;1&quot;</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>a_Int</td>
<td>Conversion source word [signed] data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Float (single precision) data after conversion</td>
<td>FLOAT (Single Precision)</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [signed] data stored in a device specified in \( \text{D0} \) into float (single precision) data, and outputs the data obtained by conversion to a device specified in \( \text{D0} \).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.
4) The number of significant figures of FLOAT (Single Precision) data is approximately 7 since the data is processed in 32-bit single precision. Accordingly, the converted data includes an error (rounding error) if an integer value is outside the range of -16777216 to 16777215.
**Program example**

In this program, word [signed] data stored in a device specified in (a) is converted into float (single precision) data, and the data obtained by conversion is output to a device specified in (b).

1) Function without EN/ENO(INT_TO_REAL)

```
[Structured ladder/FBD] [ST]
g_int1=5923
a_int
INT_TO_REAL
__g_real1 := INT_TO_REAL(g_int1);
```

2) Function with EN/ENO(INT_TO_REAL_E)

```
[Structured ladder/FBD] [ST]
g_bool1
a_int
INT_TO_REAL_E
__g_bool3 := INT_TO_REAL_E(g_bool1, g_int1, g_real1);
```
5.12  DINT_TO_REAL(_E) / Double word [signed] data → float (single precision) data conversion

Outline
This function converts double word [signed] data into float (single precision) data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT_TO_REAL</td>
<td>DINT_TO_REAL(a_Dint);</td>
</tr>
<tr>
<td>DINT_TO_REAL_E</td>
<td>DINT_TO_REAL_E(EN,a_Dint,EN0);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>a_Dint</td>
<td>Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Float (single precision) data after conversion</td>
<td>FLOAT (Single Precision)</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts double word [signed] data stored in a device specified in  into float (single precision) data, and outputs the data obtained by conversion to a device specified in .

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.
Program example

In this program, double word [signed] data stored in a device specified in is converted into float (single precision) data, and the data obtained by conversion is output to a device specified in .

1) Function without EN/ENO(DINT_TO_REAL)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{dint1}}=65000.0 \\
g_{\text{real1}}=65000.0 \\
g_{\text{real1}} := \text{DINT\_TO\_REAL}(g_{\text{dint1}});
\end{array}
\]

2) Function with EN/ENO(DINT\_TO\_REAL\_E)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool1}} \\
g_{\text{dint1}} \\
g_{\text{real1}} := \text{DINT\_TO\_REAL\_E}(g_{\text{bool1}}, g_{\text{dint1}}, g_{\text{real1}});
\end{array}
\]
5.13 INT_TO_STR(_E) / Word [signed] data → string data conversion

Outline
This function converts word [signed] data into string data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_STR</td>
<td>D0 _INT *1 Label</td>
<td>INT_TO_STR(INT);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Label:=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INT_TO_STR(D0);</td>
<td></td>
</tr>
<tr>
<td>INT_TO_STR_E</td>
<td>X000 EN ENO *1 Label</td>
<td>INT_TO_STR_E(EN,INT,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output_label);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INT_TO_STR_E(X000,D0,Label);</td>
<td></td>
</tr>
</tbody>
</table>

\*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>EN</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>_INT ( )</td>
<td>Conversion source word [signed] data</td>
</tr>
<tr>
<td>Output</td>
<td>ENO</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>*1 ( )</td>
<td>String data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function converts word [signed] data stored in a device specified in ( ) into string data, and outputs the data obtained by conversion to a device specified in ( ).

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>Sign data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code for ten-thousands place</td>
<td>ASCII code for hundreds place</td>
<td>ASCII code for thousands place</td>
</tr>
<tr>
<td>ASCII code for ones place</td>
<td>ASCII code for tens place</td>
<td>0000H</td>
</tr>
</tbody>
</table>

Automatically stored at the end of the character string

2) In "Sign data", "20H (space)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

3) "20H (space)" is stored in high-order digits when the number of significant figures is small.
   Example: When "-123" is input

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>Sign data</th>
</tr>
</thead>
<tbody>
<tr>
<td>20H (space)</td>
<td>2DH (-)</td>
<td></td>
</tr>
<tr>
<td>31H (1)</td>
<td>20H (space)</td>
<td></td>
</tr>
<tr>
<td>33H (3)</td>
<td>32H (2)</td>
<td></td>
</tr>
<tr>
<td>0000H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4) "00H" is automatically stored at the end (4th word) of the character string.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data. Use global labels when specifying labels.
Error

An operation error occurs in the following case. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the number of points occupied by the string data storage destination (device specified in ) exceeds the range of the corresponding device
   (Error code: K6706)

Program example

In this program, word [signed] data stored in a device specified in is converted into string data, and the data obtained by conversion is output to a device specified in .

1) Function without EN/ENO(INT_TO_STR)

   [Structured ladder/FBD] [ST]
   \[ \begin{array}{c}
   | g\_int1=\_INT \hspace{1cm} g\_string:=\_INT\_TO\_STR(g\_int1); \\
   \end{array} \]

2) Function with EN/ENO(INT_TO_STR_E)

   [Structured ladder/FBD] [ST]
   \[ \begin{array}{c}
   | g\_bool1 \hspace{1cm} INT\_TO\_STR\_E \hspace{1cm} g\_bool3 := INT\_TO\_STR\_E(g\_bool1, g\_int1, g\_string1); \\
   | g\_int1 \hspace{1cm} g\_bool3 \hspace{1cm} g\_string1 \\
   \end{array} \]
5.14 DINT_TO_STR(\_E) / Double word [signed] data → string data conversion

Outline
This function converts double word [signed] data into string data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td>DINT_TO_STR</td>
<td>Label1 DINT_TO_STR(_DINT);</td>
</tr>
<tr>
<td></td>
<td>Example: Label2= DINT_TO_STR(_DINT);</td>
</tr>
<tr>
<td>DINT_TO_STR_E</td>
<td>X000 DINT_TO_STR_E(EN,_DINT,</td>
</tr>
<tr>
<td></td>
<td>Label1, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_DINT (_DINT)</td>
<td>Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (_DINT)</td>
<td>String data after conversion</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function converts double word [signed] data stored in a device specified in ( ) into string data, and outputs the data obtained by conversion to a device specified in ( ).

2) In "Sign data", "20H (space)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

3) "20H (space)" is stored in high-order digits when the number of significant figures is small. Example: When ",-123456" is input

4) "00H" is automatically stored at the end (high-order byte of the 6th word) of the character string.
Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

Error

An operation error occurs in the following case. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the number of points occupied by the string data storage destination (device specified in ) exceeds the range of the corresponding device (Error code: K6706)

Program example

In this program, double word [signed] data stored in a device specified in  is converted into string data, and the data obtained by conversion is output to a device specified in .

1) Function without EN/ENO(DINT_TO_STR)

[Structured ladder/FBD]

\[
g_{\text{dint}} := -12345678; \quad \text{DINT} \rightarrow \text{STR}
\]

[ST]

\[g_{\text{string}} := \text{DINT} \rightarrow \text{STR}(g_{\text{dint}});\]

2) Function with EN/ENO(DINT_TO_STR_E)

[Structured ladder/FBD]

\[
g_{\text{bool1}} \quad \text{DINT} \rightarrow \text{STR}_E \quad \text{DINT} \quad \text{EN/ENO} \quad g_{\text{bool3}} \quad g_{\text{string}}
\]

[ST]

\[g_{\text{bool3}} := \text{DINT} \rightarrow \text{STR}_E(g_{\text{bool1}}, g_{\text{dint}}, g_{\text{string}});\]
5.15 INT_TO_WORD(_E) / Word [signed] data → word [unsigned]/bit string [16-bit] data conversion

Outline
This function converts word [signed] data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_WORD</td>
<td>D0 → INT_TO_WORD(INT) *1 → D10</td>
<td>INT_TO_WORD(_INT); Example: D10:= INT_TO_WORD(D0);</td>
<td></td>
</tr>
<tr>
<td>INT_TO_WORD_E</td>
<td>X000 → INT_TO_WORD_E(EN, _INT, Output_label); Example: INT_TO_WORD_E(X000,D0,D10);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_INT</td>
<td>Conversion source word [signed] data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word [unsigned]/Bit String [16-bit] data after conversion</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [signed] data stored in a device specified in ( ) into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
Use the function having "_E" in its name to connect a bus.
Program example
In this program, word [signed] data stored in a device specified in \( s \) is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(INT_TO_WORD)

[Structured ladder/FBD]
\[
g_{\text{int1}}=5923 \quad \text{INT_TO_WORD} \quad g_{\text{word1}}=16\#1723
\]

[ST]
\[
g_{\text{word1}} := \text{INT_TO_WORD}(g_{\text{int1}});
\]

2) Function with EN/ENO(INT_TO_WORD_E)

[Structured ladder/FBD]
\[
g_{\text{bool1}} \quad \text{INT_TO_WORD_E} \quad g_{\text{bool3}} \quad g_{\text{int1}} \quad \text{EN} \quad \text{ENO} \quad g_{\text{word1}}
\]

[ST]
\[
g_{\text{bool3}} := \text{INT_TO_WORD_E}(g_{\text{bool1}}, g_{\text{int1}}, g_{\text{word1}});
\]
5.16 DINT_TO_WORD(_E) / Double word [signed] data → word [unsigned]/bit string [16-bit] data conversion

Outline
This function converts double word [signed] data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT_TO_WORD</td>
<td>Label DINT_TO_WORD_DINT &quot;I&quot; D10</td>
</tr>
<tr>
<td>DINT_TO_WORD_E</td>
<td>X000 DINT_TO_WORD_E EN ENO DINT &quot;I&quot; D10</td>
</tr>
</tbody>
</table>

*1 Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td>DINT (＜＞)</td>
<td>Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td>&quot;I&quot; (＜＞)</td>
<td>Word [unsigned]/bit string [16-bit] data after conversion</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts double word [signed] data stored in a device specified in (＜＞) into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in (＜＞).

```
12345678 ➔ 614EH
```

Double word [signed] data ➔ Word [unsigned]/bit string [16-bit] data

12345678 ➔ 614E

The information stored in high-order 16 bits is discarded.

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, double word [signed] data stored in a device specified in  is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in .

1) Function without EN/ENO(DINT_TO_WORD)

[Structured ladder/FBD]

```
g_dint1 = 12345678
DINT_TO_WORD
DINT
```

[ST]

```
g_word1 := DINT_TO_WORD(g_dint1);
```

2) Function with EN/ENO(DINT_TO_WORD_E)

[Structured ladder/FBD]

```
g_bool1
DINT_TO_WORD_E
EN
DINT
ENO
```

[ST]

```
g_bool3 := DINT_TO_WORD_E(g_bool1, g_dint1, g_word1);
```
5.17 INT_TO_DWORD(_E) / word [signed] data → double word [unsigned]/bit string [32-bit] data conversion

Outline
This function converts word [signed] data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_DWORD</td>
<td>D0 INT_TO_DWORD _INT *1 Label</td>
<td>INT_TO_DWORD(_INT); Example: Label:= INT_TO_DWORD(D0);</td>
</tr>
<tr>
<td>INT_TO_DWORD _E</td>
<td>X000 INT_TO_DWORD_E EN ENO D0 _INT *1 Label</td>
<td>INT_TO_DWORD_E(EN, INT, Output_label); Example: INT_TO_DWORD_E(X000, D0, Label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>- _INT ( )</td>
<td>Conversion source word [signed] data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Double word [unsigned]/bit string [32-bit] data after conversion</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [signed] data stored in a device specified in ( ) into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in ( ).

![Data conversion example]

Each of high-order 16 bits becomes "0" after data conversion.

Cautions
1) Use the function having ".E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Program example

In this program, word [signed] data stored in a device specified in \( \text{a} \) is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in \( \text{d} \).

1) Function without EN/ENO(INT_TO_DWORD)

```
[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{int1}} = 10 \\
\text{INT_TO_DWORD} \\
\text{INT} \\
g_{\text{dword1}} = 16\#0000000A \\
\end{array}
\]

[ST]
\( g_{\text{dword1}} := \text{INT_TO_DWORD}(g_{\text{int1}}); \)
```

2) Function with EN/ENO(INT_TO_DWORD_E)

```
[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool1}} \\
g_{\text{int1}} \\
\text{INT_TO_DWORD_E} \\
\text{EN} \\
\text{ENO} \\
g_{\text{bool3}} \\
g_{\text{dword1}} \\
\end{array}
\]

[ST]
\( g_{\text{bool3}} := \text{INT_TO_DWORD_E}(g_{\text{bool1}}, g_{\text{int1}}, g_{\text{dword1}}); \)
5.18 DINT_TO_DWORD(_E) / Double word [signed] data → double word [unsigned]/bit string [32-bit] data conversion

Outline
This function converts double word [signed] data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT_TO_DWORD</td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>DINT_TO_DWORD(_E)</td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_DINT ( )</td>
<td>Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Double word [unsigned]/bit string [32-bit] data after conversion</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts double word [signed] data stored in a device specified in ( ) into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
1) Use the function having " _E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, double word [signed] data stored in a device specified in  is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in .

1) Function without EN/ENO(DINT_TO_DWORD)

[Structured ladder/FBD]

\[
\begin{align*}
g\_\text{dint1} &= 74565 \\
g\_\text{dword1} &= 16\#00012345
\end{align*}
\]

[ST]
\[g\_\text{dword1} := \text{DINT\_TO\_DWORD}(g\_\text{dint1});\]

2) Function with EN/ENO(DINT\_TO\_DWORD\_E)

[Structured ladder/FBD]

\[
\begin{align*}
g\_\text{bool1} &\quad \text{DINT\_TO\_DWORD\_E} \\
g\_\text{dint1} &\quad \text{DINT} \\
g\_\text{bool3} &\quad \text{ENO} \\
g\_\text{dword1} &\quad \text{EN}
\end{align*}
\]

[ST]
\[g\_\text{bool3} := \text{DINT\_TO\_DWORD\_E}(g\_\text{bool1}, g\_\text{dint1}, g\_\text{dword1});\]
5.19 INT_TO_BCD(_E) / Word [signed] data → BCD data conversion

Outline
This function converts word [signed] data into BCD data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_BCD</td>
<td>D0 → INT_TO_BCD(_INT) *1</td>
<td>D10</td>
<td></td>
</tr>
<tr>
<td>INT_TO_BCD_E</td>
<td>X000 → INT_TO_BCD_E(EN, _INT, Output_label)</td>
<td>D0 → ENO *1</td>
<td>D10</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN ( )</td>
<td>Execution condition</td>
</tr>
<tr>
<td>_INT ( )</td>
<td>Conversion source word [signed] data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO ( )</td>
<td>Execution status</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>BCD data after conversion</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [signed] data stored in a device specified in ( ) into BCD data, and outputs the data obtained by conversion to a device specified in ( ).

<table>
<thead>
<tr>
<th>9999</th>
<th>9999H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010 0110 1100 0000 1111 1111</td>
<td>0010 1100 0000 1111 1111</td>
</tr>
</tbody>
</table>

Thousands place Hundreds place Tens place Ones place

Cautions
Use the function having "_E" in its name to connect a bus.

Error
An operation error occurs when the value stored in a device specified in ( ) is outside the range from "0" to "9,999".
Program example

In this program, word [signed] data stored in a device specified in \( \text{s} \) is converted into BCD data, and the data obtained by conversion is output to a device specified in \( \text{d} \).

1) Function without EN/ENO(INT_TO_BCD)

   [Structured ladder/FBD]

   \[
   \begin{align*}
   &\text{g\_int1}=5923 \\
   &\text{g\_word}=16\#5923
   \end{align*}
   \]

   [ST]
   
   \[
   \text{g\_word1} := \text{INT\_TO\_BCD}(\text{g\_int1});
   \]

2) Function with EN/ENO(INT_TO_BCD_E)

   [Structured ladder/FBD]

   \[
   \begin{align*}
   &\text{g\_bool1} \\
   &\text{g\_int1} \\
   &\text{g\_bool3} \\
   &\text{g\_word1}
   \end{align*}
   \]

   [ST]
   
   \[
   \text{g\_bool3} := \text{INT\_TO\_BCD\_E}(\text{g\_bool1}, \text{g\_int1}, \text{g\_word1});
   \]
5.20 DINT_TO_BCD_(E) / Double word [signed] data → BCD data conversion

Outline
This function converts double word [signed] data into BCD data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT_TO_BCD</td>
<td>Structured ladder/FBD: Label1 → DINT_TO_BCD(DINT) → *1 → Label2</td>
</tr>
<tr>
<td></td>
<td>EX: DINT_TO_BCD(DINT); Label2 := DINT_TO_BCD(Label1);</td>
</tr>
<tr>
<td>DINT_TO_BCD_E</td>
<td>X000 → DINT_TO_BCD_E(EN,DINT, Output_label) → *1 → Label2</td>
</tr>
<tr>
<td></td>
<td>EX: DINT_TO_BCD_E(X000, Label1, Label2);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>_DINT (__) Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>*1 (__) BCD data after conversion</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts double word [signed] data stored in a device specified in ( ) into BCD data, and outputs the data obtained by conversion to a device specified in ( ).

Example:

99999999 → 99999999H

Make sure to set them to "0".

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
Error
An operation error occurs when the value stored in a device specified in \(s\) is outside the range from "0" to "99,999,999".

Program example
In this program, double word [signed] data stored in a device specified in \(s\) is converted into BCD data, and the data obtained by conversion is output to a device specified in \(d\).

1) Function without EN/ENO(DINT_TO_BCD)

[Structured ladder/FBD]
\[
g\text{_dint1} = 20000 \quad \text{DINT} \quad \text{DINT\_TO\_BCD} \quad g\text{\_dword1} = 16\#00020000
\]

[ST]
g\text{\_dword1} := \text{DINT\_TO\_BCD}(g\text{\_dint1});

2) Function with EN/ENO(DINT_TO_BCD_E)

[Structured ladder/FBD]
\[
g\text{\_bool1}\quad \text{DINT\_TO\_BCD\_E} \quad \text{EN} \quad \text{ENO}\quad g\text{\_bool3}\quad \text{DINT}\quad g\text{\_dword1}
\]

[ST]
g\text{\_bool3} := \text{DINT\_TO\_BCD\_E}(g\text{\_bool1}, g\text{\_dint1}, g\text{\_dword1});
5.21 INT_TO_TIME(_E) / Word [signed] data → time data conversion

Outline

This function converts word [signed] data into time data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_TIME</td>
<td>D0 INT_TO_TIME(_INT) Label</td>
</tr>
<tr>
<td>INT_TO_TIME_E</td>
<td>X000 INT_TO_TIME_E(EN,_INT, Output_label) Label</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in ( ) into time data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, word [signed] data stored in a device specified in (s) is converted into time data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(INT_TO_TIME)

[Structured ladder/FBD]

```
g_int1  INT_TO_TIME  g_time1
```

[ST]

```
g_time1 := INT_TO_TIME(g_int1);
```

2) Function with EN/ENO(INT_TO_TIME_E)

[Structured ladder/FBD]

```
g_bool1  INT_TO_TIME_E  g_bool3
        EN    ENO
```

[ST]

```
g_bool3 := INT_TO_TIME_E(g_bool1, g_int1, g_time1);
```
5.22 DINT_TO_TIME(_E) / Double word [signed] data \(\rightarrow\) time data conversion

Outline

This function converts double word [signed] data into time data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT_TO_TIME</td>
<td>(\text{DINT} \rightarrow \text{Time})</td>
<td>(\text{DINT} \rightarrow \text{Time})</td>
</tr>
<tr>
<td>DINT_TO_TIME_E</td>
<td>(\text{DINT} \rightarrow \text{Time})</td>
<td>(\text{DINT} \rightarrow \text{Time})</td>
</tr>
</tbody>
</table>

\*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_DINT</td>
<td>Conversion source double word [signed] data</td>
<td>Double Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Time data after conversion</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in \(\text{s}\) into time data, and outputs the data obtained by conversion to a device specified in \(\text{d}\).

\[7FFFFFFF\text{h} \rightarrow 24d20h31m23s647ms\]

Double word [signed] data \(\rightarrow\) Time data

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, double word [signed] data stored in a device specified in is converted into time data, and the data obtained by conversion is output to a device specified in .

1) Function without EN/ENO(DINT_TO_TIME)

2) Function with EN/ENO(DINT_TO_TIME_E)
5.23 REAL_TO_INT(E) / Float (single precision) data → word [signed] data conversion

Outline
This function converts float (single precision) data into word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL_TO_INT</td>
<td>Label → REAL_TO_INT(a_real) → D10</td>
<td>REAL_TO_INT(a_real); Example: D10:= REAL_TO_INT(Label);</td>
</tr>
<tr>
<td>REAL_TO_INT_E</td>
<td>X000 → REAL_TO_INT_E(EN,a_real,Output_label) → D10</td>
<td>REAL_TO_INT_E(EN,a_real,Output_label); Example: REAL_TO_INT_E(X000, Label, D10);</td>
</tr>
</tbody>
</table>

*1 Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td>a_real ( )</td>
<td>Conversion source float (single precision) data</td>
<td>FLOAT (Single Precision)</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td>*1</td>
<td>Word [signed] data after conversion</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts float (single precision) data stored in a device specified in into word [signed] data, and outputs the data obtained by conversion to a device specified in .

![Diagram](image)

The portion after the decimal point is rounded off.

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.
4) In the data obtained by conversion, the portion after the decimal point of the float (single precision) data (source data) is rounded off.
5) When the FLOAT (Single Precision) data to is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, float (single precision) data stored in a device specified in \( a \) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in \( b \).

1) Function without EN/ENO(REAL_TO_INT)

[Structured ladder/FBD]

\[
\begin{align*}
g_{\text{real}1} &= 5923.5 \\
\text{REAL_TO_INT} & \quad a_{\text{real}} \\
g_{\text{int}1} &= 5923
\end{align*}
\]

[ST]

\[g_{\text{int}1} := \text{REAL_TO_INT}(g_{\text{real}1});\]

2) Function with EN/ENO(REAL_TO_INT_E)

[Structured ladder/FBD]

\[
\begin{align*}
g_{\text{bool}1} & \\
g_{\text{real}1} & \quad \text{REAL_TO_INT_E} \\
\text{EN} & \quad a_{\text{real}} \\
\text{ENO} & \quad g_{\text{bool}3} \\
g_{\text{int}1} &
\end{align*}
\]

[ST]

\[g_{\text{bool}3} := \text{REAL_TO_INT_E}(g_{\text{bool}1}, g_{\text{real}1}, g_{\text{int}1});\]
5.24 REAL_TO_DINT(_E) / Float (single precision) data → double word [signed] data conversion

Outline

This function converts float (single precision) data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL_TO_DINT</td>
<td>Label1 REAL_TO_DINT(a_real) Label2</td>
</tr>
<tr>
<td>REAL_TO_DINT_E</td>
<td>X000 REAL_TO_DINT_E(EN,a_real, Output_label)</td>
</tr>
</tbody>
</table>

Example:

REAL_TO_DINT(a_real)

REAL_TO_DINT_E(X000, Label1, Label2);

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>a_real</td>
<td>Conversion source float (single precision) data</td>
<td>FLOAT (Single Precision)</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Double word [signed] data after conversion</td>
<td>Double Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts float (single precision) data stored in a device specified in into double word [signed] data, and outputs the data obtained by conversion to a device specified in.

\[
\text{FLOAT (single precision) data} \rightarrow \text{Double word [signed] data}
\]

The portion after the decimal point is rounded off.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.
4) In the data obtained by conversion, the portion after the decimal point of the float (single precision) data (source data) is rounded off.
5) When the FLOAT (Single Precision) data to is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, float (single precision) data stored in a device specified in \( \text{a} \) is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in \( \text{c} \).

1) Function without EN/ENO(REAL_TO_DINT)

[Structured ladder/FBD]

```
\( \text{REAL_TO_DINT} \)
```

\( \text{g\_real1} = 65000.5 \quad \text{g\_dint1} = 65000 \)

[ST]

\( \text{g\_dint1} := \text{REAL\_TO\_DINT}(\text{g\_real1}); \)

2) Function with EN/ENO(DINT_TO_TIME_E)

[Structured ladder/FBD]

```
\( \text{REAL\_TO\_DINT\_E} \)
```

\( \text{g\_bool1} \quad \text{g\_real1} \quad \text{g\_dint1} \quad \text{g\_bool3} \)

[ST]

\( \text{g\_bool3} := \text{REAL\_TO\_DINT\_E}(\text{g\_bool1}, \text{g\_real1}, \text{g\_dint1}); \)
5.25 REAL_TO_STR(_E) / Float (single precision) data → string data conversion

Outline
This function converts float (single precision) data into string data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL_TO_STR</td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td></td>
<td>ST</td>
</tr>
<tr>
<td></td>
<td>REAL_TO_STR(REAL);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Label2:= REAL_TO_STR(Label1);</td>
</tr>
<tr>
<td>REAL_TO_STR_E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REAL_TO_STR_E(EN_REAL,</td>
</tr>
<tr>
<td></td>
<td>Output_label);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>REAL_TO_STR_E(X000, Label1,</td>
</tr>
<tr>
<td></td>
<td>Label2);</td>
</tr>
</tbody>
</table>

*1 Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_REAL</td>
<td>Conversion source float (single precision) data</td>
<td>FLOAT (Single Precision)</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>String data after conversion</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function converts float (single precision) data stored in a device specified in ( ) into string (exponent) data, and outputs the data obtained by conversion to a device specified in ( ).

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>20H (space)</td>
<td>Sign data (integer part)</td>
<td>String 1st word</td>
</tr>
<tr>
<td>ASCII code (2EH) for decimal point (.)</td>
<td>ASCII code for integer part</td>
<td></td>
</tr>
<tr>
<td>ASCII code for 2nd decimal place</td>
<td>ASCII code for 1st decimal place</td>
<td></td>
</tr>
<tr>
<td>ASCII code for 4th decimal place</td>
<td>ASCII code for 3rd decimal place</td>
<td></td>
</tr>
<tr>
<td>45H(E)</td>
<td>ASCII code for 5th decimal place</td>
<td></td>
</tr>
<tr>
<td>ASCII code for tens place of exponent part</td>
<td>Sign data (exponent part)</td>
<td></td>
</tr>
<tr>
<td>00H</td>
<td>ASCII code for ones place of exponent part</td>
<td></td>
</tr>
</tbody>
</table>

Automatically stored at the end of the character string.
2) The string data obtained by conversion is output to a device specified in as follows:
   a) The number of digits is fixed respectively for the integer part, decimal part and exponent part as follows:
      Integer part: 1, decimal part: 5, exponent part: 2
      "2EH (.)" is automatically stored in the 3rd byte, and "45H (E)" is automatically stored in the 9th byte.

      \[
      \begin{array}{c}
      \text{Float (single precision) data} \\
      \text{Total number of digits (12 digits)} \\
      \text{Integer part (1 digit)} \\
      \text{Decimal part (5 digits)} \\
      \text{Exponent part (2 digits)} \\
      \end{array}
      \]

      \[
      \begin{array}{c}
      -12.3456 \\
      \end{array}
      \]

      \[
      \begin{array}{c}
      \text{Total number of digits (12 digits)} \\
      \text{Number of digits of decimal part (5)} \\
      \end{array}
      \]

      \[
      \begin{array}{c}
      \text{Total number of digits (12 digits)} \\
      \text{Number of digits of exponent part (2)} \\
      \end{array}
      \]

      \[
      \begin{array}{c}
      \text{Float (single precision) data} \\
      \end{array}
      \]

   b) In "Sign data (integer part)", "20H (space)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

   c) The 6th and later digits of the decimal part are rounded.

   d) "30H (0)" is stored in the decimal part when the number of significant figures is small.

   e) In "Sign data (exponent part)", "2BH (+)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

   f) "30H (0)" is stored in the tens place of the exponent part when the exponent part consists of 1 digit.

3) "00H" is automatically stored at the end (7th word) of the character string.

Cautions

1) Use the function having " _E" in its name to connect a bus.

2) When handling character string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.

   You can specify 32-bit counters directly, however, because they are 32-bit devices.

   Use global labels when specifying labels.

3) When the FLOAT (Single Precision) data to is set from the programming tool, a rounding error may be generated.

   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Error
An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the value stored in a device specified in $\text{D}$ is outside the following range:
   \[0, \pm 2^{-126} \leq \text{Value of device specified in } \text{D} \leq \pm 2^{128}\]
   (Error code: K6706)

2) When the range of a device which will store the character string obtained by conversion (device specified in $\text{D}$) exceeds the range of the corresponding device
   (Error code: K6706)

3) When the conversion result exceeds the specified total number of digits
   (Error code: K6706)

Program example
In this program, float (single precision) data stored in a device specified in $\text{D}$ is converted into string data, and the data obtained by conversion is output to a device specified in $\text{D}$.

1) Function without EN/ENO(REAL_TO_STR)

   \[
g_{\text{real1}}=12.34567 \quad \text{REAL_TO_STR} \quad \text{REAL} \quad \text{g_{string1}}=\text{"-1.23457E+01"}
   \]

   \[
   \text{[ST]} \\
   g_{\text{string1}} := \text{REAL_TO_STR}(g_{\text{real1}});
   \]

2) Function with EN/ENO(REAL_TO_STR_E)

   \[
g_{\text{bool1}} \quad \text{REAL_TO_STR_E} \quad \text{EN} \quad \text{ENO} \\
   g_{\text{real1}} \quad \text{REAL} \\
   g_{\text{bool1}}, g_{\text{real1}}, g_{\text{string1}} \quad \text{g_{bool3}}
   \]

   \[
   \text{[ST]} \\
   g_{\text{bool3}} := \text{REAL_TO_STR_E}(g_{\text{bool1}}, g_{\text{real1}}, g_{\text{string1}});
   \]
5.26 WORD_TO_BOOL(_E) / Word [unsigned] / bit string [16-bit] data → bit data conversion

Outline
This function converts word [unsigned]/bit string [16-bit] data into bit data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td>WORD_TO_BOOL</td>
<td>D0 WORD_TO_BOOL(WORD) *1</td>
</tr>
<tr>
<td>_E</td>
<td></td>
</tr>
<tr>
<td>WORD_TO_BOOL</td>
<td>X000 WORD_TO_BOOL_E(EN,</td>
</tr>
<tr>
<td>_E</td>
<td>_WORD, Output_label)</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_WORD ( )</td>
<td>Conversion source word [unsigned]/bit string [16-bit] data</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Bit data after conversion</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in ( ) into bit data, and outputs the data obtained by conversion to a device specified in ( ).

<table>
<thead>
<tr>
<th>Word [unsigned]/bit string [16-bit] data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
</tr>
<tr>
<td>1567H</td>
</tr>
</tbody>
</table>

Cautions
Use the function having "_E" in its name to connect a bus.
Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in $\text{a}$ is converted into bit data, and the data obtained by conversion is output to a device specified in $\text{d}$.

1) Function without EN/ENO(WORD_TO_BOOL)

![Structured ladder/FBD]

```
WORD_TO_BOOL

g_word1=16#0001
```

[ST]

```plaintext
g_bool1 := WORD_TO_BOOL(g_word1);
```

2) Function with EN/ENO(WORD_TO_BOOL_E)

![Structured ladder/FBD]

```
WORD_TO_BOOL_E

EN

g_word1
```

[ST]

```plaintext
g_bool3 := WORD_TO_BOOL_E(g_bool1, g_word1, g_bool2);
```
5.27 **DWORD_TO_BOOL(_E) / Double word [unsigned]/bit string [32-bit] data → bit data conversion**

**Outline**

This function converts double word [unsigned]/bit string [32-bit] data into bit data, and outputs the data obtained by conversion.

**1. Format**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_TO_BOOL</td>
<td><strong>DWORD_TO_BOOL(_DWORD);</strong> <strong>M0 := DWORD_TO_BOOL(Label);</strong></td>
</tr>
<tr>
<td>DWORD_TO_BOOL_E</td>
<td><strong>DWORD_TO_BOOL_E(EN, _DWORD, Output_label);</strong></td>
</tr>
<tr>
<td></td>
<td><strong>M0 := DWORD_TO_BOOL_E(X000, Label, M0);</strong></td>
</tr>
</tbody>
</table>

*1. Output variable*

**2. Set data**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td>Output</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>*1 (3)</td>
<td>Bit data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

**Explanation of function and operation**

This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in (5) into bit data, and outputs the data obtained by conversion to a device specified in (3).

<table>
<thead>
<tr>
<th>Double word [unsigned]/bit string [32-bit] data</th>
<th>Bit data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>FALSE</td>
</tr>
<tr>
<td>12345678H</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

**Cautions**

1) Use the function having " _E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in (5) is converted into bit data, and the data obtained by conversion is output to a device specified in (6).

1) Function without EN/ENO(DWORD_TO_BOOL)

[Structured ladder/FBD]

```
| g_dword1=16#00000001 | DWORD_TO_BOOL
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g_bool1</td>
<td>DWORD</td>
</tr>
</tbody>
</table>
```

[ST]

```
g_bool1 := DWORD_TO_BOOL(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_BOOL_E)

[Structured ladder/FBD]

```
| g_bool1 | DWORD_TO_BOOL_E
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g_dword1</td>
<td>EN</td>
</tr>
<tr>
<td></td>
<td>ENO</td>
</tr>
<tr>
<td></td>
<td>g_bool3</td>
</tr>
<tr>
<td></td>
<td>g_bool2</td>
</tr>
</tbody>
</table>
```

[ST]

```
g_bool3 := DWORD_TO_BOOL_E(g_bool1, g_dword1, g_bool2);
```
5.28 WORD_TO_INT(_E) / Word [unsigned]/bit string [16-bit] data → word [signed] data conversion

Outline
This function converts word [unsigned]/bit string [16-bit] data into word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD_TO_INT</td>
<td>D0 → WORD_TO_INT(_WORD)_D10</td>
</tr>
<tr>
<td>WORD_TO_INT_E</td>
<td>X000 → WORD_TO_INT_E(EN,_WORD,ENO,D10)</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_WORD ( )</td>
<td>Conversion source word [unsigned]/bit string [16-bit] data</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Word [signed] data after conversion</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in 5678H into word [signed] data, and outputs the data obtained by conversion to a device specified in 22136.

Cautions
Use the function having "_E" in its name to connect a bus.
Program example
In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in \( s \) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(WORD_TO_INT)

[Structured ladder/FBD]

\[
g_{\text{word1}}=16\#000A \quad \text{WORD_TO_INT} \quad \text{g_int1}=10
\]

[ST]
\[
g_{\text{int1}} := \text{WORD_TO_INT}(g_{\text{word1}});
\]

2) Function with EN/ENO(WORD_TO_INT_E)

[Structured ladder/FBD]

\[
g_{\text{bool1}} \quad \text{WORD_TO_INT_E} \quad \text{g_bool3}
\]

[ST]
\[
g_{\text{bool3}} := \text{WORD_TO_INT_E}(g_{\text{bool1}}, g_{\text{word1}}, g_{\text{int1}});
\]
5.29 WORD_TO_DINT(_E) / Word [unsigned]/bit string [16-bit] data → double word [signed] data conversion

Outline
This function converts word [unsigned]/bit string [16-bit] data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

### Outline
This function converts word [unsigned]/bit string [16-bit] data into double word [signed] data, and outputs the data obtained by conversion.

### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_WORD</td>
<td>Conversion source word [unsigned]/bit string [16-bit] data</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Double word [signed] data after conversion</td>
<td>Double Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

### Explanation of function and operation
This function converts word [unsigned]/bit string [16-bit] data storeds in a device specified in ° into double word [signed] data, and outputs the data obtained by conversion to a device specified in °.

<table>
<thead>
<tr>
<th>5678H</th>
<th>Double word [signed] data after conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010110110101110100</td>
<td>22136</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

### Cautions
1) Use the function having " _E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in \(s\) is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in \(d\).

1) Function without EN/ENO(WORD_TO_DINT)

```plaintext
[Structured ladder/FBD]
g_word1=16#1234

WORD_TO_DINT

WORD

s_g_dint1=4660
d_g_word1

[ST]
g_dint1 := WORD_TO_DINT(g_word1);
```

2) Function with EN/ENO(WORD_TO_DINT_E)

```plaintext
[Structured ladder/FBD]
g_bool1
g_word1

WORD_TO_DINT_E

EN

ENO

s_g_dint1

d_g_word1

[ST]
g_bool3 := WORD_TO_DINT_E(g_bool1, g_word1, g_dint1);
```
5.30 DWORD_TO_INT(_E) / Double word [unsigned]/bit string [32-bit] data → Word [signed] data conversion

Outline
This function converts double word [unsigned]/bit string [32-bit] data into word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_TO_INT</td>
<td>DWORD_TO_INT(Label)</td>
<td>D10</td>
</tr>
<tr>
<td>DWORD_TO_INT_E</td>
<td>DWORD_TO_INT_E(EN, ENO, Label)</td>
<td>D10</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>DWORD ( )</td>
<td>Conversion source double word [unsigned]/bit string [32-bit] data</td>
<td>Double Word [unsigned]/ Bit string [32-bit]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word [signed] data after conversion</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in ( ) into word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
1) Use the function having "E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in \( \text{5} \) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in \( \text{4} \).

1) Function without EN/ENO(DWORD_TO_INT)

([Structured ladder/FBD]

\[
g_{\text{dword1}} = 16\#00012345 \quad \text{DWORD TO INT} \quad g_{\text{int1}} = 9029
\]

[ST]

\[
g_{\text{int1}} := \text{DWORD_TO_INT}(g_{\text{dword1}});
\]

2) Function with EN/ENO(DWORD_TO_INT_E)

([Structured ladder/FBD]

\[
g_{\text{bool1}} \quad \text{DWORD_TO_INT_E} \quad \text{EN} \quad \text{ENO} \quad g_{\text{bool3}}
g_{\text{dword1}} \quad \text{DWORD} \quad g_{\text{int1}}
\]

[ST]

\[
g_{\text{bool3}} := \text{DWORD_TO_INT_E}(g_{\text{bool1}}, g_{\text{dword1}}, g_{\text{int1}});
\]
5.31 DWORD_TO_DINT(_E) / Double word [unsigned]/bit string [32-bit] data → double word [signed] data conversion

Outline
This function converts double word [unsigned]/bit string [32-bit] data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_TO_DINT</td>
<td>Structured ladder/FBD: DWORD_TO_DINT(DWORD); Execution condition Bit: EN, Conversion source double word [unsigned]/bit string [32-bit] data: _DWORD, Output double word [signed] data: ENO.</td>
</tr>
<tr>
<td>DWORD_TO_DINT_E</td>
<td>Structured ladder/FBD: DWORD_TO_DINT_E(EN, DWORD, Output_label); Execution condition Bit: EN, Conversion source double word [unsigned]/bit string [32-bit] data: _DWORD, Output double word [signed] data: ENO.</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_DWORD</td>
<td>Conversion source double word [unsigned]/bit string [32-bit] data</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Double word [signed] data after conversion</td>
<td>Double Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in into double word [signed] data, and outputs the data obtained by conversion to a device specified in .

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in (S) is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in (O).

1) Function without EN/ENO(DWORD_TO_DINT)

[Structured ladder/FBD]

```
g_dword1=16#00012345
```

```
DWORD_TO_DINT_dw
```

```
g_dint1=74565
```

```
[ST]
g_dint1 := DWORD_TO_DINT(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_DINT_E)

[Structured ladder/FBD]

```
g_bool1
```

```
g_dword1
```

```
DWORD_TO_DINT_E
```

```
EN
```

```
ENO
```

```
g_dint1
```

```
g_bool3
```

```
[ST]
g_bool3 := DWORD_TO_DINT_E(g_bool1, g_dword1, g_dint1);
```
Outline

This function converts word [unsigned]/bit string [16-bit] data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD_TO_DWORD  _E</td>
<td>WORD_TO_DWORD(_E)(EN, _WORD, Output_label); Example:</td>
</tr>
<tr>
<td></td>
<td>WORD_TO_DWORD(_E)(EN, X000, D0, Label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_WORD</td>
<td>Conversion source word [unsigned]/bit string [16-bit] data</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Double word [unsigned]/bit string [32-bit] data after conversion</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in ( ) into double word [unsigned]/bit [32-bit] data, and outputs the data obtained by conversion to a device specified in ( ). Each of high-order 16 bits becomes "0" after data conversion.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in (s) is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(WORD_TO_DWORD)

[Structured ladder/FBD]

```
| g_word1=16#1234 | WORD_TO_DWORD | g_dword1=16#00001234 |
```

[ST]

g_dword1 := WORD_TO_DWORD(g_word1);

2) Function with EN/ENO(WORD_TO_DWORD_E)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_bool1</th>
<th>g_word1</th>
<th>WORD_TO_DWORD_E</th>
<th>g_bool3</th>
<th>g_dword1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

[ST]

g_bool3 := WORD_TO_DWORD_E(g_bool1, g_word1, g_dword1);
5.33 DWORD_TO_WORD(_E) / Double word [unsigned]/bit string[32-bit] data → word [unsigned]/bit string [16-bit] data conversion

Outline
This function converts double word [unsigned]/bit string[32-bit] data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_TO_WORD</td>
<td>Label</td>
<td>DWORD_TO_WORD(_DWORD); D10=DWORD_TO_WORD(Label);</td>
<td></td>
</tr>
<tr>
<td>DWORD_TO_WORD(_E)</td>
<td>X000</td>
<td>DWORD_TO_WORD_E(EN, _DWORD, Output_label); D10=DWORD_TO_WORD_E(X000, Label, D10);</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_DWORD</td>
<td>Conversion source double word [unsigned]/bit string [32-bit] data</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word [unsigned]/bit string [16-bit] data after conversion</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in  into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in  

12345678H → 5678H

Double word [unsigned]/bit string [32-bit] data → Word [unsigned]/bit string [16-bit] data

The information stored in high-order 16 bits is discarded.

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in \( S \) is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in \( D \).

1) Function without EN/ENO(WORD_TO_DWORD)

[Structured ladder/FBD]

```
  g_dword1=16#12345678  DWORD_TO_WORD  g_word1=16#5678
```

[ST]

```
g_word1 := DWORD_TO_WORD(g_dword1);
```

2) Function with EN/ENO(WORD_TO_DWORD_E)

[Structured ladder/FBD]

```
g_bool1  DWORD_TO_WORD_E 
  g_dword1  EN  ENO  g_bool3
  DWORD  g_word1
```

[ST]

```
g_bool3 := DWORD_TO_WORD_E(g_bool1, g_dword1, g_word1);
```
5.34 WORD_TO_TIME( _E) / Word [unsigned]/bit string [16-bit] data →
time data conversion

Outline
This function converts word [unsigned]/bit string [16-bit] data into time data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD_TO_TIME</td>
<td>WORD_TO_TIME(_WORD);</td>
</tr>
<tr>
<td></td>
<td>Example: Label: WORD_TO_TIME(D0);</td>
</tr>
<tr>
<td>WORD_TO_TIME _E</td>
<td>WORD_TO_TIME_E(EN,_WORD, Output_label);</td>
</tr>
<tr>
<td></td>
<td>Example: LABEL_TO_TIME_E(X000,D0, Label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td>variable</td>
<td>_WORD ( )</td>
<td>Conversion source word [unsigned]/bit string [16-bit] data</td>
</tr>
<tr>
<td>Output</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td>variable</td>
<td>*1 ( )</td>
<td>Time data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in ( ) into time data, and outputs the data obtained by conversion to a device specified in ( )

| 0         | 0ms          |
| 1234      | 1s234ms      |

Word [unsigned]/bit string [16-bit] data → Time data

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in (s) is converted into time data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(WORD_TO_TIME)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th></th>
<th>WORD_TO_TIME</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g_word1</td>
<td></td>
<td>g_time1</td>
</tr>
</tbody>
</table>
```

[ST]

g_time1 := WORD_TO_TIME(g_word1);

2) Function with EN/ENO(WORD_TO_TIME_E)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th></th>
<th>WORD_TO_TIME_E</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g_bool1</td>
<td>EN</td>
<td>g_bool3</td>
</tr>
<tr>
<td>g_word1</td>
<td>ENO</td>
<td>g_time1</td>
</tr>
</tbody>
</table>
```

[ST]

g_bool3 := WORD_TO_TIME_E(g_bool1, g_word1, g_time1);
5.35 DWORD_TO_TIME(_E) / Double word [unsigned]/bit string [32-bit] data → time data conversion

Outline
This function converts double word [unsigned]/bit string [32-bit] data into time data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_TO_TIME_E</td>
<td>Label1 ─────────── DWORD_TO_TIME(EN, _DWORD, Output_label) ─── Label2</td>
</tr>
<tr>
<td>DWORD_TO_TIME_E_E</td>
<td>X000 ─── ─── DWORD_TO_TIME(EN, _DWORD, Output_label) ─── Label2</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_DWORD</td>
<td>Conversion source</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
<tr>
<td>*1 NO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Time data after conversion</td>
<td>Time</td>
</tr>
</tbody>
</table>

Explanation of function and operation
This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in ( ) into time data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
**Program example**

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in \( \text{G} \) is converted into time data, and the data obtained by conversion is output to a device specified in \( \text{D} \).

1) Function without EN/ENO(DWORD_TO_TIME)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{dword1}} \quad \text{DWORD_TO_TIME} \quad \text{g_time1} \\
\end{array}
\]

[ST]

\[g_{\text{time1}} := \text{DWORD_TO_TIME}(g_{\text{dword1}}) ;\]

2) Function with EN/ENO(DWORD_TO_TIME_E)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool1}} \quad \text{DWORD_TO_TIME_E} \quad \text{EN} \quad \text{ENO} \quad g_{\text{bool3}} \\
\end{array}
\]

[ST]

\[g_{\text{bool3}} := \text{DWORD_TO_TIME_E}(g_{\text{bool1}}, g_{\text{dword1}}, g_{\text{time1}}) ;\]
5.36 STR_TO_BOOL(_E) / String data → bit data conversion

Outline
This function converts string data into bit data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR_TO_BOOL</td>
<td>STR_TO_BOOL(_STRING);</td>
</tr>
<tr>
<td>STR_TO_BOOL_E</td>
<td>STR_TO_BOOL_E(EN, _STRING, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Bit</td>
</tr>
<tr>
<td>_STRING</td>
<td>Conversion source string data</td>
<td>String</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Bit data after conversion</td>
<td></td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts string data stored in a device specified in _STRING ( ) into bit data, and outputs the data obtained by conversion to a device specified in *1 ( ).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data.

Use global labels when specifying labels.
Program example

In this program, string data stored in a device specified in (s) is converted into bit data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(STR_TO_BOOL)

[Structured ladder/FBD]

```
| g_string1         | STR_TO_BOOL
| string            | g_bool1
```

[ST]
g_bool1 := STR_TO_BOOL(g_string1);

2) Function with EN/ENO(STR_TO_BOOL_E)

[Structured ladder/FBD]

```
| g_bool1          | STR_TO_BOOL_E
| g_string1        | EN
| string           | ENO
| g_bool3          | g_bool2
```

[ST]
g_bool3 := STR_TO_BOOL_E(g_bool1, g_string1, g_bool2);
5.37 STR_TO_INT(_E) / String data → word [signed] data conversion

Outline
This function converts string data into word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR_TO_INT</td>
<td>Label STR_TO_INT _STRING *1 D10</td>
</tr>
<tr>
<td>STR_TO_INT_E</td>
<td>X000 STR_TO_INT_E EN ENO *1 D10</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td>_STRING ( )</td>
<td>Conversion source string data</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>*1 ( )</td>
<td>Word [signed] data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts string data (3 words) stored in a device specified in (5) into word [signed] data, and outputs the data obtained by conversion to a device specified in (6).

<table>
<thead>
<tr>
<th>String</th>
<th>1st word</th>
<th>ASCII code for ten-thousands place</th>
<th>Sign data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>ASCII code for hundreds place ASCII code for thousands place</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd word</td>
<td>ASCII code for ones place ASCII code for tens place</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data. Use global labels when specifying labels.
Error

1) When the sign data (low-order byte) of the 1st word stored in a device specified in $S$ is any other than "20H (space)" or "2DH (-)"
   (Error code: K6706)

2) When the ASCII code for each place (digit) stored in $S$ to $S$+2 is any other than "30H" to "39H", "20H (space)" or "00H (NULL)"
   (Error code: K6706)

3) When the value stored in $S$ to $S$+2 is outside the following range:
   -32768 to +32767
   (Error code: K6706)

4) When any of devices $S$ to $S$+2 exceeds the device range
   (Error code: K6706)

Program example

In this program, string data stored in a device specified in $S$ is converted into word [signed] data, and the data obtained by conversion is output to a device specified in $D$.

1) Function without EN/ENO(STR_TO_INT)

   [Structured ladder/FBD]
   
   ```
   g_string1="-12345"                  STR_TO_INT_STRING
   g_int1 := STR_TO_INT(g_string1);
   ```

2) Function with EN/ENO(STR_TO_INT_E)

   [Structured ladder/FBD]
   
   ```
   g_bool1
   g_string1
   STR_TO_INT_E_STRING ENO g_bool3
   g_int1 := STR_TO_INT_E(g_bool1, g_string1, g_int1);
   ```
5.38 **STR_TO_DINT(_E)** / **String data → double word [signed] data conversion**

### Outline

This function converts string data into double word [signed] data, and outputs the data obtained by conversion.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR_TO_DINT</td>
<td>Structure ladder/FBD: <code>STR_TO_DINT(_:STRING);</code> Example: <code>STR_TO_DINT(Label1);</code></td>
</tr>
<tr>
<td>STR_TO_DINT_E</td>
<td>Structure ladder/FBD: <code>STR_TO_DINT_E(EN, _:STRING, Output_label);</code> Example: <code>STR_TO_DINT_E(X000, Label1, Label2);</code></td>
</tr>
</tbody>
</table>

*1. Output variable

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_STRING</td>
<td>Conversion source string data</td>
<td>String</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Double word [signed] data after conversion</td>
<td>Double Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

### Explanation of function and operation

This function converts string data (6 words) stored in a device specified in ( ) into double word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

<table>
<thead>
<tr>
<th>String</th>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>ASCII code for billions place</td>
<td>Sign data</td>
</tr>
<tr>
<td>2nd word</td>
<td>ASCII code for ten-millions place</td>
<td></td>
</tr>
<tr>
<td>3rd word</td>
<td>ASCII code for hundred-millions place</td>
<td></td>
</tr>
<tr>
<td>4th word</td>
<td>ASCII code for thousands place</td>
<td>ASCII code for ten-thousands place</td>
</tr>
<tr>
<td>5th word</td>
<td>ASCII code for hundred-thousands place</td>
<td>ASCII code for tens place</td>
</tr>
<tr>
<td>6th word</td>
<td>ASCII code for millions place</td>
<td>ASCII code for hundreds place</td>
</tr>
</tbody>
</table>

(Ignore)

### Cautions

1) Use the function having " _E" in its name to connect a bus.

2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Error

1) When the sign data (low-order byte) of the 1st word stored in a device specified in \( S \) is any other than "20H (space)" or "2DH (-)"
   (Error code: K6706)

2) When the ASCII code for each place (digit) stored in \( S \) to \( S + 5 \) is any other than "30H" to "39H", "20H (space)" or "00H (NULL)"
   (Error code: K6706)

3) When the value stored in \( S \) to \( S + 5 \) is outside the following range:
   -2,147,483,648 to +2,147,483,647
   (Error code: K6706)

4) When any of devices \( S \) to \( S + 5 \) exceeds the device range
   (Error code: K6706)

Program example

In this program, string data stored in a device specified in \( S \) is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in \( D \).

1) Function without EN/ENO(STR_TO_DINT)

   [Structured ladder/FBD]
   \[
   \begin{align*}
   \text{g_string1}=\_\_65000\_ & \quad \text{STR_TO_DINT} \\
   \text{g_dint1} = 65000
   \end{align*}
   \]

   [ST]
   \[
   \text{g_dint1} := \text{STR_TO_DINT}(\text{g_string1});
   \]

2) Function with EN/ENO(STR_TO_DINT_E)

   [Structured ladder/FBD]
   \[
   \begin{align*}
   \text{g_bool1} & \quad \text{g_string1} & \quad \text{STR_TO_DINT_E} \\
   \text{g_bool3} & \quad \text{g_dint1}
   \end{align*}
   \]

   [ST]
   \[
   \text{g_bool3} := \text{STR_TO_DINT_E}(\text{g_bool1}, \text{g_string1}, \text{g_dint1});
   \]
5.39 STR_TO_REAL(_E) / String data → float (single precision) data conversion

Outline

This function converts string data into float (single precision) data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
</tr>
</thead>
</table>
| STR_TO_REAL        | Label1  
|                    | STR_TO_REAL(_STRING)  
|                    | Label2                                        |                       |
| STR_TO_REAL_E      | X000  
|                    | STR_TO_REAL_E(EN,_STRING,  
|                    | Output_label)  
|                    | Label1  
|                    | STRING  
|                    | Label2                                        |                       |

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>EN</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>_STRING (3)</td>
<td>String</td>
</tr>
<tr>
<td>Output</td>
<td>ENO</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td><strong>1</strong> (3)</td>
<td>Float (single precision) data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function converts string data (in the decimal format or exponent format) stored in a device specified in ( ) into float (single precision) data, and outputs the data obtained by conversion to a device specified in ( ) .

---

<table>
<thead>
<tr>
<th>String</th>
<th>1st word</th>
<th>ASCII code for 1st character</th>
<th>Sign data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2nd word</td>
<td>ASCII code for 3rd character</td>
<td>ASCII code for 2nd character</td>
</tr>
<tr>
<td></td>
<td>3rd word</td>
<td>ASCII code for 5th character</td>
<td>ASCII code for 4th character</td>
</tr>
<tr>
<td></td>
<td>4th word</td>
<td>ASCII code for 7th character</td>
<td>ASCII code for 6th character</td>
</tr>
<tr>
<td></td>
<td>5th word</td>
<td>ASCII code for 9th character</td>
<td>ASCII code for 8th character</td>
</tr>
<tr>
<td></td>
<td>6th word</td>
<td>ASCII code for 11th character</td>
<td>ASCII code for 10th character</td>
</tr>
<tr>
<td></td>
<td>7th word</td>
<td>0000H (Indicates the end of the character string)</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>String</th>
<th>1st word</th>
<th>31H (1)</th>
<th>2DH (-)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2nd word</td>
<td>33H (3)</td>
<td>2EH (-)</td>
</tr>
<tr>
<td></td>
<td>3rd word</td>
<td>30H (0)</td>
<td>35H (5)</td>
</tr>
<tr>
<td></td>
<td>4th word</td>
<td>34H (4)</td>
<td>33H (3)</td>
</tr>
<tr>
<td></td>
<td>5th word</td>
<td>0000H</td>
<td>-1.35034</td>
</tr>
</tbody>
</table>

2) The conversion source string data can be in the decimal format or exponent format.

a) In the case of decimal format
b) In the case of exponent format

<table>
<thead>
<tr>
<th>String</th>
<th>1st word</th>
<th>2nd word</th>
<th>3rd word</th>
<th>4th word</th>
<th>5th word</th>
<th>6th word</th>
<th>7th word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31H (1)</td>
<td>2DH (-)</td>
<td>33H (3)</td>
<td>2EH (3)</td>
<td>30H (0)</td>
<td>33H (3)</td>
<td>31H (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Discarded} \]

\[ \text{Float (single precision) data} \]

3) With regard to string data, six digits excluding the sign, decimal point and exponent part are valid, and the 7th and later digits are discarded during conversion.

a) In the case of decimal format

\[ \cdots 35034 1 2 0 2 3 \Rightarrow -1.35034 \]

b) In the case of exponent format

\[ \cdots 35034 1 2 E \cdots \Rightarrow -1.35034E-10 \]

4) String data in the decimal format is handled as positive value during conversion when the sign is set to "2BH (+)" or when the sign is omitted. It is handled as negative value during conversion when the sign is set to "2DH (-)".

5) String data in the exponent format is handled as positive value during conversion when the sign of the exponent part is set to "2BH (+)" or when the sign is omitted. It is handled as negative value during conversion when the sign is set to "2DH (-)".

6) When "20H (space)" or "30H (0)" exists between the sign and the first number except "0" in string data, "20H (space)" or "30H (0)" is ignored during conversion.

a) In the case of decimal format

\[ \cdots 0 \cdots 35034 \Rightarrow -1.35034 \]

b) In the case of exponent format

\[ \cdots 0 \cdots 35034 \Rightarrow -1.35034E-10 \]

7) When "30H (0)" exists between "E" and a number in character string data (in the exponent format), "30H (0)" is ignored during conversion.

\[ \cdots 35034 \cdots \Rightarrow -1.35034E-2 \]

8) When "20H (space)" is contained in character string, "20H (space)" is ignored during conversion.

9) Up to 24 characters can be input as string data.

Each of "20H (space)" and "30H (0)" contained in string is counted as 1 character respectively.

**Cautions**

1) Use the function having "_E" in its name to connect a bus.

2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.
Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When any character other than "30H (0)" to "39H (9)" exists in the integer or decimal part (Error code: K6706)

2) When "2EH (.)" exists in two or more positions inside the character string specified in \( s \) (Error code: K6706)

3) When any character other than "45H (E)" , "2BH (+)" or "2DH (-)" exists in the exponent part, or when two or more exponent parts exist (Error code: K6706)

4) When the number of characters after \( \leq \) is "0" or any value larger than "24" (Error code: K6706)

Program example

In this program, string data stored in a device specified in \( s \) is converted into float (single precision) data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(STR_TO_REAL)

[Structured ladder/FBD]

\[
g_{\text{str}} \rightarrow \text{STR_TO_REAL} \rightarrow g_{\text{real}}
\]

[ST]

\[
g_{\text{real}} := \text{STR_TO_REAL}(g_{\text{str}});
\]

2) Function with EN/ENO(STR_TO_REAL_E)

[Structured ladder/FBD]

\[
g_{\text{bool}} \rightarrow \text{EN} \rightarrow \text{STR_TO_REAL_E} \rightarrow \text{ENO} \rightarrow g_{\text{bool}} \rightarrow g_{\text{real}}
\]

[ST]

\[
g_{\text{bool}} := \text{STR_TO_REAL_E}(g_{\text{bool}}, g_{\text{str}}, g_{\text{real}});
\]
5.40 STR_TO_TIME(_E) / String data → time data conversion

Outline
This function converts string data into time data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR_TO_TIME</td>
<td>STR_TO_TIME(_STRING);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Label2:= STR_TO_TIME(_STRING); Label1;</td>
</tr>
<tr>
<td>STR_TO_TIME_E</td>
<td>STR_TO_TIME_E(EN_STRING, Output_label);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>STR_TO_TIME_E(X000, Label1, Label2);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition Bit</td>
</tr>
<tr>
<td></td>
<td>_STRING ( )</td>
<td>Conversion source string data String</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status Bit</td>
</tr>
<tr>
<td></td>
<td>*1 ( )</td>
<td>Time data after conversion Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts string data stored in a device specified in ( ) into time data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the sign data of numeric data specified in \( \text{\textbf{5}} \) is any other than "20H (space)" or "2DH (-)"
   (Error code: K6706)

2) When the ASCII code for each digit of character string data specified in \( \text{\textbf{5}} \) is any other than "30H (0)" to "39H (9)", "20H (space)" or "00H (NULL)"
   (Error code: K6706)

3) When the numeric value specified in \( \text{\textbf{5}} \) is outside the following range:
   -2,147,483,648 to +2,147,483,647
   (Error code: K6706)

Program example

In this program, string data stored in a device specified in \( \text{\textbf{5}} \) is converted into time data, and the data obtained by conversion is output to a device specified in \( \text{\textbf{4}} \).

1) Function without EN/ENO(STR_TO_TIME)

   [Structured ladder/FBD]
   
   g_string1 \( \text{\textbf{STR_TO_TIME}} \) STRING g_time1

   [ST]
   g_time1 := STR_TO_TIME(g_string1);

2) Function with EN/ENO(STR_TO_TIME_E)

   [Structured ladder/FBD]
   
   g_bool1 \( \text{\textbf{STR_TO_TIME_E}} \) EN STRING ENO g_bool3 g_time1

   [ST]
   g_bool3 := STR_TO_TIME_E(g_bool1, g_string1, g_time1);
5.41 BCD_TO_INT(_E) / BCD data → word [signed] data conversion

### Outline
This function converts BCD data into word [signed] data, and outputs the data obtained by conversion.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCD_TO_INT</td>
<td>D0 BCD_TO_INT(_BCD) → D10 *1 BCD_TO_INT(BCD);</td>
</tr>
<tr>
<td>BCD_TO_INT_E</td>
<td>X000 BCD_TO_INT_E(_BCD) → D0 EN D10 ENO BCD_TO_INT_E(EN,_BCD, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_BCD</td>
<td>Conversion source BCD data</td>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
<tr>
<td>Output</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word [signed] data after conversion</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

### Explanation of function and operation
This function converts BCD data stored in a device specified in ( ) into word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

- **Word [unsigned]/bit string [16-bit] data**
- **Word [signed] data**

#### Cautions
Use the function having "_E" in its name to connect a bus.

#### Error
When the source data is not BCD (decimal number), M8067 (operation error) turns ON.
Program example
In this example, BCD data stored in a device specified in (1) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in (2).

1) Function without EN/ENO(BCD_TO_INT)

[Structured ladder/FBD]

```
  g_word1=16#1234  BCD_TO_INT  g_int1=1234
```

[ST]

```
g_int1 := BCD_TO_INT(g_word1);
```

2) Function with EN/ENO(BCD_TO_INT_E)

[Structured ladder/FBD]

```
g_bool1  BCD_TO_INT_E  g_bool3
  g_word1         g_int1
```

[ST]

```
g_bool3 := BCD_TO_INT_E(g_bool1, g_word1, g_int1);
```
### 5.42 BCD_TO_DINT(_E) / BCD data → double word [signed] data conversion

#### Outline
This function converts BCD data into double word [signed] data, and outputs the data obtained by conversion.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structure in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCD_TO_DINT</td>
<td><img src="BCD_TO_DINT.png" alt="Diagram" /></td>
</tr>
<tr>
<td>BCD_TO_DINT_E</td>
<td><img src="BCD_TO_DINT_E.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Expression in each language**

- **Structured ladder/FBD**: `BCD_TO_DINT(_BCD) ;`
- **ST**: `BCD_TO_DINT(_BCD) ;`

**Example:**

- `BCD_TO_DINT(_BCD) ;`
- `BCD_TO_DINT(_BCD) ;

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_BCD</td>
<td>Conversion source BCD data</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>Output variable:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Double word [signed] data after conversion</td>
<td>Double Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

#### Explanation of function and operation

This function converts BCD data stored in a device specified in ( ) into double word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

**Example:**

- `99999999H` → `99999999`

#### Cautions

Use the function having "_E" in its name to connect a bus.

#### Error

When the source data is not BCD (decimal number), M8067 (operation error) turns ON.
Program example

In this example, BCD data stored in a device specified in $\text{g}_5$ is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in $\text{d}_4$.

1) Function without EN/ENO(BCD_TO_DINT)

[Structured ladder/FBD]

$$\text{g}_5 \text{word1}=16\#0000 \quad \text{BCD} \quad \text{g}_4 \text{dint1}=0$$

[ST]

$$\text{g}_4 \text{dint1} := \text{BCD\_TO\_DINT(} \text{g}_5 \text{word1});$$

2) Function with EN/ENO(BCD_TO_DINT_E)

[Structured ladder/FBD]

$$\text{g}_3 \text{bool1} \quad \text{BCD\_TO\_DINT\_E} \quad \text{g}_4 \text{bool3}$$

[ST]

$$\text{g}_4 \text{bool3} := \text{BCD\_TO\_DINT\_E(} \text{g}_3 \text{bool1, g}_5 \text{word1, g}_4 \text{dint1});$$
5.43 TIME_TO_BOOL(_E) / Time data → bit data conversion

Outline
This function converts time data into bit data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_TO_BOOL</td>
<td>Label TIME_TO_BOOL(_TIME); M0</td>
<td>TIME_TO_BOOL(_TIME);</td>
</tr>
<tr>
<td>TIME_TO_BOOL_E</td>
<td>X000 TIME_TO_BOOL_E(EN, _TIME, Output_label); M0</td>
<td>TIME_TO_BOOL_E(EN, _TIME, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_TIME (<em>s</em>)</td>
<td>Conversion source time data</td>
<td>Time</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (<em>d</em>)</td>
<td>Bit data after conversion</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts time data stored in a device specified in (_s_) into bit data, and outputs the data obtained by conversion to a device specified in (_d_).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
Program example

In this program, time data stored in a device specified in 5 is converted into bit data, and the data obtained by conversion is output to a device specified in 6.

1) Function without EN/ENO(TIME_TO_BOOL)

[Structured ladder/FBD]

```
g_time1  TIME
       TIME_TO_BOOL
      _TIME
       g_bool1

[ST]
g_bool1 := TIME_TO_BOOL(g_time1);
```

2) Function with EN/ENO(TIME_TO_BOOL_E)

[Structured ladder/FBD]

```
g_bool1  TIME
       TIME_TO_BOOL_E
      _TIME
      _EN
       g_bool3

[ST]
g_bool3 := TIME_TO_BOOL_E(g_bool1, g_time1, g_bool2);
```
5.44 TIME_TO_INT(_E) / Time data → word [signed] data conversion

Outline
This function converts time data into word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_TO_INT</td>
<td>TIME_TO_INT(_TIME);</td>
</tr>
<tr>
<td>TIME_TO_INT_E</td>
<td>TIME_TO_INT_E(EN,_TIME, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_TIME</td>
<td>Conversion source time data</td>
<td>Time</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word [signed] data after conversion</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts time data stored in a device specified in ( ) into word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

<table>
<thead>
<tr>
<th>Time data</th>
<th>Word [signed] data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s234ms</td>
<td>1234</td>
</tr>
</tbody>
</table>

Cautions
1) Use the function having " _E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
Program example

In this program, time data stored in a device specified in \( g_{\text{time1}} \) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in \( g_{\text{int1}} \).

1) Function without EN/ENO(TIME_TO_INT)

[Structured ladder/FBD]
\[
\begin{array}{c}
g_{\text{time1}} \\
\text{TIME TO INT} \\
g_{\text{int1}}
\end{array}
\]

[ST]
\[ g_{\text{int1}} := \text{TIME}_\text{TO}_\text{INT}(g_{\text{time1}}); \]

2) Function with EN/ENO(TIME_TO_INT_E)

[Structured ladder/FBD]
\[
\begin{array}{c}
g_{\text{bool1}} \\
\text{TIME TO INT}_\text{E} \\
g_{\text{bool3}}
\end{array}
\begin{array}{c}
\text{EN} \\
\text{TIME} \\
\text{ENO}
\end{array}
\begin{array}{c}
g_{\text{time1}} \\
\text{TIME} \\
g_{\text{int1}}
\end{array}
\]

[ST]
\[ g_{\text{bool3}} := \text{TIME}_\text{TO}_\text{INT}_\text{E}(g_{\text{bool1}}, g_{\text{time1}}, g_{\text{int1}}); \]
5.45 TIME_TO_DINT(_E) / Time data → double word [signed] data conversion

Outline
This function converts time data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured ladder/FBD</td>
<td>ST</td>
</tr>
<tr>
<td>TIME_TO_DINT</td>
<td>TIME_TO_DINT TIME *1</td>
</tr>
<tr>
<td>TIME_TO_DINT_E</td>
<td>TIME_TO_DINT_E EN ENO</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td>TIME ( )</td>
<td>Conversion source time data</td>
</tr>
<tr>
<td>Output</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>*1 ( )</td>
<td>Double word [signed] data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts time data stored in a device specified in ( ) into double word [signed] data, and outputs the data obtained by conversion to a device specified in ( ).

Cautions
1. Use the function having "_E" in its name to connect a bus.
2. When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
Program example

In this program, time data stored in a device specified in \( g \) is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(TIME_TO_DINT)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_time1} & \quad \text{TIME\_TO\_DINT} \\
\text{g\_dint1} & \quad \text{g\_time1}\end{align*}
\]

[ST]
\[\text{g\_dint1} := \text{TIME\_TO\_DINT(g\_time1)};\]

2) Function with EN/ENO(TIME_TO_DINT_E)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_bool1} & \quad \text{TIME\_TO\_DINT\_E} \\
\text{EN} & \quad \text{ENO} \\
\text{g\_time1} & \quad \text{g\_dint1} \\
\text{g\_bool3} & \quad \text{g\_bool1, g\_time1, g\_dint1}\end{align*}
\]

[ST]
\[\text{g\_bool3} := \text{TIME\_TO\_DINT\_E(g\_bool1, g\_time1, g\_dint1)};\]
5.46 TIME_TO_STR(E) / Time data → string data conversion

Outline

This function converts time data into string data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_TO_STR</td>
<td>Label1 ← TIME_TO_STR _TIME *1 ← Label2</td>
<td>TIME_TO_STR(_TIME); Label2:= TIME_TO_STR(Label1);</td>
<td></td>
</tr>
<tr>
<td>TIME_TO_STR_E</td>
<td>X000 ← TIME_TO_STR_E EN ENO ← _TIME *1 ← Label2</td>
<td>TIME_TO_STR_E(EN,_TIME, Output_label); Example: TIME_TO_STR_E(X000, Label1, Label2);</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td>_TIME (₃₄₅₆₇₈₉₀)</td>
<td>Conversion source time data</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>*1 (₃₄₅₆₇₈₉₀)</td>
<td>String data after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts time data stored in a device specified in (₃₄₅₆₇₈₉₀) into string data, and outputs the data obtained by conversion to a device specified in (₃₄₅₆₇₈₉₀).

| 20m34s567ms | 1234567 |
| Time data | String data |

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

Error

An operation error occurs in the following case. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the number of points occupied by the device specified in (₃₄₅₆₇₈₉₀) exceeds the range of the corresponding device.
Program example
In this program, time data stored in a device specified in $s$ is converted into string data, and the data obtained by conversion is output to a device specified in $d$.

1) Function without EN/ENO(TIME_TO_STR)

[Structured ladder/FBD]

```
g_time1
    TIME_TO_STR
    TIME
    g_string1
```

[ST]

```
g_string1 := TIME_TO_STR(g_time1);
```

2) Function with EN/ENO(TIME_TO_STR_E)

[Structured ladder/FBD]

```
g_bool1
    TIME_TO_STR_E
    EN
    TIME
    ENO
    g_bool3
    g_string1
    g_time1
```

[ST]

```
g_bool3 := TIME_TO_STR_E(g_bool1, g_time1, g_string1);
```
5.47 **TIME_TO_WORD(_E)** / Time data → word [unsigned]/bit string [16-bit] data conversion

### Outline

This function converts time data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

**1. Format**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
</table>
| TIME_TO_WORD  | Label
| TIME_TO_WORD(_E) | X000 |

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EN</strong></td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>_TIME ( )</strong></td>
<td>Conversion source time data</td>
<td>Time</td>
</tr>
<tr>
<td><strong>ENO</strong></td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>_<em>1</em> ( )</strong></td>
<td>Word [unsigned]/bit string [16-bit] data after conversion</td>
<td>Word [unsigned]/bit string [16-bit]</td>
</tr>
</tbody>
</table>

*1. Output variable

**2. Set data**

In explanation of functions, I/O variables inside ( ) are described.

**Explanation of function and operation**

This function converts time data stored in a device specified in ( ) into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in ( ).

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time data</td>
</tr>
</tbody>
</table>

**Cautions**

1) Use the function having " _E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.
Program example

In this program, time data stored in a device specified in \( \text{s} \) is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in \( \text{d} \).

1) Function without EN/ENO(TIME_TO_WORD)

[Structured ladder/FBD]

\[
g\text{_time1} \rightarrow \text{TIME_TO_WORD(}\_\text{TIME} \rightarrow g\text{_word1}
\]

[ST]
\[g\text{_word1} := \text{TIME_TO_WORD}(g\text{_time1});\]

2) Function with EN/ENO(TIME_TO_WORD_E)

[Structured ladder/FBD]

\[
g\text{_bool1} \rightarrow \text{TIME_TO_WORD_E(}\_\text{TIME} \leftarrow \text{EN} \rightarrow g\text{_bool3}
\]

[ST]
\[g\text{_bool3} := \text{TIME_TO_WORD_E}(g\text{_bool1}, g\text{_time1}, g\text{_word1});\]
5.48 TIME_TODWORD(_E) / Time data → double word [unsigned]/bit string [32-bit] data conversion

Outline
This function converts time data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_TODWORD</td>
<td>TIME_TODWORD(_TIME); Label1 → Label2</td>
</tr>
<tr>
<td>TIME_TODWORD_E</td>
<td>TIME_TODWORD_E(EN, _TIME, Output_label); Label1 → Label2</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_TIME ()</td>
<td>Conversion source time data</td>
<td>Time</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ()</td>
<td>Double word [unsigned]/bit string [32-bit] data after conversion</td>
<td>Double Word [unsigned]/Bit string [32-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts time data stored in a device specified in _S_ into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in _D_.

Cautions
1) Use the function having " _E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
**Program example**

In this program, time data stored in a device specified in \( g \) is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(TIME_TO_DWORD)

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_time1</td>
</tr>
</tbody>
</table>

   \[ST\]
   \[\] \( g\_dword1 := \text{TIME\_TO\_DWORD}(g\_time1); \)

2) Function with EN/ENO(TIME\_TO\_DWORD\_E)

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_bool1</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

   \[ST\]
   \[\] \( g\_dword1 := \text{TIME\_TO\_DWORD}(g\_time1); \)
5.49 BITARR_TO_INT(_E) / Bit array → Word [signed] type, word [unsigned]/bit String [16-bit] data conversion

Outline
This function converts specified number of bits of a bit array into word [signed] data or word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITARR_TO_INT</td>
<td>Structured ladder/FBD BITARR_TO_INT(BitArr, n);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>D10 := BITARR_TO_INT(Label1[*2], K4);</td>
</tr>
<tr>
<td>BITARR_TO_INT_E</td>
<td>ST BITARR_TO_INT_E(EN, BitArr, n, Output_label);</td>
</tr>
<tr>
<td></td>
<td>Example: BITARR_TO_INT_E(X000, Label1[*2], K4, D10);</td>
</tr>
</tbody>
</table>

*1. Output variable  
*2. Specify an array element.

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>BitArr ( ) Start bit of conversion source bit array elements</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Number of specified bits</td>
<td>Word [signed]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot; ( ) Word [signed] data or word [unsigned]/bit [16-bit] data after conversion</td>
<td>ANY16</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function converts number of bits specified in ( ) starting from a bit array element stored in a device specified in ( ) into word [signed] data or word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in ( ).
Only a constant 4, 8, 12 or 16 can be specified in ( ).
"0" is set to output bits beyond the specified number of bits.

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
Program example

In this program, 8 bits starting from the 0th element of a bit array stored in a device specified in \( s \) are converted into word [signed] data, and the data obtained by conversion is output to a device specified in \( d \).

1) Function without EN/ENO(BITARR_TO_INT)

[Structured ladder/FBD]

```
BITARR_TO_INT

n
K8
```

\[ g\_int1 := \text{BITARR\_TO\_INT}(g\_bool4[0], K8); \]

2) Function with EN/ENO(BITARR_TO_INT_E)

[Structured ladder/FBD]

```
BITARR_TO_INT_E

n
EN
ENO
```

\[ g\_int1 := \text{BITARR\_TO\_INT\_E}(g\_bool1, g\_bool4[0], K8, g\_int1); \]
5.50 BITARR_TO_DINT(_E) / Bit array → Double word [signed] type, double word [unsigned]/bit string [32-bit] data conversion

Outline

This function converts specified number of bits of a bit array into double word [signed] data or double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>BitArray to Double Word Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITARR_TO_DINT</td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td>BITARR_TO_DINT_E</td>
<td>ST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>BitArr（_E）</td>
<td>Start bit of conversion source bit array elements</td>
<td>Bit</td>
</tr>
<tr>
<td>n</td>
<td>Number of specified bits</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Double word [signed] data or double word [unsigned]/bit string [32-bit] data after conversion</td>
<td>ANY32</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts number of bits specified in ( ) starting from a bit array element stored in a device specified in ( ) into double word [signed] data or double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in ( ).

Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified in ( ). “0” is set to output bits beyond the specified number of bits.

Cautions

1) Use the function having " _E" in its name to connect a bus.

2) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, 8 bits starting from the 0th element of a bit array stored in a device specified in (s) are converted into double word [signed] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BITARR_TO_DINT)

[Structured ladder/FBD]

```
BITARR_TO_DINT

n
K8

BITARR_TO_DINT(g_bool4[0], K8);
```

2) Function with EN/ENO(BITARR_TO_DINT_E)

[Structured ladder/FBD]

```
BITARR_TO_DINT_E

EN
ENO
n

BITARR_TO_DINT_E(g_bool1, g_bool4[0], K8, g_dint1);
```

```plaintext
s

BITARR_TO_DINT

n
K8

BITARR_TO_DINT_E

g_dint1

BITARR_TO_DINT_E(g_bool1, g_bool4[0], K8, g_dint1);
```
### 5.51 INT_TO_BITARR\(_E\) / Word [signed] data, word [unsigned]/bit string [16-bit] data → bit array conversion

**Outline**
This function outputs low-order "n" bits of word [signed] data or word [unsigned]/bit string [16-bit] data to a bit array.

**1. Format**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_BITARR</td>
<td>( INT_TO_BITARR(s, n); )</td>
</tr>
<tr>
<td>INT_TO_BITARR(_E)</td>
<td>( INT_TO_BITARR(_E)(EN, s, n, BitArr); )</td>
</tr>
</tbody>
</table>

*1. Specify an array element.

**2. Set data**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conversion source word [signed] data or word [unsigned]/bit string [16-bit] data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of specified bits</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start bit of bit array elements after conversion</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

**Explanation of function and operation**
This function outputs low-order "n" bits of word [signed] data or word [unsigned]/bit string [16-bit] data stored in a device specified in ( ) to a device specified in ( ).

Only a constant 4, 8, 12 or 16 can be specified in ( ).
Output bits beyond the specified number of bits are not changed.

**Cautions**
1) Use the function having "\_E" in its name to connect a bus.
2) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, low-order 4 bits of word [signed] data stored in a device specified in $\text{a}$ are output to a device specified in $\text{b}$.

1) Function without EN/ENO(INT_TO_BITARR)

[Structured ladder/FBD]

$\text{g_int1} \rightarrow \text{INT_TO_BITARR} \rightarrow \text{g_bool4[0]}$

[K4]

$\text{BitArr}$

$\text{n}$

[ST]

$\text{g_bool4[0]} := \text{INT_TO_BITARR} (\text{g_int1}, \text{K4})$;

2) Function with EN/ENO(INT_TO_BITARR_E)

[Structured ladder/FBD]

$\text{g_bool1} \rightarrow \text{INT_TO_BITARR_E} \rightarrow \text{g_bool2}$

$\text{g_int1}$

$\text{K4}$

[ST]

$\text{EN}$

$\text{ENO}$

$\text{s}$

$\text{BitArr}$

$\text{n}$

$\text{g_bool2} := \text{INT_TO_BITARR_E} (\text{g_bool1, g_int1, K4, g_bool4[0]})$;
5.52 DINT_TO_BITARR(_E) / Double word [signed] data, double word [unsigned]/bit string [32-bit] data → bit array conversion

Outline
This function outputs low-order "n" bits of double word [signed] data or double word [unsigned]/bit string [32-bit] data to a bit array.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINT_TO_BITARR</td>
<td>DINT_TO_BITARR Label1, Label2[*1]</td>
</tr>
<tr>
<td>DINT_TO_BITARR_E</td>
<td>DINT_TO_BITARR_E EN, Label1, Label2[*1]</td>
</tr>
</tbody>
</table>

1. Specify an array element.

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Conversion source double word [signed] data or double word [unsigned]/bit string [32-bit] data</td>
<td>ANY32</td>
</tr>
<tr>
<td></td>
<td>Number of specified bits</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>BitArr</td>
<td>Start bit of bit array elements after conversion</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function outputs low-order "n" bits of double word [signed] data or double word [unsigned]/bit string [32-bit] data stored in a device specified in ☐ to a device specified in ☒.
Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified in ☒.
Output bits beyond the specified number of bits are not changed.

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, low-order 4 bits of double word [signed] data stored in a device specified in (g) are output to a device specified in (s).

1) Function without EN/ENO(DINT_TO_BITARR)

[Structured ladder/FBD]

```
g_dint1
K4

DINT_TO_BITARR
s
BitArr
n
```

[ST]

\[
g_{\text{bool4}}[0] := \text{DINT\_TO\_BITARR}(g_{\text{dint1}}, K4);
\]

2) Function with EN/ENO(INT_TO_BITARR_E)

[Structured ladder/FBD]

```
g_{\text{bool1}}
g_{\text{dint1}}
K4

DINT_TO_BITARR_E
EN
ENO
s
BitArr
n
```

[ST]

\[
g_{\text{bool2}} := \text{DINT\_TO\_BITARR\_E}(g_{\text{bool1}}, g_{\text{dint1}}, K4, g_{\text{bool4}}[0]);
\]
5.53 CPY_BITARR(_E) / Bit array copy

Outline

This function copies specified number of bits of a bit array.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPY_BITARR</td>
<td>CPY_BITARR(BitArrIn, n);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: Label2[*1]= CPY_BITARR(Label1[*1], K4);</td>
<td></td>
</tr>
<tr>
<td>CPY_BITARR_E</td>
<td>CPY_BITARR_E(EN, BitArrIn,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>n, BitArrOut, n, BitArrOut);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example: CPY_BITARR_E(X000,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Label1[*1], K4, Label2[*1]);</td>
<td></td>
</tr>
</tbody>
</table>

* Specify an array element.

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>BitArrIn</td>
<td>Start bit of copy source bit array elements</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Number of specified bits</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>BitArrOut</td>
<td>Start bit of copy destination bit array elements</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function outputs "n" bits of a bit array stored in a device specified in ( ) to a device specified in ( ). Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified in ( ).

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, 12 bits starting from the "num1"th element of a bit array stored in a device specified in are output to the "num2"th element and later of a bit array stored in device specified in .

1) Function without EN/ENO(CPY_BITARR)

[Structured ladder/FBD]

```
g_bool4[num1]  g_bool5[num2]
  CPY_BITARR
  BitArrIn  BitArrOut
  K12   n
```

[ST]

```
g_bool5[num2] := CPY_BITARR(g_bool4[num1], K12);
```

2) Function with EN/ENO(CPY_BITARR_E)

[Structured ladder/FBD]

```
g_bool1  g_bool2
     CPY_BITARR_E
   EN    ENO  g_bool2
  g_bool4[num1]  g_bool5[num2]
     BitArrIn  BitArrOut
     K12   n
```

[ST]

```
g_bool2 := CPY_BITARR_E(g_bool1, g_bool4[num1], K12, g_bool5[num2]);
```
5.54 GET_BIT_OF_INT(_E) / Specified bit read of word [signed] data

Outline
This function reads a value of a specified bit of word [signed] data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_BIT_OF_INT</td>
<td>D10→s K2→n *1 M10</td>
<td></td>
</tr>
<tr>
<td>GET_BIT_OF_INT_E</td>
<td>X000 EN ENO→ D10→s K2→n *1 M10</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Read source word [signed] data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td></td>
<td>Specified bit position</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>Read destination bit</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function reads the value of the "n"th bit of a device specified in (s), and outputs the read value to a device specified in (d).

Cautions
1) Use the function having ",_E" in its name to connect a bus.
Program example

In this program, the value of the 5th bit of a device specified in \( s \) is read, and the read value is output to a device specified in \( d \).

1) Function without EN/ENO(GET_BIT_OF_INT)

[Structured ladder/FBD]

\[
\text{g\_int1} \\
\text{K5}
\]

\[
\text{GET\_BIT\_OF\_INT}
\]

\[
\text{g\_bool3}
\]

[ST]

\[
g\_bool3 := \text{GET\_BIT\_OF\_INT}(g\_int1, K5);
\]

2) Function with EN/ENO(GET_BIT_OF_INT_E)

[Structured ladder/FBD]

\[
\text{g\_bool1} \\
\text{g\_int1} \\
\text{K5}
\]

\[
\text{GET\_BIT\_OF\_INT\_E}
\]

\[
\text{EN} \\
\text{ENO} \\
\text{g\_bool2} \\
\text{g\_bool3}
\]

[ST]

\[
g\_bool2 := \text{GET\_BIT\_OF\_INT\_E}(g\_bool1, g\_int1, K5, g\_bool3);
\]
5.55 SET_BIT_OF_INT(_E) / Specified bit write of word [signed] data

Outline
This function writes a value to a specified bit of word [signed] data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_BIT_OF_INT</td>
<td><code>M0 → s, K4 → n *1 → D10</code></td>
<td><code>SET_BIT_OF_INT(s, n);</code> Example: <code>D10 = SET_BIT_OF_INT(M0, K4);</code></td>
</tr>
<tr>
<td>SET_BIT_OF_INT_E</td>
<td><code>X000 → EN, M0 → s, K4 → n *1 → D10</code></td>
<td><code>SET_BIT_OF_INT_E(EN, s, n, *1);</code> Example: <code>SET_BIT_OF_INT_E(X000, M0, K4, D10);</code></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write source bit data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specified bit position</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td>Write destination word [signed] data</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function reads the value of a device specified in ( ), and writes the read value to the "n"th bit of a device specified in ( )

Cautions
1) Use the function having "_E" in its name to connect a bus.
Program example

In this program, the value of a device specified in $d_1$ is written to the 3rd bit of a device specified in $d_2$.

1) Function without EN/ENO(SET_BIT_OF_INT)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_bool1</th>
<th>s</th>
<th>g_int3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K3</td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>
```

[ST]

$g_{int3} := \text{SET\_BIT\_OF\_INT}(g_{bool1}, K3)$;

2) Function with EN/ENO(SET_BIT_OF_INT_E)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_bool1</th>
<th>s</th>
<th>g_int3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K3</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>g_bool2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>g_bool1</th>
<th>s</th>
<th>g_bool3</th>
</tr>
</thead>
<tbody>
<tr>
<td>K3</td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>
```

[ST]

$g_{bool3} := \text{SET\_BIT\_OF\_INT\_E}(g_{bool2}, g_{bool1}, K3, g_{int3})$;
5.56 CPY_BIT_OF_INT(_E) / Specified bit copy of word [signed] data

Outline
This function copies a specified bit of word [signed] data to a specified bit of another word [signed] data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPY_BIT_OF_INT</td>
<td>CPY_BIT_OF_INT(s, n1, n2);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D10 = CPY_BIT_OF_INT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(D0, K1, K4);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPY_BIT_OF_INT_E</td>
<td>CPY_BIT_OF_INT_E(EN, s,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n1, n2, *1);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPY_BIT_OF_INT_E(X000,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D0, K1, K4, D10);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>s</td>
<td>Copy source word [signed] data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>n1</td>
<td>Specified bit position (copy source)</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>n2</td>
<td>Specified bit position (copy destination)</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>s, n1, n2, *1</td>
<td>Copy destination word [signed] data</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function copies the value of the "n1"th bit of a device specified in (s) to the "n2"th bit of a device specified in (d).

Cautions
1) Use the function having ".E" in its name to connect a bus.
Program example

In this program, the value of the 5th bit of a device specified in \( s \) is written to the 3rd bit of a device specified in \( d \).

1) Function without EN/ENO(CPY_BIT_OF_INT)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_int1</th>
<th>s</th>
<th>K5</th>
<th>n1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

[ST]

\[
g_{int3} := \text{CPY\_BIT\_OF\_INT}(g_{int1}, K5, K3);
\]

2) Function with EN/ENO(CPY_BIT_OF_INT_E)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_bool2</th>
<th>s</th>
<th>g_int1</th>
<th>n1</th>
<th>K5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

[ST]

\[
g_{bool3} := \text{CPY\_BIT\_OF\_INT\_E}(g_{bool2}, g_{int1}, K5, K3, g_{int3});
\]
5.57 GET_BOOL_ADDR / Acquisition of start data

Outline
This function outputs the start data as bit data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_BOOL_ADDR</td>
<td>Label → GET_BOOL_ADDR</td>
<td>GET_BOOL_ADDR(s); Example: M10 = GET_BOOL_ADDR (Label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Input data</td>
<td>ANY</td>
</tr>
<tr>
<td>Output variable</td>
<td>Output data</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function outputs the start data of data specified in (s) as bit data to a device specified in (d).

<table>
<thead>
<tr>
<th>Input data type</th>
<th>Output data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit, Bit string</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Cautions
When the FLOAT (Single Precision) data to (s) is set from the programming tool, a rounding error may be generated.
Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example
In this program, MOV instruction is executed using the start bit of bit array data stored in a device specified in (s).

[Structured ladder/FBD]

```
  g_array  GET_BOOL_ADDR  MOV
         s               EN  ENO
         g_int1  g_int2    g_int2
```

[ST]

```
MOV(GET_BOOL_ADDR(g_array), g_int1, g_int2);
```
5.58 GET_INT_ADDR / Acquisition of start data

Outline
This function outputs the start data as word [signed] data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_INT_ADDR</td>
<td></td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label s</td>
<td>GET_INT_ADDR(s);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>D10 := GET_INT_ADDR</td>
</tr>
<tr>
<td></td>
<td>(Label);</td>
</tr>
</tbody>
</table>
```

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>s</td>
<td>ANY</td>
</tr>
<tr>
<td>Output variable</td>
<td>*1 (s)</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function outputs the start data of data specified in s as word [signed] data to a device specified in d.

<table>
<thead>
<tr>
<th>Input data type</th>
<th>Output data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word [signed], Double Word [signed], Word [unsigned]/Bit String [16-bit],</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>FLOAT (Single Precision), String, Time, Array of Word [signed],</td>
<td></td>
</tr>
<tr>
<td>Array of double word [signed], Array of word [unsigned]/bit string [16-bit],</td>
<td></td>
</tr>
<tr>
<td>Array of double word [unsigned]/bit string [32-bit], Array of real number,</td>
<td></td>
</tr>
<tr>
<td>Array of time type</td>
<td></td>
</tr>
</tbody>
</table>

Cautions
1) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to s is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example
In this program, low-order 16-bit data of double word [signed] data specified in s is written as word [signed] data to buffer memories.

[Structured ladder/FBD]

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>g_dint1</td>
<td>GET_INT_ADDR</td>
</tr>
<tr>
<td>s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>g_bool1</td>
</tr>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>k0</td>
<td>n1</td>
</tr>
<tr>
<td>k27</td>
<td>n2</td>
</tr>
<tr>
<td>k1</td>
<td>n3</td>
</tr>
</tbody>
</table>
```

[ST]
```
TO(g_bool1, GET_INT_ADDR(g_dint1), K0, K27, K1);
```
5.59 GET_WORD_ADDR / Acquisition of start data

Outline
This function outputs the start data as word [unsigned]/bit string [16-bit] data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_WORD_ADDR</td>
<td>GET_WORD_ADDR(s);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>D10 = GET_WORD_ADDR(...)</td>
</tr>
<tr>
<td></td>
<td>(Label);</td>
</tr>
</tbody>
</table>

1) Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>’in’</td>
<td>Input data</td>
</tr>
<tr>
<td></td>
<td>‘bout’</td>
<td>ANY</td>
</tr>
<tr>
<td>Output</td>
<td>‘bout’</td>
<td>Output data</td>
</tr>
<tr>
<td></td>
<td>‘bout’</td>
<td>Word [unsigned]/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit String [16-bit]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function outputs the start data of data specified in ‘bout’ as word [unsigned]/bit string [16-bit] data to a device specified in ‘bout’.

<table>
<thead>
<tr>
<th>Input data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word [signed], Double Word [signed], Word [unsigned]/Bit String [16-bit],</td>
</tr>
<tr>
<td>FLOAT (Single Precision), String, Time, Array of Word [signed],</td>
</tr>
<tr>
<td>Array of double word [signed], Array of word [unsigned]/bit string [16-bit],</td>
</tr>
<tr>
<td>Array of double word [unsigned]/bit string [32-bit], Array of real number,</td>
</tr>
<tr>
<td>Array of time type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word [unsigned]/Bit String [16-bit]</td>
</tr>
</tbody>
</table>

Cautions
1) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to ‘bout’ is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example
In this program, low-order 16-bit data of double word [signed] data specified in ‘bout’ is written as word [unsigned]/bit string [16-bit] data to buffer memories.

[Structured ladder/FBD]

```
    g_dint1  g_bool1
GET_WORD_ADDR s  EN
TO   g_bool1, GET_WORD_ADDR(g_dint1), K0, K27, K1;
```

[ST]

```
TO(g_bool1, GET_WORD_ADDR(g_dint1), K0, K27, K1);
```
6. Applied Functions (Standard Functions Of One Numeric Variable)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(_E)</td>
<td>Absolute value</td>
<td>Section 6.1</td>
</tr>
</tbody>
</table>
6.1 ABS(_E) / Absolute value

Outline

This function obtains the absolute value, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td>ABS</td>
<td>D0 _IN ABS *1 D10</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ABS_E</td>
<td>X000 _IN ABS_E *1 D10</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition Bit</td>
<td></td>
</tr>
<tr>
<td>_IN ( )</td>
<td>Data whose absolute value is to be obtained, or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status Bit</td>
<td></td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the absolute value of word [signed]/double word [signed]/float (single precision) data stored in a device specified in <s>, and outputs the operation result to a device specified in <d> using the data type of data stored in devices specified in a device specified in <s>.

   This function is expressed as follows when the input value is "A" and the output operation result is "B".
   
   B=|A|

2) When the data type stored in a device specified in <s> is word [signed] and the stored data is "-32768", this function outputs "-32768" to a device specified in <d>. (The maximum absolute value handled by this function is "32,767".)

   When the data type stored in a device specified in <s> is double word [signed] and the stored data is "-2147483648", this function outputs "-2147483648" to a device specified in <d>. (The maximum absolute value handled by this function is "2147483647".)
Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When the FLOAT (Single Precision) data to is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this program, the absolute value is obtained for word [signed] data stored in a device specified in , and the operation result is output to a device specified in using the data type same as the data stored in a device specified in .

1) Function without EN/ENO(ABS)

   [Structured ladder/FBD]
   
   ![Structured ladder/FBD diagram]

   [ST]
   
   g_int2 := ABS(g_int1);

2) Function with EN/ENO(ABS_E)

   [Structured ladder/FBD]
   
   ![Structured ladder/FBD diagram]

   [ST]
   
   g_bool3 := ABS_E(g_bool1, g_int1, g_int2);
# 7. Applied Functions (Standard Arithmetic Functions)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_E</td>
<td>Addition</td>
<td>Section 7.1</td>
</tr>
<tr>
<td>SUB_E</td>
<td>Subtraction</td>
<td>Section 7.2</td>
</tr>
<tr>
<td>MUL_E</td>
<td>Multiplication</td>
<td>Section 7.3</td>
</tr>
<tr>
<td>DIV_E</td>
<td>Division</td>
<td>Section 7.4</td>
</tr>
<tr>
<td>MOD(_E)</td>
<td>Modulus operation</td>
<td>Section 7.5</td>
</tr>
<tr>
<td>EXPT(_E)</td>
<td>Exponentiation</td>
<td>Section 7.6</td>
</tr>
<tr>
<td>MOVE(_E)</td>
<td>Move operation</td>
<td>Section 7.7</td>
</tr>
</tbody>
</table>
7.1 ADD_E / Addition

Outline

This function performs addition using two values (A + B = C), and outputs the operation result.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_E</td>
<td>X000 ADD_E(EN, IN, IN, Output_label);</td>
</tr>
<tr>
<td></td>
<td>EN ENO D0 IN D10 IN D20</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>IN (to 28)</td>
<td>Data for addition or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (to 28)</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function performs addition (s1 + s2 + ... + s28) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in s1 to s28, and outputs the operation result to a device specified in s1 to s28. Example: When the data type is word [signed]

```
  1234 + 5678 \rightarrow 6912
```

2) The number of pins for s1 can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction
Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

2) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. "TRUE" is output from ENO. However, note that the obtained operation result is not accurate in this case.

Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

<table>
<thead>
<tr>
<th>Device</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| M8020  | Zero   | ON : When the operation result is "0"  
OFF: When the operation result is any other than "0" |
| M8021  | Borrow| ON : When the operation result is less than "-32,768" (16-bit operation) or less than "-2,147,483,648" (32-bit operation)  
OFF: When the operation result is "-32,768" (16-bit operation) or more or "-2,147,483,648" (32-bit operation) or more |
| M8022  | Carry  | ON : When the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation)  
OFF: When the operation result is "32,767" (16-bit operation) or less or "2,147,483,647" (32-bit operation) or less |

3) When the FLOAT (Single Precision) data to \( S \) is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this program, addition is performed using double word [signed] data stored in devices specified in \( S \) and \( D \), and the operation result is output to a device specified in \( D \).

[Structured ladder/FBD]

```
ADD_E       g_bool3
  EN    ENO  g_bool1
  IN    IN   g_dint1
  g_dint2
```

```
g_bool3:=ADD_E(g_bool1,g_dint1,g_dint2,g_dint3);
```
7.2 SUB_E / Subtraction

Outline

This function performs subtraction using two values (A - B = C), and outputs the operation result.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB_E</td>
<td>X000 SUB_E EN _IN1 _IN2, ENO D0 D10</td>
</tr>
</tbody>
</table>

Example: When the data type is word [signed]

\[
\begin{align*}
S1 & = 12345 \\
S2 & = 6789 \\
D & = 5556
\end{align*}
\]

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Data to be subtracted or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>_IN2</td>
<td>Data for subtraction or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

*1. Output variable

Explanation of function and operation

This function performs subtraction (\( S1 - S2 \)) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in \( S1 \) and \( S2 \), and outputs the operation result to a device specified in \( d \) using the data type of data stored in devices specified in \( S1 \) and \( S2 \).

Example: When the data type is word [signed]

\[
\begin{align*}
S1 & = 12345 \\
S2 & = 6789 \\
D & = 5556
\end{align*}
\]
Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

2) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. "TRUE" is output from ENO. However, note that the obtained operation result is not accurate in this case.

Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

<table>
<thead>
<tr>
<th>Device</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M8020</td>
<td>Zero</td>
<td>ON : When the operation result is &quot;0&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OFF: When the operation result is any other than &quot;0&quot;</td>
</tr>
<tr>
<td>M8021</td>
<td>Borrow</td>
<td>ON : When the operation result is less than &quot;-32,768&quot; (16-bit operation) or less than &quot;-2,147,483,648&quot; (32-bit operation)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OFF: When the operation result is &quot;-32,768&quot; (16-bit operation) or more or &quot;-2,147,483,648&quot; (32-bit operation) or more</td>
</tr>
<tr>
<td>M8022</td>
<td>Carry</td>
<td>ON : When the operation result exceeds &quot;32,767&quot; (16-bit operation) or &quot;2,147,483,647&quot; (32-bit operation)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OFF: When the operation result is &quot;32,767&quot; (16-bit operation) or less or &quot;2,147,483,647&quot; (32-bit operation) or less</td>
</tr>
</tbody>
</table>

3) When the FLOAT (Single Precision) data to \( s \) is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this program, subtraction is performed using word [signed] data stored in devices specified in \( s1 \) and \( s2 \), and the operation result is output to a device specified in \( s \).

[Structured ladder/FBD]

```
SUB_E

[ST]
g_bool3:=SUB_E(g_bool1,g_int1,g_int2,g_int3);
```
7.3 MUL_E / Multiplication

Outline
This function performs multiplication using two or more values (A × B = C), and outputs the operation result.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL_E</td>
<td>X000</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN ( to )</td>
<td>Data for multiplication or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status Bit</td>
<td></td>
</tr>
<tr>
<td>_1*</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function performs multiplication (×⋯×) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in to , and outputs the operation result to a device specified in using the data type of data stored in devices specified in to .

   Example: When the data type is word [signed]

   \[
   \begin{align*}
   100 \times 15 & = 1500 \\
   \text{(Word } [\text{signed}] \text{ data) } & \times \text{(Word } [\text{signed}] \text{ data) } \\
   \end{align*}
   \]

2) The number of pins for can be changed in the range of 2 to 28.  

   → Refer to Section 3. Function Construction

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

   You can specify 32-bit counters directly, however, because they are 32-bit devices.

   Use global labels when specifying labels.

2) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. "TRUE" is output from ENO.

   However, note that the obtained operation result is not accurate in this case.

3) When the FLOAT (Single Precision) data to is set from the programming tool, a rounding error may be generated.

   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, multiplication is performed using double word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d.

[Structured ladder/FBD]

```
g_bool1
   g_dint1
   g_dint2
MUL_E
   EN
   ENO
   g_bool3
   g_dint3
```

[ST]

g_bool3:=MUL_E(g Bool1, g_dint1, g_dint2, g_dint3);
7.4 DIV_E / Division

Outline
This function performs division using two values \( A / B = C \ldots \) remainder, and outputs the quotient.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV_E</td>
<td>X000 DIV_E(EN, IN1, IN2, Output_label);</td>
</tr>
<tr>
<td></td>
<td>Example: DIV_E(X000, D0, D10, D20);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Data to be divided, or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>_IN2</td>
<td>Data for division (divisor), or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function performs division \((s1) / (s2)\) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in \(s1\) and \(s2\), and outputs the operation result to a device specified in \(d\) using the data type of data stored in devices specified in \(s1\) and \(s2\).

Example: When the data type is word [signed]

\[
\begin{array}{c|c|c}
5 & / & 2 \\
\hline
s1 & (Word [signed] data) & s2 & (Word [signed] data) \\
\hline
& & & (Quotient) \\
\end{array}
\]

Cautions
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to \(s\) is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Error
1) An operation error occurs when the divisor stored in a device specified in \(s2\) is "0", and the function is not executed.
2) An operation error occurs when the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation).
Program example

In this program, division is performed using double word [signed] data stored in devices specified in \( g_1 \) and \( g_2 \), and the operation result is output to a device specified in \( g_3 \) using the data type of data stored in devices specified in \( g_1 \) and \( g_2 \).

[Structured ladder/FBD]

```
  g_bool1
  g_dint1
  g_dint2
  DIV_E
  _IN1
  _IN2
  | EN
  | ENO
  | g_bool3
  | g_dint3
```

[ST]

```plaintext
  g_bool3 := DIV_E(g_bool1, g_dint1, g_dint2, g_dint3);
```
## 7.5 MOD(_E) / Modulus operation

### Outline
This function performs division using two values \( \frac{A}{B} = C \ldots \text{remainder} \), and outputs the remainder.

**1. Format**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD</td>
<td>Label1 _IN1 MOD _IN2; *1 Label3</td>
<td>IN1 MOD _IN2; *2 Example: Label3 := Label1 MOD Label2;</td>
</tr>
<tr>
<td>MOD_E</td>
<td>X000 MOD_E(EN, _IN1, _IN2, Output_label); *2 Example: MOD_E(X000,Label1, Label2,Label3);</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable  
*2. Refer to "Cautions".

**2. Set data**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1 ( )</td>
<td>Data to be divided, or word device which stores such data</td>
<td>ANY_INT</td>
</tr>
<tr>
<td>_IN2 ( )</td>
<td>Data for division (divisor), or word device which stores such data</td>
<td>ANY_INT</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Word device which will store the operation result</td>
<td>ANY_INT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

**Explanation of function and operation**
This function performs division \( \frac{\text{S1}}{\text{S2}} \) using word [signed]/double word [signed] data stored in devices specified in \( \text{S1} \) and \( \text{S2} \), and outputs the remainder to a device specified in \( \text{S1} \) using the data type of data stored in devices specified in \( \text{S1} \) and \( \text{S2} \).

Example: When the data type is word [signed]

![Example Diagram](https://example.com/example.png)

**Cautions**

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.  
   You can specify 32-bit counters directly, however, because they are 32-bit devices.  
   Use global labels when specifying labels.
3) Note that the "MOD" description method is different from other function description methods in the ST language.
Error

1) An operation error occurs when the divisor stored in a device specified in is "0", and the function is not executed.

2) An operation error occurs when the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation).

Program example

In this program, division is performed using double word [signed] data stored in devices specified in and , and the remainder is output to a device specified in using the data type of data stored in devices specified in and .

1) Function without EN/ENO(MOD)

[Structured ladder/FBD]

```
MOD
IN1
IN2
```

```
g_dint1=5678
g_dint2=1234
```

g_dint3:=g_dint1 MOD g_dint2;

2) Function with EN/ENO(MOD_E)

[Structured ladder/FBD]

```
MOD_E
EN
ENO
```

```
g_bool1
```

g_dint1

g_dint2

g_bool3

g_dint3

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
7.6 EXPT(_E) / Exponentiation

Outline
This function obtains raised result, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPT</td>
<td>Label1 ← EXPT In1, In2; Label2 = EXPT(Label1, D10);</td>
<td></td>
</tr>
<tr>
<td>EXPT_E</td>
<td>X000 ← EXPT_E (EN, In1, In2, Output_label); Example: EXPT_E(X000, Label1, D10, Label2);</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>In1 (s1)</td>
<td>Data to be raised, or word device which stores such data</td>
<td>FLOAT (Single Precision)</td>
</tr>
<tr>
<td>In2 (s2)</td>
<td>Power data, or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (d)</td>
<td>Word device which will store the operation result</td>
<td>FLOAT (Single Precision)</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function raises float (single precision) data stored in a device specified in (s1) (to the power of the value stored in a device specified in (s2)), and outputs the operation result to a device specified in (d).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
3) When the FLOAT (Single Precision) data to (s5) is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
**Error**

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the value stored in a device specified in \( \text{s1} \) is negative  
   (Error code: K6706)

2) When the value stored in a device specified in \( \text{s1} \) is "0"  
   (Error code: K6706)

3) When the operation result is outside the following range:  
   (Error code: K6706)  
   \[ 2^{-126} \leq |\text{Operation result}| < 2^{128} \]

**Program example**

In this program, the value stored in a device specified in \( \text{s1} \) is raised to the power of the value stored in a device specified in \( \text{s2} \), and the operation result is output to a device specified in \( \text{d} \) using the data type of data stored in a device specified in \( \text{s1} \).

1) Function without EN/ENO(EXPT)

   [Structured ladder/FBD]

   ```
   g_real1 \rightarrow \text{EXPT} \rightarrow g_real2 \\
   g_int1 \rightarrow \text{ln1} \rightarrow \text{ln2}
   ```

   [ST]
   
   \[ g\_real2:=\text{EXPT}(g\_real1,g\_int1); \]

2) Function with EN/ENO(EXPT_E)

   [Structured ladder/FBD]

   ```
   g\_bool1 \rightarrow \text{EXPT_E} \rightarrow g\_bool3 \\
   EN \rightarrow g\_real2 \\
   ENO \rightarrow g\_real1 \\
   g\_int1 \rightarrow \text{ln1} \rightarrow \text{ln2}
   ```

   [ST]
   
   \[ g\_bool3:=\text{EXPT}_E(g\_bool1,g\_real1,g\_int1,g\_real2); \]
### 7.7 MOVE(_E) / Move operation

#### Outline

This function transfers data stored in a device to another device.

1. **Format**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>D0 ＿IN ＿D10</td>
<td>MOVE(＿IN); D10＝MOVE(D0);</td>
</tr>
<tr>
<td>MOVE_E</td>
<td>X000 ＿IN ＿D10</td>
<td>MOVE_E(EN, ＿IN, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. **Set data**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>＿IN</td>
<td>Transfer source data, or word device which stores such data</td>
<td>ANY</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>＿D</td>
<td>Transfer destination word device</td>
<td>ANY</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

#### Explanation of function and operation

This function transfers data stored in a device specified in ( ) to a device specified in ( ).

![Diagram of data transfer]

**S** Word [signed] data  →  **D** Word [signed] data

#### Cautions

1) Use the function having "._E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

3) When the FLOAT (Single Precision) data to ( ) is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, word [signed] data stored in a device specified in \( \text{g_int1} \) is transferred to a device specified in \( \text{g_int2} \).

1) Function without EN/ENO(MOVE)

[Structured ladder/FBD]

\[
\begin{array}{c}
\text{g_int1} \quad \text{IN} \\
\quad \text{MOVE} \\
\quad \text{g_int2}
\end{array}
\]

[ST]
\[\text{g_int2} := \text{MOVE(g_int1)};\]

2) Function with EN/ENO(MOVE_E)

[Structured ladder/FBD]

\[
\begin{array}{c}
\text{g_bool1} \\
\quad \text{EN} \\
\quad \text{ENO}
\end{array}
\quad \text{MOVE_E} 
\quad \begin{array}{c}
\text{g_int1} \\
\quad \text{IN} \\
\quad \text{g_int2}
\end{array}
\quad \begin{array}{c}
\text{g_bool3} \\
\quad \text{EN} \\
\quad \text{ENO}
\end{array}
\]

[ST]
\[\text{g_bool3} := \text{MOVE_E(g_bool1, g_int1, g_int2)};\]
8. Applied Functions (Standard Bit Shift Functions)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHL(_E)</td>
<td>Left shift</td>
<td>Section 8.1</td>
</tr>
<tr>
<td>SHR(_E)</td>
<td>Right shift</td>
<td>Section 8.2</td>
</tr>
</tbody>
</table>
8.1 SHL(\_E) / Left shift

Outline

This function shifts data of specified bit length leftward by the specified number of bits.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHL ( (_E) )</td>
<td>( SHL(_E,_N,_IN,EN,ENO,D10) )</td>
</tr>
<tr>
<td>SHL ( (_E) )</td>
<td>( SHL(_E(EN,_IN,_N,Output_label)) )</td>
</tr>
</tbody>
</table>

1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( _IN )</td>
<td>Word device which stores data to be shifted leftward</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>( _N )</td>
<td>Number of shifted bits</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function shifts word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in a device specified in \( \_IN \) leftward by \( \_N \) bits, and outputs the obtained data to a device specified in \( \_IN \) using the data type of data stored in a device specified in \( \_N \).

Data is shifted leftward by \( \_N \) bits specified in \( \_N \).

Example: When word [unsigned]/bit string [16-bit] data is stored in a device specified in \( \_IN \), and "8" is specified in \( \_N \)

\[
\begin{align*}
\text{D10} &= \text{SHL(D0,K1)}; \\
\text{D10} &= \text{SHL_\_N,\_IN,\_IN,Output_label);}
\end{align*}
\]

2) \( \_N \) bits from the least significant bit become "0".
Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in is shifted leftward by "n" bits, and the obtained data is output to a device specified in using the data type of data stored in a device specified in .

1) Function without EN/ENO(SHL)

[Structured ladder/FBD]

```
<SHL>
  g_word1=16#F30F
  g_const_word1=16#0008
  g_word2=16#0F00
<IN>
<IN>

[ST]
  g_word2:=SHL(g_word1,g_const_word1);
```

2) Function with EN/ENO(SHL_E)

[Structured ladder/FBD]

```
<SHL_E>
  g_bool1
  g_word1
  g_const_word1
  g_word2
  g_bool3
  g_word2
<EN>
<ENO>

[ST]
  g_bool3:=SHL_E(g_bool1,g_word1,g_const_word1,g_word2);
```
8.2 SHR(_E) / Right shift

Outline
This function shifts data of specified bit length rightward by the specified number of bits.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHR</td>
<td><code>D0 _IN *1 D10</code></td>
<td>SHR(_IN,_K); Example: D10:= SHR(D0,K1);</td>
</tr>
<tr>
<td>SHR_E</td>
<td><code>X000 EN _ENO *1 D10</code></td>
<td>SHR_E(EN,_IN,N,Output_label); Example: SHR_E(X000,D0,K1,D10);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN (<em>s</em>)</td>
<td>Word device which stores data to be shifted rightward</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>_K,_N (<em>n</em>)</td>
<td>Number of shifted bits</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (<em>d</em>)</td>
<td>Word device which will store data obtained by shift</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
1) This function shifts word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in a device specified in (_s_) rightward by "n" bits, and outputs the obtained data to a device specified in (_d_), using the data type of data stored in a device specified in (_s_). Data is shifted rightward by "n" bits specified in (_n_).

Example: When word [unsigned]/bit string [16-bit] data is stored in a device specified in (_s_), and "8" is specified in (_n_)...

2) "n" bits from the most significant bit become "0".
Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in (1) is shifted rightward by "n" bits, and the obtained data is output to a device specified in (2) using the data type of data stored in a device specified in (3).

1) Function without EN/ENO(SHR)

   [Structured ladder/FBD]
   
   \[
   \begin{array}{c}
   \text{g\_word1=16#EEE} \\
   \text{g\_const\_word1=16#0006}
   \end{array}
   \]
   \[
   \begin{array}{c}
   \text{SHR} \\
   \text{IN} \\
   \text{K}
   \end{array}
   \]
   \[
   \begin{array}{c}
   \text{g\_word2=16#03BB}
   \end{array}
   \]

   [ST]
   \[
   \text{g\_word2:=SHR(g\_word1,g\_const\_word1);}
   \]

2) Function with EN/ENO(SHR_E)

   [Structured ladder/FBD]
   
   \[
   \begin{array}{c}
   \text{g\_bool1}
   \end{array}
   \]
   \[
   \begin{array}{c}
   \text{SHR\_E} \\
   \text{EN} \\
   \text{ENO}
   \end{array}
   \]
   \[
   \begin{array}{c}
   \text{g\_word1} \\
   \text{g\_const\_word1}
   \end{array}
   \]
   \[
   \begin{array}{c}
   \text{g\_word2}
   \end{array}
   \]

   [ST]
   \[
   \text{g\_bool3:=SHR\_E(g\_bool1,g\_word1,g\_const\_word1,g\_word2);}
   \]
9. Applied Functions (Standard Bitwise Boolean Functions)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND_E</td>
<td>Logical product</td>
<td>Section 9.1</td>
</tr>
<tr>
<td>OR_E</td>
<td>Logical sum</td>
<td>Section 9.2</td>
</tr>
<tr>
<td>XOR_E</td>
<td>Exclusive logical sum</td>
<td>Section 9.3</td>
</tr>
<tr>
<td>NOT(E)</td>
<td>Logical negation</td>
<td>Section 9.4</td>
</tr>
</tbody>
</table>
Outline
This function obtains the logical product of two or more bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND_E</td>
<td>X000 AND_E(EN, _IN, _IN, Output_label);</td>
<td>AND_E(EN, _IN, _IN, Output_label);</td>
</tr>
<tr>
<td></td>
<td>M0 _IN *1 M20</td>
<td>Example: AND_E(X000,M0,M10,M20);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input variable</strong></td>
<td><strong>EN</strong></td>
<td>Bit</td>
</tr>
<tr>
<td><strong>_IN (s1 to s28)</strong></td>
<td>Device used to obtain the logical product</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td><strong>Output variable</strong></td>
<td><strong>ENO</strong></td>
<td>Bit</td>
</tr>
<tr>
<td>*<strong>1 (d)</strong></td>
<td>Device which will store the operation result</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the logical product using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in s1 to s28, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 to s28.

Example: When the data type is word [unsigned]/bit string [16-bit]

```
s1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
Logical product
```

```
s2 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0
```

```
d 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0
```

2) The number of pins for s1 can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.
Program example

In this program, the logical product is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in $s1$ and $s2$, and the operation result is output to a device specified in $d$ using the data type of data stored in devices specified in $s1$ and $s2$.

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_bool1</th>
</tr>
</thead>
</table>
| g_word1
| g_word2 |

AND_E
<table>
<thead>
<tr>
<th>EN</th>
<th>g_bool3</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>g_word3</td>
</tr>
</tbody>
</table>
```

[ST]

```c
G_bool3 := AND_E(g_bool1, g_word1, g_word2, g_word3);
```
9.2 OR_E / Logical sum

Outline

This function obtains the logical sum of two or more bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR_E</td>
<td>X000</td>
<td>OR_E(EN, _IN, _IN, Output_label); OR_E(X000, M0, M10, M20);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN ( from 31 to 28)</td>
<td>Device used to obtain the logical sum</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( from 3)</td>
<td>Device which will store the operation result</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the logical sum using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in 31 to 28, and outputs the operation result to a device specified in 3 using the data type of data stored in devices specified in 31 to 28. Example: When the data type is word [unsigned]/bit string [16-bit]

<table>
<thead>
<tr>
<th>d1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100111100001111</td>
</tr>
</tbody>
</table>

Logical sum

<table>
<thead>
<tr>
<th>d2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000010010011010</td>
</tr>
</tbody>
</table>

2) The number of pins for 3 can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
Program example

In this program, the logical sum is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in \( s_1 \) and \( s_2 \), and the operation result is output to a device specified in \( d \) using the data type of data stored in devices specified in \( s_1 \) and \( s_2 \).

[Structured ladder/FBD]

[ST]
\[
g_{\text{bool}3} := \text{OR}_E(g_{\text{bool}1}, g_{\text{word}1}, g_{\text{word}2}, g_{\text{word}3});
\]
9.3 XOR_E / exclusive logical sum

Outline

This function obtains the exclusive logical sum of two or more bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR_E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN (to)</td>
<td>Device used to obtain the exclusive logical sum</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (to)</td>
<td>Device which will store the operation result</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the exclusive logical sum using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in (S) to (S), and outputs the operation result to a device specified in (S) using the data type of data stored in devices specified in (S) to (S).

Example: When the data type is word [unsigned]/bit string [16-bit]

```
(S) 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
```

Exclusive logical sum

```
(S) 0 0 0 1 1 1 1 1 1 0 0 0 0
```

```
(d) 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0
```

2) The number of pins for can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction
3) If there are 3 or more inputs, the exclusive logical sum is obtained using the "exclusive logical sum of" and and .
If there is one input, the exclusive logical sum is obtained using the "exclusive logical sum of" and . In this way, the exclusive logical sum is obtained the required number of times for all input labels .
Example: When the data type is bit

Cautions
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example
In this program, the exclusive logical sum is obtained using each bit of word [unsigned] and string [16-bit] data stored in devices specified in and , and the operation result is output to a device specified in using the data type of data stored in devices specified in and .

```
[Structured ladder/FBD]
  g_bool1
  g_word1
  g_word2
  XOR_E
  g Bool3
  g_word3

[ST]
  g_bool3:=XOR_E(g_bool1,g_word1,g_word2,g_word3);
```
9.4 NOT(_E) / logical negation

Outline

This function obtains the logical negation of bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>M0 NOT _IN *1 M10</td>
<td></td>
</tr>
<tr>
<td>NOT_E</td>
<td>X000 NOT_E _IN ENO *1 M10</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN ( )</td>
<td>Device used to obtain the logical negation</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Device which will store the operation result</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function obtains the logical negation using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in a device specified in ( ), and outputs the operation result to a device specified in ( ) using the data type of data stored in a device specified in ( ).

Example: When the data type is word [unsigned]/bit string [16-bit]

```
0110101100001111
1001010011110000
```

Logical negation

```
1001010011110000
0110101100001111
```

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.
Program example

In this program, the logical negation is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in a device specified in , and the operation result is output to a device specified in using the data type of data stored in a device specified in .

1) Function without EN/ENO(NOT)

[Structured ladder/FBD]

```
| g_word1 | NOT | g_word2 |
```

[ST]

```g_word2 := NOT(g_word1);```

2) Function with EN/ENO(NOT_E)

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_bool1</th>
<th>NOT_E</th>
<th>g_bool3</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_word1</td>
<td>EN</td>
<td>g_word2</td>
</tr>
</tbody>
</table>
```

[ST]

```g_bool3 := NOT_E(g_bool1, g_word1, g_word2);```
## 10. Applied Functions (Standard Selection Functions)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL(_E)</td>
<td>Selection</td>
<td>Section 10.1</td>
</tr>
<tr>
<td>MAXIMUM(_E)</td>
<td>Maximum selection</td>
<td>Section 10.2</td>
</tr>
<tr>
<td>MINIMUM(_E)</td>
<td>Minimum selection</td>
<td>Section 10.3</td>
</tr>
<tr>
<td>LIMITATION(_E)</td>
<td>Upper/Lower limit control</td>
<td>Section 10.4</td>
</tr>
<tr>
<td>MUX(_E)</td>
<td>Multiplexer</td>
<td>Section 10.5</td>
</tr>
</tbody>
</table>
### 10.1 SEL(_E) / Selection

Outline

This function selects either one between two data in accordance with the input condition, and outputs the selection result.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL</td>
<td></td>
<td>SEL(_G,_IN0,_IN1);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: D20:= SEL(M0,D0,D10);</td>
</tr>
<tr>
<td>SEL_E</td>
<td></td>
<td>SEL_E(EN,_G,_IN0,_IN1, _NO);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: SEL_E(X000,M0,D0,D10,D20);</td>
</tr>
</tbody>
</table>

*1. Output variable

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_G</td>
<td>Bit data used as the selection condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN0</td>
<td>Selectable data, or word device which stores such data</td>
<td>ANY</td>
</tr>
<tr>
<td>_IN1</td>
<td>Selectable data, or word device which stores such data</td>
<td>ANY</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>_NO</td>
<td>Word device which will store the selection result</td>
<td>ANY</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

#### Explanation of function and operation

This function outputs either one between the values stored in devices specified in _G_ and _IN_ in accordance with the value stored in a device specified in _IN0_ to a device specified in _IN1_ using the data type of data stored in a device specified in _G_ and _IN_.

When the value stored in a device specified in _IN0_ is "FALSE", this function outputs the value stored in a device specified in _IN1_ to a device specified in _IN0_.

When the value stored in a device specified in _IN0_ is "TRUE", this function outputs the value stored in a device specified in _IN1_ to a device specified in _IN0_.

Example: When the data type of input variables _G_ and _IN_ is word [signed]
Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) When the FLOAT (Single Precision) data to is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this program, either one between the values stored in devices specified in and is output in accordance with the value stored in a device specified in using the data type of data stored in devices specified in and .

1) Function without EN/ENO(SEL)

   [Structured ladder/FBD]

   \[
   \begin{array}{c}
   g\_bool1 \\
   g\_word1 \\
   g\_word2 \\
   \hline
   SEL \\
   \_G \\
   \_IN0 \\
   \_IN1 \\
   g\_word3 \\
   \end{array}
   \]

   \[
   \text{[ST]} \\
   g\_word3 := \text{SEL}(g\_bool1, g\_word1, g\_word2);
   \]

2) Function with EN/ENO(SEL_E)

   [Structured ladder/FBD]

   \[
   \begin{array}{c}
   g\_bool1 \\
   g\_bool2 \\
   g\_word1 \\
   g\_word2 \\
   \hline
   \text{SEL}_E \\
   \text{EN} \\
   \text{ENO} \\
   \_G \\
   \_IN0 \\
   \_IN1 \\
   g\_word3 \\
   g\_bool3 \\
   \end{array}
   \]

   \[
   \text{[ST]} \\
   g\_bool3 := \text{SEL}_E(g\_bool1, g\_bool2, g\_word1, g\_word2, g\_word3);
   \]
Outline
This function searches the maximum value among data, and outputs the maximum value.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXIMUM</td>
<td>MAXIMUM(_IN,_IN)</td>
<td>MAXIMUM(_IN,_IN)</td>
</tr>
<tr>
<td>MAXIMUM_E</td>
<td>MAXIMUM_E(EN,_IN,_IN, Output_label);</td>
<td>MAXIMUM_E(EN,_IN,_IN, Output_label);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN (s1 to s28)</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>_1 (s1 to s28)</td>
<td>Word device which will store the maximum value</td>
<td>ANY_SIMPLE</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function outputs the maximum value among ANY_SIMPLE type data stored in devices specified in s1 to s28 to a device specified in s1 using the data type of data stored in devices specified in s1 to s28.

Example: When the data type is word [signed]

![Example Diagram]

2) The number of pins for s can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.

3) When the FLOAT (Single Precision) data to s is set from the programming tool, a rounding error may be generated.

Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, the maximum value among word [signed] data stored in devices specified in \( \text{g1} \) and \( \text{g2} \) is output to a device specified in \( \text{g3} \) using the data type of data stored in devices specified in \( \text{g1} \) and \( \text{g2} \).

1) Function without EN/ENO(MAXIMUM)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{int}1} = 5678 \\
g_{\text{int}2} = 1234 \\
g_{\text{int}3} = 5678
\end{array}
\]

\[
\text{MAXIMUM}
\]

\[
\begin{array}{c}
g_{\text{int}3} := \text{MAXIMUM}(g_{\text{int}1}, g_{\text{int}2});
\end{array}
\]

2) Function with EN/ENO(MAXIMUM_E)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool}1} \\
g_{\text{int}1} \\
g_{\text{int}2} \\
g_{\text{bool}3} \\
g_{\text{int}3}
\end{array}
\]

\[
\text{MAXIMUM_E}
\]

\[
\begin{array}{c}
g_{\text{bool}3} := \text{MAXIMUM_E}(g_{\text{bool}1}, g_{\text{int}1}, g_{\text{int}2}, g_{\text{int}3});
\end{array}
\]
10.3 MINIMUM(_E) / Minimum selection

Outline
This function searches the minimum value among data, and outputs the minimum value.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINIMUM</td>
<td>MINIMUM(_IN,_IN)</td>
</tr>
<tr>
<td>MINIMUM_E</td>
<td>MINIMUM_E(EN,_IN,_IN,</td>
</tr>
<tr>
<td></td>
<td>Output_label)</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td>Word device which will store the minimum value</td>
<td>ANY_SIMPLE</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function outputs the minimum value among ANY_SIMPLE type data stored in devices specified in  to  to a device specified in  using the data type of data stored in devices specified in  to .

Example: When the data type is word [signed]

```
WORD [signed] data

1234
MINIMUM
1234

5678
MINIMUM
```

2) The number of pins for  can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
3) When the FLOAT (Single Precision) data to  is set from the programming tool, a rounding error may be generated.

Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
**Program example**

In this program, the minimum value among word [signed] data stored in devices specified in \( s1 \) and \( s2 \) is output to a device specified in \( d \) using the data type of data stored in devices specified in \( s1 \) and \( s2 \).

1) Function without EN/ENO(MINIMUM)

[Structured ladder/FBD]

\[
\begin{array}{c}
g\text{\_int1}=5678 \\
g\text{\_int2}=1234 \\
\text{MINIMUM} \\
g\text{\_int3}=1234
\end{array}
\]

[ST]
\[g\text{\_int3}:={\text{MINIMUM}}(g\text{\_int1},g\text{\_int2});\]

2) Function with EN/ENO(MINIMUM\text{\_E})

[Structured ladder/FBD]

\[
\begin{array}{c}
g\text{\_bool1} \\
g\text{\_int1} \\
g\text{\_int2} \\
\text{MINIMUM\text{\_E}} \\
g\text{\_bool3} \\
g\text{\_int3}
\end{array}
\]

[ST]
\[g\text{\_bool3}:={\text{MINIMUM\_E}}(g\text{\_bool1},g\text{\_int1},g\text{\_int2},g\text{\_int3});\]
10.4 LIMITATION(_E) / Upper/Lower limit control

Outline
This function judges whether data is located within the range between the upper limit value and the lower limit value.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIMITATION</td>
<td>D0 MN *1 D10 IN D20 MX</td>
<td>LIMITATION(MN, IN, MX); Example: D30:= LIMITATION(D0, D10, D20);</td>
</tr>
<tr>
<td>LIMITATION_E</td>
<td>X000 EN IN ENO D0 MN *1 D10 IN D20 MX</td>
<td>LIMITATION_E(EN, MN, IN, MX, Output_label); Example: LIMITATION_E(X000, D0, D10, D20, D30);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>MN ( )</td>
<td>Lower limit data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>IN ( )</td>
<td>Input data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>MX ( )</td>
<td>Upper limit data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Word device which will store the output data</td>
<td>ANY_SIMPLE</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function outputs data whose type is same as the data stored in devices specified in ( ) and ( ) to a device specified in ( ) in accordance with ANY_SIMPLE type data stored in devices specified in ( ), ( ) and ( ).

1) In the case of "Contents of a device specified in ( ) > Contents of a device specified in ( )", this function outputs the contents of a device specified in ( ) to a device specified in ( ).
2) In the case of "Contents of a device specified in ( ) < Contents of a device specified in ( )", this function outputs the contents of the contents of a device specified in ( ) to a device specified in ( ).
3) In the case of "Contents of a device specified in ( ) ≤ Contents of a device specified in ( ) ≤ Contents of a device specified in ( )", this function outputs the contents of a device specified in ( ) to a device specified in ( ).

Example: When the data type is word [signed]
Cautions

1) Use the function having ".E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) When the FLOAT (Single Precision) data to is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Error

An operation error occurs when this function is executed in the following setting status. The error flag M8067 turns ON, and D8067 stores the error code K6706.

Contents of a device specified in (Lower limit data) > Contents of a device specified in (Upper limit data)

Program example

In this program, data whose type is same as the data stored in devices specified in , , and is output to a device specified in in accordance with ANY_SIMPLE type data stored in devices specified in , , and .

1) Function without EN/ENO(LIMITATION)

[Structured ladder/FBD]

```plaintext
  g_int1=500
  g_int2=1300
  g_int3=5000

  LIMITATION
    _MN
    _IN
    _MX

  g_int4=1300

  [ST]
  g_int4:=LIMITATION(g_int1,g_int2,g_int3);
```

2) Function with EN/ENO(LIMITATION_E)

[Structured ladder/FBD]

```plaintext
  g_bool1
  g_int1
  g_int2
  g_int3

  LIMITATION_E
    EN
    MN
    _IN
    _MX

  g_bool3
  g_int4

  [ST]
  g_bool3:=LIMITATION_E(g_bool1,g_int1,g_int2,g_int3,g_int4);
```
### Outline

This function selects data, and outputs the selected data.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUX</td>
<td></td>
</tr>
<tr>
<td>MUX_E</td>
<td></td>
</tr>
</tbody>
</table>

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_K (<em>N</em>)</td>
<td>Selection data, or word device which stores such data</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>_IN (<em>N</em>) to (<em>N</em>)</td>
<td>Selectable data, or word device which stores such data</td>
<td>ANY</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Word device which will store the selected data</td>
<td>ANY</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

### Explanation of function and operation

1) This function outputs either one among values stored in devices specified in (_N_) to (_N_) to a device specified in (_N_) in accordance with the value specified in (_N_) using the data type of data stored in devices specified in (_N_) to (_N_).
   a) When the value specified in (_N_) is "1", this function outputs the value stored in a device specified in (_N_) to a device specified in (_N_).
   b) When the value specified in (_N_) is "n", this function outputs the value stored in a device specified in (_N_) to a device specified in (_N_).

Example: When the data type is word [signed]

![Diagram of function operation]

2) When a value input to (_N_) is outside the pin number range for (_N_) to (_N_), this function outputs an indefinite value to a device specified in (_N_).
   (An operation error does not occur. "MUX_E" outputs "FALSE" from ENO.)

3) The number of pins for (_N_) can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction
Cautions

1) Use the function having ",_E\" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

3) When the FLOAT (Single Precision) data to \( s \) is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this example, either one among values stored in devices specified in \( s_1 \) and \( s_2 \) is output to a device specified in \( s \) in accordance with the value specified in \( n \) using the data type of data stored in devices specified in \( s_1 \) and \( s_2 \).

1) Function without EN/ENO(MUL)

   \[
   \text{MUX} \quad \text{g_int1}=2 \quad \text{IN} \quad \text{g_int2}=1234 \quad \text{IN} \quad \text{g_int3}=5678 \quad \text{IN} \quad \text{g_int4}=5678
   \]

   \[
   \text{g_int4}:=\text{MUX(g_int1,g_int2,g_int3)};
   \]

2) Function with EN/ENO(MUL_E)

   \[
   \text{MUX_E} \quad \text{g_bool1} \quad \text{EN} \quad \text{g_int1}=2 \quad \text{K} \quad \text{EN} \quad \text{g_int2}=1234 \quad \text{IN} \quad \text{EN} \quad \text{g_int3}=5678 \quad \text{IN} \quad \text{EN} \quad \text{g_int4}
   \]

   \[
   \text{g_bool3}:=\text{MUX_E(g_bool1,g_int1,g_int2,g_int3,g_int4)};
   \]
# 11. Applied Functions (Standard Comparison Functions)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT_E</td>
<td>Comparison</td>
<td>Section 11.1</td>
</tr>
<tr>
<td>GE_E</td>
<td>Comparison</td>
<td>Section 11.2</td>
</tr>
<tr>
<td>EQ_E</td>
<td>Comparison</td>
<td>Section 11.3</td>
</tr>
<tr>
<td>LE_E</td>
<td>Comparison</td>
<td>Section 11.4</td>
</tr>
<tr>
<td>LT_E</td>
<td>Comparison</td>
<td>Section 11.5</td>
</tr>
<tr>
<td>NE_E</td>
<td>Comparison</td>
<td>Section 11.6</td>
</tr>
</tbody>
</table>
11.1 GT_E / Comparison

Outline
This function compares data with regard to "> (larger)".

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT_E</td>
<td>X000 GT_E D0 D10 M0</td>
<td>GT_E(EN,_IN,_IN,Output_label); Example: GT_E(X000,D0,D10,M0);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN (s1 to s28)</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (s3)</td>
<td>Device which will store the comparison result</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in s1 to s28, and outputs the operation result expressed as the bit type data to a device specified in s3.

This function executes comparison [s1 > s2] & [s2 > s3] & … & [sn-1 > sn].

a) This function outputs "TRUE" when all comparison results are "s_n-1 > sn".
b) This function outputs "FALSE" when any comparison result is "s_n-1 ≤ sn".

2) The number of pins for s1 can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects.

Use labels when handling 32-bit data.

You can specify 32-bit counters directly, however, because they are 32-bit devices.

Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to s1 is set from the programming tool, a rounding error may be generated.

Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
**Program example**

In this program, the contents of devices specified in \( s1 \) and \( s2 \) are compared, and the operation result is output to a device specified in \( d \).


```plaintext
[Structured ladder/FBD]

```
g_bool1

```
g_bool3:=GT_E(g_bool1,g_int1,g_int2,g_bool2);
```

[ST]

```plaintext
g_bool3:=GT_E(g_bool1,g_int1,g_int2,g_bool2);
```
11.2 GE_E / Comparison

Outline

This function compares data with regard to "≥ (larger or equal)".

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE_E</td>
<td>X000 EN IN ENO</td>
<td>GE_E(EN, IN, IN, Output_label); GE_E(X000, D0, D10, M0);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>_IN (≥1 to ≤256) Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>*1 (≥1) Device which will store the comparison result</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in _IN_ (≥1 to ≤28), and outputs the operation result expressed as the bit type data to a device specified in _OUT_.

   This function executes comparison \([S1 ≥ S2] & [S2 ≥ S3] & \ldots & [S_{n-1} ≥ S_n]\).

   a) This function outputs "TRUE" when all comparison results are "≥ S_n".

   b) This function outputs "FALSE" when any comparison result is "< S_n".

2) The number of pins for _IN_ can be changed in the range of 2 to 28.

   → Refer to Section 3. Function Construction

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

   You can specify 32-bit counters directly, however, because they are 32-bit devices.

   Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to _IN_ is set from the programming tool, a rounding error may be generated.

   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, the contents of devices specified in $s1$ and $s2$ are compared, and the operation result is output to a device specified in $d$.

[Structured ladder/FBD]

```
  g_bool1
  g_int1
  g_int2
  GE_E
    EN
    IN
    IN
    ENO
    g_bool3
    g_bool2
```

[ST]

```
g_bool3 := GE_E(g_bool1, g_int1, g_int2, g_bool2);
```
11.3 EQ_E / Comparison

Outline

This function compares data with regard to "= (equal)".

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ_E</td>
<td>EQ_E(EN, _IN, _IN, Output_label);</td>
<td>EQ_E(X000, D0, D10, M0);</td>
</tr>
</tbody>
</table>

1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Device which will store the comparison result</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in (s1) to (s28), and outputs the operation result expressed as the bit type data to a device specified in (M0).

2) The number of pins for (s) can be changed in the range of 2 to 28.

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

2) When the FLOAT (Single Precision) data to (s) is set from the programming tool, a rounding error may be generated.

Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, the contents of devices specified in \( s_1 \) and \( s_2 \) are compared, and the operation result is output to a device specified in \( d \).

```
g_bool3 := EQ_E(g_bool1, g_int1, g_int2, g_bool2);
```
11.4 LE_E / Comparison

Outline
This function compares data with regard to "≤ (smaller or equal)".

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE_E</td>
<td>LE_E(EN, _IN, _IN, Output_label);</td>
<td>LE_E(X000,D0,D10,M0);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN (  to )</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 (  )</td>
<td>Device which will store the comparison result</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in (  ) to (  ), and outputs the operation result expressed as the bit type data to a device specified in (  ).

   This function executes comparison [ (  ) ≤ (  ) ] & [ (  ) ≤ (  ) ] & ... & [ (  ) ≤ (  ) ].

   a) This function outputs "TRUE" when all comparison results are "≤ (  )".

   b) This function outputs "FALSE" when any comparison result is "> (  )".

2) The number of pins for (  ) can be changed in the range of 2 to 28.

   → Refer to Section 3. Function Construction

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

   You can specify 32-bit counters directly, however, because they are 32-bit devices.

   Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to (  ) is set from the programming tool, a rounding error may be generated.

   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, the contents of devices specified in $s_1$ and $s_2$ are compared, and the operation result is output to a device specified in $d$.

[Structured ladder/FBD]

```
g_bool1
EN   g_bool3
IN   g_bool2
IN   g_int1
IN   g_int2
```

[ST]

```g_bool3:=LE_E(g_bool1,g_int1,g_int2,g_bool2);```
11.5 LT_E / Comparison

Outline
This function compares data with regard to "< (smaller)".

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT_E</td>
<td>LT_E(EN, _IN, _IN, Output_label); Example: LT_E(X000, D0, D10, M0);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>_IN (31 to 28)</td>
<td>Compared data, or word device which stores such data</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in (31 to 28), and outputs the operation result expressed as the bit type data to a device specified in (4).
   This function executes comparison: \[ \text{(31)} < \text{(32)} \] & \[ \text{(32)} < \text{(33)} \] & ... & \[ \text{(n-1)} < \text{(n)} \].
   a) This function outputs "TRUE" when all comparison results are "\( \text{(n-1)} < \text{(n)} \)".
   b) This function outputs "FALSE" when any comparison result is "\( \text{(n-1)} \geq \text{(n)} \)".

2) The number of pins for \( \text{LT_E} \) can be changed in the range of 2 to 28.
   → Refer to Section 3. Function Construction

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to \( \text{LT_E} \) is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, the contents of devices specified in \( \text{s1} \) and \( \text{s2} \) are compared, and the operation result is output to a device specified in \( \text{d} \).

[Structured ladder/FBD]

\[
\text{LT}_E \quad \text{EN} \quad \text{ENO}
\]
\[
g\_\text{bool1} \quad \text{g\_int1} \quad \text{g\_int2} \quad \text{g\_bool3} \quad \text{g\_bool2}
\]

[ST]

\[
g\_\text{bool3} := \text{LT}_E(\text{g\_bool1}, \text{g\_int1}, \text{g\_int2}, \text{g\_bool2});
\]
11.6 NE_E / Comparison

Outline

This function compares data with regard to "≠ (unequal)".

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE_E</td>
<td>X000 NE_E EN ENO D0 _IN1 _IN2 D10 _IN2 M0</td>
</tr>
<tr>
<td></td>
<td>NE_E(EN, _IN1, _IN2, Output_label); Example: NE_E(X000, D0, D10, M0);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>_IN2</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Device which will store the comparison result</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function compares the contents of devices specified in (S1) and (S2), and outputs the operation result expressed as the bit type data to a device specified in (S).

This function executes comparison \[ S1 \neq S2 \].

a) This function outputs "TRUE" when in the case of \( S1 \neq S2 \)

b) This function outputs "FALSE" when in the case of \( S1 = S2 \)

Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.

   You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to (S) is set from the programming tool, a rounding error may be generated.

   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
Program example

In this program, the contents of devices specified in \( s_1 \) and \( s_2 \) are compared, and the operation result is output to a device specified in \( d \).

[Structured ladder/FBD]

\[
g_{\text{bool1}}
\]

\[
g_{\text{int1}}
\]

\[
g_{\text{int2}}
\]

\[
\text{NE}_E
\]

\[
\text{ENO}
\]

\[
g_{\text{bool3}}
\]

\[
g_{\text{bool2}}
\]

[ST]

\[
g_{\text{bool3}} := \text{NE}_E(g_{\text{bool1}}, g_{\text{int1}}, g_{\text{int2}}, g_{\text{bool2}});
\]
## 12. Applied Functions (Standard Character String Functions)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID(_E)</td>
<td>Extract mid string</td>
<td>Section 12.1</td>
</tr>
<tr>
<td>CONCAT(_E)</td>
<td>String concatenation</td>
<td>Section 12.2</td>
</tr>
<tr>
<td>INSERT(_E)</td>
<td>String insertion</td>
<td>Section 12.3</td>
</tr>
<tr>
<td>DELETE(_E)</td>
<td>String deletion</td>
<td>Section 12.4</td>
</tr>
<tr>
<td>REPLACE(_E)</td>
<td>String replacement</td>
<td>Section 12.5</td>
</tr>
<tr>
<td>FIND(_E)</td>
<td>Searches a character string</td>
<td>Section 12.6</td>
</tr>
</tbody>
</table>
12.1 MID(_E) / Extract mid string

Outline
This function obtains a character string from a specified position.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>MID(_IN,_L ,_P);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Label2 = MID(Label1,D10,D20);</td>
<td></td>
</tr>
<tr>
<td>MID_E</td>
<td>MID_E(EN,_IN,_L ,_P,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output_label);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MID_E(X000,Label1,D10,D20, Label2);</td>
<td></td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN</td>
<td>Head word device which stores a character string</td>
<td>String</td>
</tr>
<tr>
<td>_L</td>
<td>Word device which stores the number of characters to be obtained</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>_P</td>
<td>Word device which stores the head character position of a character string to be obtained</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>_*1</td>
<td>Head word device which will store the obtained character string</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.
12.1 MID(_E) / Extract mid string

Explanation of function and operation

1) This function extracts specified number of characters from an arbitrary position of a character string stored in devices specified in \( n_1 \), and outputs the obtained data to devices specified in \( n_2 \). The value specified in \( n_1 \) specifies the number of characters to be extracted.
The value specified in \( n_2 \) specifies the head character position of characters to be extracted.
Example: When "5" is specified in \( n_1 \) and \( n_2 \)

\[
\begin{align*}
&\text{\"ABCDEF12345\"} \quad \Rightarrow \quad \text{\"EF123\"} \\
&\text{High-order byte} \quad \text{Low-order byte} \\
&1\text{st word} \quad \begin{array}{c}
42H(B) \\
44H(D) \\
46H(F) \\
32H(2) \\
34H(4) \\
00H
\end{array} \\
&2\text{nd word} \quad \begin{array}{c}
41H(A) \\
43H(C) \\
45H(E) \\
31H(1) \\
33H(3) \\
35H(5)
\end{array} \\
&3\text{rd word} \quad \begin{array}{c}
\text{Head character position of a character string to be extracted (n2)} \\
= 5\text{th character}
\end{array} \\
&4\text{th word} \quad \begin{array}{c}
46H(F) \\
32H(2) \\
00H
\end{array} \\
&5\text{th word} \quad \begin{array}{c}
45H(E) \\
31H(1) \\
33H(3)
\end{array} \\
&6\text{th word} \quad \begin{array}{c}
00H \\
35H(5)
\end{array}
\end{align*}
\]

Number of characters to be extracted \( n_1 = 5 \)

2) A character string (data) stored in devices specified in \( s \) indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

3) When the number of characters to be extracted specified in \( n_1 \) is "0", this function does not execute processing.

4) When the number of characters to be extracted specified in \( n_1 \) is "-1", this function outputs the final character of a character string specified in \( s \) to devices specified in \( d \).

Cautions

1) Use the function having ",_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When "00H" is not set in the corresponding device range after the device specified in \( s \) (Error code: K6706)

2) When the head character position specified in \( n_2 \) exceeds the number of characters of a character string stored in devices specified in \( s \) (Error code: K6706)

3) When the number of characters specified in \( n_1 \) exceeds the range of devices specified in \( d \) (Error code: K6706)

4) When the number of devices after the device number specified in \( d \) is smaller than the number of devices required for storing an extracted character string (In this case, "00H" cannot be stored after all character strings and the final character.) (Error code: K6706)

5) When the value specified in \( n_2 \) is negative (Error code: K6706)

6) When the value specified in \( n_1 \) is ",-2" or less (Error code: K6706)

7) When the value specified in \( n_1 \) exceeds the number of characters of a character string stored in devices specified in \( s \) (Error code: K6706)
Program example

In this program, specified number of characters are extracted from an arbitrary position of a character string stored in devices specified in \( g_{\text{string}} \), and the obtained data is output to devices specified in \( g_{\text{string}} \).

1) Function without EN/ENO(MID)

   [Structured ladder/FBD]

   \[
   \text{g\_string1} = \text{"ABCDEF12345"} \quad \text{g\_int1} = 5 \quad \text{g\_int2} = 5
   \]

   \[
   \text{g\_string2} = \text{"EF123"}
   \]

   [ST]

   \[
   \text{g\_string2} := \text{MID}(\text{g\_string1}, \text{g\_int1}, \text{g\_int2});
   \]

2) Function with EN/ENO(MID_E)

   [Structured ladder/FBD]

   \[
   \text{g\_bool1}
   \]

   \[
   \text{g\_string1} \quad \text{g\_int1} \quad \text{g\_int2}
   \]

   \[
   \text{g\_strin} \text{g\_bool3}
   \]

   [ST]

   \[
   \text{g\_bool3} := \text{MID\_E}(\text{g\_bool1}, \text{g\_string1}, \text{g\_int1}, \text{g\_int2}, \text{g\_string2});
   \]
### 12.2 CONCAT(_E) / String concatenation

**Outline**

This function connects character strings.

**1. Format**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCAT</td>
<td>Label1 _IN Label2 _IN *1 Label3</td>
<td>CONCAT(_IN,_IN); Example: Label3= CONCAT(Label1,Label2);</td>
</tr>
<tr>
<td>CONCAT_E</td>
<td>X000 EN _IN _IN ENO *1 Label3</td>
<td>CONCAT_E(EN,_IN,_IN, Output_label); Example: CONCAT_E(X000,Label1,Label2, Label3);</td>
</tr>
</tbody>
</table>

*1. Output variable

**2. Set data**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN ( to )</td>
<td>Head word device which stores the data (character string) to be connected, or directly specified character string</td>
<td>String</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1 ( )</td>
<td>Head word device which will store the connected data (character string)</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

**Explanation of function and operation**

1) This function connects a character string stored in devices specified in after a character string stored in devices specified in , and outputs the character string obtained by connection to devices specified in .

When connecting a character string stored in devices specified in after a character string stored in devices specified in , this function ignores "00H" which indicates the end of a character string stored in devices specified in .

After two character strings are connected, "00H" is automatically added at the end.

2) A character string (data) stored in devices specified in indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

3) For direct specification, up to 32 characters can be specified (input).

When word devices are specified in , this restriction (up to 32 characters) is not applicable.
4) When both a character string stored in devices specified in begins with "00H" (when character = 0), this function stores "0000H" in devices specified in.

5) The number of pins for can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data. Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the number of devices after the device number specified in is smaller than the number of devices required for storing the character string obtained by connection
   (In this case, "00H" cannot be stored after all character strings and final character.)
   (Error code: K6706)

2) When devices which store character strings specified in overlap device numbers specified in which will store the character string obtained by connection
   (Error code: K6706)

3) When "00H" does not exist in the corresponding device range after devices specified in
   (Error code: K6706)

Program example

In this program, a character string stored in devices specified in is connected after a character string stored in devices specified in , and the character string obtained by connection is output to devices specified in.

1) Function without EN/ENO(CONCAT)

[Structured ladder/FBD]

```
CONCAT
IN  g_string1="ABCDEF"
IN  g_string2="12345"
```

```
[ST]
g_string3:=CONCAT(g_string1,g_string2);
```

2) Function with EN/ENO(CONCAT_E)

[Structured ladder/FBD]

```
CONCAT_E
EN  g_bool1
EN  g_string1
EN  g_string2
ENO  g_bool3
```

```
[ST]
g_bool3:=CONCAT_E(g_bool1,g_string1,g_string2,g_string3);
```
12.3 INSERT(_E) / String insertion

Outline

This function inserts a character string.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>INSERT(_IN1,_IN2,_P);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Label3:=INSERT(Label1,Label2,D20);</td>
</tr>
<tr>
<td>INSERT_E</td>
<td>INSERT_E(EN,_IN1,_IN2,_P,</td>
</tr>
<tr>
<td></td>
<td>Output_label);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>INSERT_E(X000,Label1,Label2,</td>
</tr>
<tr>
<td></td>
<td>D20,Label3);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Head word device which stores a character string to get insertion</td>
<td>String</td>
</tr>
<tr>
<td>_IN2</td>
<td>Head word device which stores a character string to be inserted</td>
<td>String</td>
</tr>
<tr>
<td>_P</td>
<td>Word device which stores a character position to get insertion</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store a character string obtained by insertion</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.
Explanation of function and operation

1) This function inserts a character string stored in devices specified in \(s_2\) into an arbitrary position (counted from the head) of a character string stored in devices specified in \(s_1\), and outputs the character string obtained by insertion to devices specified in \(s_2\).

The value specified in \(n\) specifies the position from which the character string stored in devices specified in \(s_2\) is inserted.

After inserting a character string stored in devices specified in \(s_2\) into a character string stored in devices specified in \(s_1\), this function ignores "00H" which indicates the end of a character string stored in devices specified in \(s_2\).

Example: When "4" is specified in \(n\)

```
Input value to \(s_1\) = "ABCDE"

Output value "ABC123456DE"
```

2) A character string (data) stored in devices specified in \(s_3\) indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

Cautions

1) Use the function having " _E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data. Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the number of devices after the device number specified in \(d\) is smaller than the number of devices required for storing the output data obtained by insertion (Error code: K6706)

2) When devices which store character strings specified in \(s_1\) and \(s_2\) overlap device numbers specified in \(d\) which will store the character string obtained by connection (Error code: K6706)

3) When "00H" does not exist in the corresponding device range after devices specified in \(s_1\) and \(s_2\) (Error code: K6706)

4) When the number of characters of a character string stored in devices specified in \(s_2\) is 32768 or more (Error code: K6706)

5) When the value specified in \(n\) is negative (Error code: K6706)
Program example

In this program, a character string stored in devices specified in \( s_2 \) is inserted into an arbitrary position (counted from the head) of a character string stored in devices specified in \( s_1 \), and the character string obtained by insertion is output to devices specified in \( d \).

1) Function without EN/ENO(INSERT)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_string}1 &= \text{"ABCDEF"} \\
\text{g\_string}2 &= \text{"12345"} \\
\text{g\_int}1 &= 3 \\
\hline
\text{INSERT} & \quad \text{g\_string}3 &= \text{"AB12345CDEF"} \\
\text{IN}1 & \quad \text{IN}2 \\
\text{P} &
\end{align*}
\]

\[\text{[ST]}\]
\[\text{g\_string}3 := \text{INSERT(g\_string}1, g\_string2, g\_int1);\]

2) Function with EN/ENO(INSERT_E)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_bool}1 & \\
\text{g\_string}1 & \\
\text{g\_string}2 & \\
\text{g\_int}1 & \\
\hline
\text{INSERT_E} & \quad \text{g\_bool}3 \\
\text{EN} & \quad \text{ENO} \\
\text{IN}1 & \quad \text{IN}2 \\
\text{P} &
\end{align*}
\]

\[\text{[ST]}\]
\[\text{g\_bool}3 := \text{INSERT_E(g\_bool}1, g\_string}1, g\_string2, g\_int1, g\_string}3);\]
### 12.4 DELETE(_E) / String deletion

**Outline**

This function deletes a character string.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DELETE</strong></td>
<td>Label1 _IN _L _P; Label2</td>
</tr>
<tr>
<td><strong>DELETE_E</strong></td>
<td>X000 _IN _L _P; Label2</td>
</tr>
</tbody>
</table>

Example:

```
DELETE(Label1,D10,D20);
```

Example:

```
DELETE_E(X000, Label1, D10, D20, Label2);
```

\*1. Output variable

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN</td>
<td>Head word device which stores a character string to get deletion</td>
<td>String</td>
</tr>
<tr>
<td>_L</td>
<td>Number of characters to be deleted</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>_P</td>
<td>Head position to get deletion</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store a character string remaining after deletion</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

**Explanation of function and operation**

1) This function deletes specified number of characters from an arbitrary position of a character string stored in devices specified in ( ), and outputs the character string remaining after deletion to devices specified in ( ).

The value specified in ( ) specifies the number of characters to be deleted.

The value specified in ( ) specifies the position from which specified number of characters are deleted.

Example: When "5" is specified in ( ) and ( )

```
"ABCDEF12345"  "ABCD45"
```

```
1st word: 42H(B) | 41H(A)
2nd word: 44H(D) | 43H(C)
3rd word: 46H(F) | 45H(E)
4th word: 32H(2) | 31H(1)
5th word: 34H(4) | 33H(3)
6th word: 00H | 35H(5)
```

Deletion start position ( ) = 5th character

Number of characters to be deleted: n1 = 5

2) A character string (data) stored in devices specified in ( ) indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.
12.4 DELETE(_E) / String deletion

**Cautions**

1) Use the function having "_E" in its name to connect a bus.
2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
   Use global labels when specifying labels.

**Error**

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When "00H" does not exist in the corresponding device range after the device specified in  
   (Error code: K6706)
2) When the number of characters of a character string stored in devices specified in  is 32768 or more
   (Error code: K6706)
3) When the number of devices after the device number specified in  is smaller than the number of
   devices required for storing the character string remaining after deletion of specified number of
   characters
   (Error code: K6706)
4) When the value specified in  is negative
   (Error code: K6706)

**Program example**

In this program, specified number of characters are deleted from an arbitrary position of a character string
stored in devices specified in  , and the character string remaining after deletion is output to devices
specified in  .

1) Function without EN/ENO(DELETE)

   [Structured ladder/FBD]
   
   ![Structured ladder/FBD diagram]

   
   [ST]
   
   \( g\text{\_string2} := \text{DELETE}(g\text{\_string1}, g\text{\_int1}, g\text{\_int2}) ; \)

2) Function with EN/ENO(DELETE_E)

   [Structured ladder/FBD]
   
   ![Structured ladder/FBD diagram]

   [ST]
   
   \( g\text{\_bool3} := \text{DELETE\_E}(g\text{\_bool1}, g\text{\_string1}, g\text{\_int1}, g\text{\_int2}, g\text{\_string2}) ; \)
12.5 REPLACE(_E) / String replacement

Outline

This function replaces a character string.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLACE</td>
<td>REPLACE(_IN1,_IN2,_L,_P); REPLACE(_IN1,_IN2,_L,_P); Label3:= REPLACE(_IN1,_IN2,_L,_P); Example: Label3:= REPLACE(_IN1,_IN2,_L,_P);</td>
<td>REPLACE(_IN1,_IN2,_L,_P); REPLACE(_IN1,_IN2,_L,_P);</td>
</tr>
<tr>
<td>REPLACE_E</td>
<td>REPLACE_E(EN,_IN1,_IN2,_L,_P,Output_label); Example: REPLACE_E(X000,Label1,Label2,D20,D30,Label3);</td>
<td>REPLACE_E(EN,_IN1,_IN2,_L,_P,Output_label); REPLACE_E(X000,Label1,Label2,D20,D30,Label3);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Head word device which stores a character string to be replaced</td>
<td>String</td>
</tr>
<tr>
<td>_IN2</td>
<td>Head word device which stores a replacement character string</td>
<td>String</td>
</tr>
<tr>
<td>_L</td>
<td>Word device which stores the number of characters to be replaced</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>_P</td>
<td>Word device which stores the head character position to be replaced in a character string to be replaced</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store a character string obtained by replacement</td>
<td>String</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.
### Explanation of function and operation

1) This function replaces specified number of characters from an arbitrary position of a character string stored in devices specified in \( s1 \) with a character string stored in devices specified in \( s2 \), and outputs the character string obtained by replacement to devices specified in \( d \).

The value specified in \( n1 \) specifies the number of characters to be replaced. The value specified in \( n2 \) specifies the position from which specified number of characters are replaced.

Example: When "5" is specified in \( n1 \) and \( n2 \)

\[
\begin{align*}
\text{Input value to } s1 & : \quad "ABCDEFGH123" \\
\text{Output value} & : \quad "ABCD1234523"
\end{align*}
\]

2) A character string (data) stored in devices specified in \( s3 \) indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

3) When \( n1+n2 \) exceeds the number of characters of a character string stored in devices specified in \( s1 \), excessive characters are not output to devices specified in \( s5 \).

4) When "-1" is specified in \( n1 \), the number of characters of a character string stored in devices specified in \( s2 \) is regarded as the value specified in \( n1 \).

### Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data. Use global labels when specifying labels.
Error
An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When "00H" does not exist in the corresponding device range after the devices specified in \( S1 \) and \( S2 \) (Error code: K6706)

2) When the value specified in \( N1 \) exceeds the number of characters of a character string stored in devices specified in \( S2 \) (Error code: K6706)

3) When the value specified in \( N2 \) is negative (Error code: K6706)

4) When the value specified in \( N1 \) is "-2" or less (Error code: K6706)

5) When the value specified in \( N2 \) exceeds the number of characters of a character string stored in devices specified in \( S1 \) (Error code: K6706)

Program example
In this program, specified number of characters starting from an arbitrary position of a character string stored in devices specified in \( S1 \) are replaced with a character string stored in devices specified in \( S2 \), and the character string obtained by replacement is output to devices specified in \( d \).

1) Function without EN/ENO(REPLACE)

[Structured ladder/FBD]

1) Function without EN/ENO(REPLACE)

[Structured ladder/FBD]

1) Function without EN/ENO(REPLACE)

[Structured ladder/FBD]

1) Function without EN/ENO(REPLACE)

[Structured ladder/FBD]

1) Function without EN/ENO(REPLACE)

[Structured ladder/FBD]

1) Function without EN/ENO(REPLACE)

[Structured ladder/FBD]
12.6 FIND(_E) / Searches a character string

Outline
This function searches a character string.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND</td>
<td>FIND(_IN1,_IN2);</td>
</tr>
<tr>
<td></td>
<td>Example: D20:= FIND(Label1,Label2);</td>
</tr>
<tr>
<td>FIND_E</td>
<td>FIND_E(_EN,_IN1,_IN2, _ENO);</td>
</tr>
<tr>
<td></td>
<td>Example: FIND_E(X000,Label1,Label2, D20);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Head word device which stores a character string to get search</td>
<td>String</td>
</tr>
<tr>
<td>_IN2</td>
<td>Head word device which stores a character string to be searched</td>
<td>String</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store the search result</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.
**Explanation of function and operation**

1) This function searches a character string stored in devices specified in \( s_2 \) from the beginning of a character string stored in devices specified in \( s_1 \), and outputs the search result to devices specified in \( d \).

   This function outputs the head character position of the searched character string detected first as the search result.

2) A character string (data) stored in devices specified in \( s_1 \) indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

3) If a character string stored in devices specified in \( s_2 \) cannot be detected in a character string stored in devices specified in \( s_1 \), this function outputs "0".

**Cautions**

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data. Use global labels when specifying labels.

**Error**

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When "00H (NULL)" does not exist in the corresponding device range specified in \( s_1 \) (Error code: K6706)

2) When "00H (NULL)" does not exist in the corresponding device range specified in \( s_2 \) (Error code: K6706)
Program example

In this program, a character string stored in devices specified in (s2) is searched from the beginning of a character string stored in devices specified in (s1), and the search result is output to devices specified in (d).

1) Function without EN/ENO(FIND)

[Structured ladder/FBD]

```plaintext
FIND

<IN1>
g_string1="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
<IN2>
g_string2="EFGHIJK"
```

[ST]

```
g_int1:=FIND(g_string1,g_string2);
```

2) Function with EN/ENO(FIND_E)

[Structured ladder/FBD]

```plaintext
FIND_E

<EN>
g_bool1
<ENO>
g_bool3
<IN1>
g_string1
<IN2>
g_string2
```

[ST]

```
g_bool3:=FIND_E(g_bool1,g_string1,g_string2,g_int1);
```
## 13. Applied Functions (Functions Of Time Data Types)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TIME(_E)</td>
<td>Addition</td>
<td>Section 13.1</td>
</tr>
<tr>
<td>SUB_TIME(_E)</td>
<td>Subtraction</td>
<td>Section 13.2</td>
</tr>
<tr>
<td>MUL_TIME(_E)</td>
<td>Multiplication</td>
<td>Section 13.3</td>
</tr>
<tr>
<td>DIV_TIME(_E)</td>
<td>Division</td>
<td>Section 13.4</td>
</tr>
</tbody>
</table>
13 Applied Functions (Functions Of Time Data Types)

13.1 ADD_TIME(E) / Addition

Outline

This function adds time data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TIME</td>
<td>Label1 --- IN1 --- *1 --- Label3, Label2 --- IN2</td>
<td>ADD_TIME(IN1, IN2); Example: Label3 := ADD_TIME(Label1, Label2);</td>
</tr>
<tr>
<td>ADD_TIME_E</td>
<td>X000 --- EN --- ENO --- Label1 --- IN1 --- *1 --- Label3, Label2 --- IN2</td>
<td>ADD_TIME_E(EN, IN1, IN2, Output_label); Example: ADD_TIME_E(X000, Label1, Label2, Label3);</td>
</tr>
</tbody>
</table>

* Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td>IN1</td>
<td>Head word device which stores time data to get addition</td>
</tr>
<tr>
<td></td>
<td>IN2</td>
<td>Head word device which stores addition time data</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>*1</td>
<td>Head word device which will store the operation result</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function performs addition (s1 + s2) of time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in d.
Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

3) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error.
However, note that the accurate operation result cannot be obtained in this case.
("ADD_TIME_E" outputs "TRUE" from ENO.)

Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

<table>
<thead>
<tr>
<th>Device</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M8020</td>
<td>Zero</td>
<td>ON : When the operation result is &quot;0&quot;&lt;br&gt;OFF: When the operation result is any other than &quot;0&quot;</td>
</tr>
<tr>
<td>M8021</td>
<td>Borrow</td>
<td>ON : When the operation result is less than &quot;-32,768&quot; (16-bit operation) or less than &quot;-2,147,483,648&quot; (32-bit operation)&lt;br&gt;OFF: When the operation result is &quot;-32,768&quot; (16-bit operation) or more or &quot;-2,147,483,648&quot; (32-bit operation) or more</td>
</tr>
<tr>
<td>M8022</td>
<td>Carry</td>
<td>ON : When the operation result exceeds &quot;32,767&quot; (16-bit operation) or more or &quot;2,147,483,647&quot; (32-bit operation) or more&lt;br&gt;OFF: When the operation result is &quot;32,767&quot; (16-bit operation) or less or &quot;2,147,483,647&quot; (32-bit operation) or less</td>
</tr>
</tbody>
</table>

Program example

In this program, addition (s1 + s2) is performed using time data stored in devices specified in s1 and s2, and the operation result expressed as time data is output to devices specified in d.

1) Function without EN/ENO(ADD_TIME)

[Structured ladder/FBD]

```
  g_time1 __IN1
  g_time2 __IN2
  ADD_TIME
  g_time3
```

[ST]

```
g_time3:=ADD_TIME(g_time1,g_time2);
```

2) Function with EN/ENO(ADD_TIME_E)

[Structured ladder/FBD]

```
  g_bool1  
  g_bool2  
  ADD_TIME_E
  g_bool3
  EN  g_time1 __IN1
  ENO  g_time2 __IN2
  g_time3
```

[ST]

```
g_bool3:=ADD_TIME_E(g_bool1,g_time1,g_time2,g_time3);
```
13.2 SUB_TIME(E) / Subtraction

Outline
This function performs subtraction of time data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB_TIME</td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td></td>
<td>SUB_TIME(_IN1,_IN2);</td>
</tr>
<tr>
<td></td>
<td>Example: Label3 = SUB_TIME(Label1, Label2);</td>
</tr>
<tr>
<td>SUB_TIME_E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X000 SUB_TIME_E(EN, _IN1,_IN2, Output_label);</td>
</tr>
<tr>
<td></td>
<td>Example: SUB_TIME_E(X000,Label1, Label2,Label3);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Head word device which stores time data to get subtraction</td>
<td>Time</td>
</tr>
<tr>
<td>_IN2</td>
<td>Head word device which stores subtraction data</td>
<td>Time</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store the operation result</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function performs subtraction (_s1 - _s2_) of time data stored in devices specified in _s1_ and _s2_, and outputs the operation result expressed as time data to devices specified in _d_.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>Head word device which stores time data to get subtraction</td>
<td>Time</td>
</tr>
<tr>
<td>IN2</td>
<td>Head word device which stores subtraction data</td>
<td>Time</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store the operation result</td>
<td>Time</td>
</tr>
</tbody>
</table>
13.2 SUB_TIME(_E) / Subtraction

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

3) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. However, note that the accurate operation result cannot be obtained in this case. ("SUB_TIME_E" outputs "TRUE" from ENO.)

Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

<table>
<thead>
<tr>
<th>Device</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M8020</td>
<td>Zero</td>
<td>ON : When the operation result is &quot;0&quot;&lt;br&gt;OFF: When the operation result is any other than &quot;0&quot;</td>
</tr>
<tr>
<td>M8021</td>
<td>Borrow</td>
<td>ON : When the operation result is less than &quot;-32,768&quot; (16-bit operation) or less than &quot;-2,147,483,648&quot; (32-bit operation)&lt;br&gt;OFF: When the operation result is &quot;-32,768&quot; (16-bit operation) or more or &quot;-2,147,483,648&quot; (32-bit operation) or more</td>
</tr>
<tr>
<td>M8022</td>
<td>Carry</td>
<td>ON : When the operation result exceeds &quot;32,767&quot; (16-bit operation) or &quot;2,147,483,647&quot; (32-bit operation)&lt;br&gt;OFF: When the operation result is &quot;32,767&quot; (16-bit operation) or less or &quot;2,147,483,647&quot; (32-bit operation) or less</td>
</tr>
</tbody>
</table>

Program example

In this program, subtraction (\( s1 - s2 \)) is performed using time data stored in devices specified in \( s1 \) and \( s2 \), and the operation result expressed as time data is output to devices specified in \( d \).

1) Function without EN/ENO(SUB_TIME)

[Structured ladder/FBD]

```plaintext
SUB_TIME
  g_time1 _IN1
  g_time2 _IN2
  g_time3
```

[ST]

\[ g\textunderscore time3:=\text{SUB\_TIME}(g\textunderscore time1,g\textunderscore time2); \]

2) Function with EN/ENO(SUB_TIME_E)

[Structured ladder/FBD]

```plaintext
SUB_TIME_E
  EN
  g_bool1
  g_time1 _IN1
  g_time2 _IN2
  g_bool3
  g_time3
```

[ST]

\[ g\textunderscore bool3:=\text{SUB\_TIME\_E}(g\textunderscore bool1,g\textunderscore time1,g\textunderscore time2,g\textunderscore time3); \]
13.3 MUL_TIME(_E) / Multiplication

Outline
This function performs multiplication of time data.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL_TIME</td>
<td>Label1 _IN1 *1 Label3</td>
<td>MUL_TIME(_IN1,_IN2); Example: Label3 := MUL_TIME(Label1,Label2);</td>
</tr>
<tr>
<td>MUL_TIME_E</td>
<td>X000 _EN _ENO Label3</td>
<td>MUL_TIME(E(_EN1,_IN2, Output:Label); Example: MUL_TIME_E(X000,Label1, Label2,Label3);</td>
</tr>
</tbody>
</table>

*1. Output variable

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Head word device which stores time data to get multiplication</td>
<td>Time</td>
</tr>
<tr>
<td>_IN2</td>
<td>Multiplication data, or head word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>_ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store the operation result</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function performs multiplication (s1 × s2) using time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in s.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
3) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. However, note that the accurate operation result cannot be obtained in this case. ("MUL_TIME_E" outputs "TRUE" from ENO.)
4) When the FLOAT (Single Precision) data to  is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.
### Program example

In this program, multiplication \((s1 \times s2)\) is performed using time data stored in devices specified in \(s1\) and \(s2\), and the operation result expressed as time data is output to devices specified in \(d\).

1) **Function without EN/ENO(MUL_TIME)**

   ![Structured ladder/FBD](image)

   
   ST
   
   \[g\_time2 := \text{MUL\_TIME}(g\_time1, g\_int1);\]

2) **Function with EN/ENO(MUL\_TIME\_E)**

   ![Structured ladder/FBD](image)

   
   ST
   
   \[g\_bool3 := \text{MUL\_TIME\_E}(g\_bool1, g\_time1, g\_int1, g\_time2);\]
### 13.4 DIV_TIME(_E) / Division

This function performs division using time data.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV_TIME</td>
<td>Label1 _IN1 *1 Label3</td>
<td></td>
</tr>
<tr>
<td>DIV_TIME</td>
<td>Label2 _IN2</td>
<td></td>
</tr>
<tr>
<td>DIV_TIME_E</td>
<td>X000</td>
<td></td>
</tr>
<tr>
<td>DIV_TIME</td>
<td>Label1 _IN1 *1 Label3</td>
<td></td>
</tr>
<tr>
<td>DIV_TIME</td>
<td>Label2 _IN2</td>
<td></td>
</tr>
</tbody>
</table>

*1 Output variable

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_IN1</td>
<td>Head word device which stores time data to get division</td>
<td>Time</td>
</tr>
<tr>
<td>_IN2</td>
<td>Division data, or head word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output variable ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>*1</td>
<td>Head word device which will store the operation result</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

#### Explanation of function and operation

1) This function performs division (\( \frac{s1}{s2} \)) using time data stored in devices specified in \( s1 \) and \( s2 \), and outputs the operation result expressed as time data to devices specified in \( d \).

2) The contents of devices specified in \( s2 \) are ANY_NUM type data except "0".

#### Cautions

1) Use the function having " _E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

3) When the FLOAT (Single Precision) data to \( s5 \) is set from the programming tool, a rounding error may be generated. Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

#### Error

1) An operation error occurs when the divisor stored in devices specified in \( s2 \) is "0", and the function is not executed.

2) An operation error occurs when the operation result exceeds "2,147,483,647".
Program example

In this program, division \((\frac{s_1}{s_2})\) is performed using time data stored in devices specified in \(s_1\) and \(s_2\), and the operation result expressed as time data is output to devices specified in \(d\).

1) Function without EN/ENO(DIV_TIME)

[Structured ladder/FBD]

\[
\begin{align*}
\text{DIV_TIME} & \quad _{\text{IN}1} \quad g_{\text{time}1} \\
_{\text{IN}2} \quad g_{\text{int}1} & \quad \quad g_{\text{time}2}
\end{align*}
\]

[ST]

\[g_{\text{time}2} := \text{DIV}\_\text{TIME}(g_{\text{time}1}, g_{\text{int}1});\]

2) Function with EN/ENO(DIV_TIME_E)

[Structured ladder/FBD]

\[
\begin{align*}
\text{DIV\_TIME\_E} & \quad EN \quad _{\text{IN}1} \quad g_{\text{bool}1} \\
& \quad ENO \quad _{\text{IN}2} \quad g_{\text{time}1} \\
& \quad \quad \quad \quad g_{\text{int}1} \quad g_{\text{time}2} \\
& \quad \quad \quad \quad g_{\text{bool}3}
\end{align*}
\]

[ST]

\[g_{\text{bool}3} := \text{DIV\_TIME\_E}(g_{\text{bool}1}, g_{\text{time}1}, g_{\text{int}1}, g_{\text{time}2});\]
## 14. Standard Function Blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TRIG(_E)</td>
<td>Rising edge detector</td>
<td>Section 14.1</td>
</tr>
<tr>
<td>F_TRIG(_E)</td>
<td>Falling edge detector</td>
<td>Section 14.2</td>
</tr>
<tr>
<td>CTU(_E)</td>
<td>Up counter</td>
<td>Section 14.3</td>
</tr>
<tr>
<td>CTD(_E)</td>
<td>Down counter</td>
<td>Section 14.4</td>
</tr>
<tr>
<td>CTUD(_E)</td>
<td>Up/Down counter</td>
<td>Section 14.5</td>
</tr>
<tr>
<td>TP(_E)</td>
<td>Pulse timer</td>
<td>Section 14.6</td>
</tr>
<tr>
<td>TP_10(_E)</td>
<td>Pulse timer</td>
<td>Section 14.6</td>
</tr>
<tr>
<td>TON(_E)</td>
<td>On delay timer</td>
<td>Section 14.7</td>
</tr>
<tr>
<td>TON_10(_E)</td>
<td>On delay timer</td>
<td>Section 14.7</td>
</tr>
<tr>
<td>TOF(_E)</td>
<td>Off delay timer</td>
<td>Section 14.8</td>
</tr>
<tr>
<td>TOF_10(_E)</td>
<td>Off delay timer</td>
<td>Section 14.8</td>
</tr>
<tr>
<td>COUNTER_FB_M</td>
<td>Counter function blocks</td>
<td>Section 14.9</td>
</tr>
<tr>
<td>TIMER_10_FB_M</td>
<td>Timer function blocks</td>
<td>Section 14.10</td>
</tr>
<tr>
<td>TIMER_CONT_FB_M</td>
<td>Timer function blocks</td>
<td>Section 14.11</td>
</tr>
<tr>
<td>TIMER_100_FB_M</td>
<td>Timer function blocks</td>
<td>Section 14.12</td>
</tr>
</tbody>
</table>
14.1 R_TRIG(_E) / Rising edge detector

Outline
This function block detects the rising edge of a signal, and outputs pulse signal.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TRIG</td>
<td>R_TRIG(_CLK,Q);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instance name(_CLK:=M0, Q:=M10);</td>
<td></td>
</tr>
<tr>
<td>R_TRIG_E</td>
<td>R_TRIG_E(EN,_CLK,Q,ENO);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instance name(EN:=X000, _CLK:=M0,Q:=M10);</td>
<td></td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>_CLK</td>
<td>Input signal whose rising edge is to be detected</td>
<td>Bit</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>Q</td>
<td>Output signal</td>
<td>Bit</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function block sets to ON a device specified in ( ) when a device specified in ( ) turns ON, and keeps ON the device specified in ( ) only for 1 operation cycle.

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) Expression of function blocks in each language
   - Set the instance when using a function block.
   - Describe the instance name when programming a function block.

Error
1) When an output number is specified in ( ) and the specified output number does not exist due to indexing, M8316 (I/O inexistence error) turns ON.
   (Applicable to the FX3U and FX3UC PLCs only)
2) When a device (M, T or C) other than I/O number is specified in ( ) and the specified device number does not exist due to indexing, an operation error (Error code: 6706) occurs.
Program example
In this program, a device specified in \( \bullet \) turns ON when the bit data stored in a device specified in \( \circ \) turns ON from OFF, and the device specified in \( \circ \) remains ON only for 1 operation cycle.

1) Function without EN/ENO(R_TRIG)

[Structured ladder/FBD]

\[
\text{R_TRIG}_\text{Instance}(\text{CLK}=g\_bool1, \text{Q}=g\_bool2);
\]

2) Function with EN/ENO(R_TRIG_E)

[Structured ladder/FBD]

\[
\text{R_TRIG}_E\_\text{Instance}(\text{EN}=X000, \_\text{CLK}=g\_bool1, \text{Q}=g\_bool2, \text{ENO}=Y010);
\]
14.2 F_TRIG(_E) / Falling edge detector

Outline
This function block detects the falling edge of a signal, and outputs pulse signal.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_TRIG</td>
<td>[ F_TRIG(EN,CLK,Q);]</td>
<td>F_TRIG(EN,CLK,Q);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Instance name(EN:=X000,</td>
<td>Instance name(EN:=X000,</td>
</tr>
<tr>
<td></td>
<td>_CLK:=M0, Q:=M10);</td>
<td>_CLK:=M0, Q:=M10);</td>
</tr>
<tr>
<td>F_TRIG_E</td>
<td>[ F_TRIG_E(ENO,CLK,Q);]</td>
<td>F_TRIG_E(ENO,CLK,Q);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Instance name(ENO:=X000,</td>
<td>Instance name(ENO:=X000,</td>
</tr>
<tr>
<td></td>
<td>_CLK:=M0, Q:=M10);</td>
<td>_CLK:=M0, Q:=M10);</td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>EN</td>
<td>Execution condition</td>
</tr>
<tr>
<td></td>
<td>_CLK ( )</td>
<td>Bit</td>
</tr>
<tr>
<td>Output variable</td>
<td>ENO</td>
<td>Execution status</td>
</tr>
<tr>
<td></td>
<td>Q ( )</td>
<td>Output signal</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function block sets to ON a device specified in ( ) when a device specified in ( ) turns OFF, and keeps ON the device specified in ( ) only for 1 operation cycle.

Cautions
1) Use the function having " _E" in its name to connect a bus.
2) Expression of function blocks in each language
   • Set the instance when using a function block.
   • Describe the instance name when programming a function block.

Error
1) When an output number is specified in ( ) and the specified output number does not exist due to indexing, M8316 (I/O inexistence error) turns ON.
   (Applicable to the FX3u and FX3uc PLCs only)
2) When a device (M, T or C) other than I/O number is specified in ( ) and the specified device number does not exist due to indexing, an operation error (Error code: 6706) occurs.
Program example

In this program, a device specified in \( \text{d} \) turns ON when the bit data stored in a device specified in \( \text{d} \) turns OFF from ON, and the device specified in \( \text{d} \) remains ON only for 1 operation cycle.

1) Function without EN/ENO(F_TRIG)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool1}} \quad \text{CLK} \quad Q \quad g_{\text{bool2}} \\
\end{array}
\]

[ST]

\[F_{\text{TRIG}}_{\text{Instance}}(_{\text{CLK}}:=g_{\text{bool1}},Q:=g_{\text{bool2}});\]

2) Function with EN/ENO(F/TRIG_E)

[Structured ladder/FBD]

\[
\begin{array}{c}
X_{000} \\
g_{\text{bool1}} \\
\end{array}
\quad
\begin{array}{c}
F_{\text{TRIG}}_{\text{E}}_{\text{Instance}} \\
\text{EN} \quad \text{ENO} \\
\text{CLK} \quad Q \\
\end{array}
\quad
\begin{array}{c}
Y_{010} \\
g_{\text{bool2}} \\
\end{array}
\]

[ST]

\[F_{\text{TRIG}}_{\text{E}}_{\text{Instance}}(_{\text{EN}}:=X_{000},_{\text{CLK}}:=g_{\text{bool1}},Q:=g_{\text{bool2}},_{\text{ENO}}:=Y_{010});\]
14.3 CTU(_E) / Up counter

Outline

This function block counts up the number of times of rising of a signal.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTU</td>
<td>CTU(CU,RESET,PV,Q,CV);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Instance name(CU:=M0,</td>
</tr>
<tr>
<td></td>
<td>RESET:=M10,PV:=D0,Q:=M20,</td>
</tr>
<tr>
<td></td>
<td>CV:=D10);</td>
</tr>
<tr>
<td>CTU_E</td>
<td>CTU_E(EN,CU,RESET,PV,Q,CV,</td>
</tr>
<tr>
<td></td>
<td>ENO);</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>Instance name(EN:=X000,</td>
</tr>
<tr>
<td></td>
<td>CU:=M0,RESET:=M10,PV:=D0,</td>
</tr>
<tr>
<td></td>
<td>Q:=M20,CV:=D10);</td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>CU</td>
<td>Count source signal</td>
<td>Bit</td>
</tr>
<tr>
<td>RESET</td>
<td>Reset input signal</td>
<td>Bit</td>
</tr>
<tr>
<td>PV</td>
<td>Counter set value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>Q</td>
<td>Count-up output signal</td>
<td>Bit</td>
</tr>
<tr>
<td>CV</td>
<td>Number of times of rising</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function block counts up (adds "1" to) the value stored in a device specified in ( ) when a device specified in ( ) turns ON.

When the count value reaches a value specified in ( ), a device specified in ( ) turns ON.

When a device specified in ( ) turns ON, this function block turns OFF a device specified in ( ), and resets the count value of a device specified in ( ).

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) Expression of function blocks in each language
   - Set the instance when using a function block.
   - Describe the instance name when programming a function block.
Program example

In this program, the number of times the bit data stored in a device specified in \( g_\text{bool1} \) turns ON from OFF is counted, and the count value is output to a device specified in \( g_\text{int2} \).

1) Function without EN/ENO(CTU)

[Structured ladder/FBD]

\[
\text{CTU\_Instance} \\
\begin{array}{c}
g_\text{bool1} \rightarrow \text{CU} \rightarrow g_\text{bool3} \\
g_\text{bool2} \rightarrow \text{RESET} \rightarrow g_\text{int2} \\
g_\text{int1} \rightarrow \text{PV}
\end{array}
\]

[ST]

\[
\text{CTU\_Instance}(:=g_\text{bool1}, \text{RESET}:=g_\text{bool2}, \text{PV}:=g_\text{int1}, Q:=g_\text{bool3}, CV:=g_\text{int2})
\]

2) Function with EN/ENO(CTU\_E)

[Structured ladder/FBD]

\[
\text{CTU\_E\_Instance} \\
\begin{array}{c}
M10 \rightarrow \text{EN} \rightarrow ENO \rightarrow M11 \\
\text{g}_\text{bool1} \rightarrow \text{CU} \rightarrow g_\text{bool3} \\
\text{g}_\text{bool2} \rightarrow \text{RESET} \rightarrow g_\text{int2} \\
\text{g}_\text{int1} \rightarrow \text{PV}
\end{array}
\]

[ST]

\[
\text{CTU\_E\_Instance}(:=M10, \text{CU}:=g_\text{bool1}, \text{RESET}:=g_\text{bool2}, \text{PV}:=g_\text{int1}, Q:=g_\text{bool3}, CV:=g_\text{int2}, \text{ENO}:=M11)
\]
14.4 CTD(_E) / Down counter

Outline
This function block counts down the number of times of rising of a signal.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTD</td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M0 CTD Q M20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M10 LOAD CV D10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTD(CD,LOAD,PV,Q,CV);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instance name(CD:=M0,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD:=M10,PV:=D0,Q:=M20,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CV:=D10);</td>
<td></td>
</tr>
<tr>
<td>CTD_E</td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X000 CTD_E EN ENO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M0 CD Q M20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M10 LOAD CV D10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTD_E(EN,CD,LOAD,PV,Q,CV,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENO);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instance name(EN:=X000,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CD:=M0,LOAD:=M10,PV:=D0,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q:=M20,CV:=D10);</td>
<td></td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN (EN)</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>CD (CD)</td>
<td>Count source signal</td>
<td>Bit</td>
</tr>
<tr>
<td>LOAD (CD)</td>
<td>Reset input signal</td>
<td>Bit</td>
</tr>
<tr>
<td>PV (PV)</td>
<td>Counter set value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>Q (Q)</td>
<td>Output signal (which turns ON when the current counter value becomes &quot;0&quot; or less)</td>
<td>Bit</td>
</tr>
<tr>
<td>CV (CV)</td>
<td>Number of times of rising</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function block counts down (subtracts "1" from) the value stored in a device specified in (CD) when a device specified in (CD) turns ON.
The value (CD) specifies the initial value for subtraction.
This function block turns ON a device specified in (CD) when the count value becomes "0".
When a device specified in (CD) turns ON, this function block turns OFF a device specified in (CD), and sets the initial value for subtraction specified in (CD) to the count value of a device specified in (CD).

Cautions
1) Use the function having "_E" in its name to connect a bus.
2) Expression of function blocks in each language
   • Set the instance when using a function block.
   • Describe the instance name when programming a function block.
Program example

In this program, the number of times the bit data stored in a device specified in \( \text{ST} \) turns ON from OFF is counted, and a device specified in \( \text{ST} \) turns ON when the value stored in a device specified in \( \text{CTD}_E \) becomes "0".

1) Function without EN/ENO(CTD)

[Structured ladder/FBD]

\[
\begin{array}{c}
\text{CTD}\_\text{Instance} \\
g\_\text{bool1} & \text{CD} & Q & g\_\text{bool3} \\
g\_\text{bool2} & \text{LOAD} & CV & g\_\text{int2} \\
g\_\text{int1} & \text{PV} & & \\
\end{array}
\]

[ST]

\[
\text{CTD}\_\text{Instance} (\text{CD}:=g\_\text{bool1}, \text{LOAD}:=g\_\text{bool2}, \text{PV}:=g\_\text{int1}, Q:=g\_\text{bool3}, CV:=g\_\text{int2});
\]

2) Function with EN/ENO(CTD)_E

[Structured ladder/FBD]

\[
\begin{array}{c}
\text{CTD}_E\_\text{Instance} \\
\text{EN} & \text{ENO} & M11 \\
\text{CD} & Q & g\_\text{bool3} \\
\text{LOAD} & CV & g\_\text{int2} \\
\text{PV} & & \\
\end{array}
\]

[ST]

\[
\text{CTD}_E\_\text{Instance} (\text{EN}:=\text{M10}, \text{CD}:=g\_\text{bool1}, \text{LOAD}:=g\_\text{bool2}, \text{PV}:=g\_\text{int1}, Q:=g\_\text{bool3}, CV:=g\_\text{int2}, \text{ENO}:=\text{M11});
\]
14.5 CTUD(_E) / Up/Down counter

Outline
This function block counts up/down the number of times of rising of a signal.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTUD</td>
<td>CTUD(CU,CD,RESET,LOAD,PV, QU,QD,CV);</td>
</tr>
<tr>
<td></td>
<td>Example: Instance name(CU:=M0, CD:=M10,RESET:=M20, LOAD:=M30,PV:=D0,QU:=M40, QD:=M50,CV:=D10);</td>
</tr>
<tr>
<td>CTUD_E</td>
<td>CTUD_E(EN,CU,CD,RESET, LOAD,PV,QU,QD,CV,ENO);</td>
</tr>
<tr>
<td></td>
<td>Example: Instance name(EN:=X000, CU:=M0,CD:=M10,RESET:=M20, LOAD:=M30,PV:=D0,QU:=M40, QD:=M50,CV:=D10);</td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>CU ( )</td>
<td>Count up signal</td>
<td>Bit</td>
</tr>
<tr>
<td>CD ( )</td>
<td>Count down signal</td>
<td>Bit</td>
</tr>
<tr>
<td>RESET ( )</td>
<td>Reset input signal</td>
<td>Bit</td>
</tr>
<tr>
<td>LOAD ( )</td>
<td>Resetting signal</td>
<td>Bit</td>
</tr>
<tr>
<td>PV ( )</td>
<td>Counter set value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>QU ( )</td>
<td>Count-up output signal</td>
<td>Bit</td>
</tr>
<tr>
<td>QD ( )</td>
<td>Output signal (which turns ON when the current counter value becomes &quot;0&quot; or less)</td>
<td>Bit</td>
</tr>
<tr>
<td>CV ( )</td>
<td>Count value data</td>
<td>Word [signed]</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation
This function block counts up (adds "1" to) the value stored in a device specified in ( ) when a device specified in ( ) turns ON.
This function block counts down (subtracts "1" from) the value stored in a device specified in ( ) when a device specified in ( ) turns ON.
( ) specifies the maximum value of the counter.
When the value stored in a device specified in ( ) reaches the maximum value ( ) of the counter, a device specified in ( ) turns ON.
When the value stored in a device specified in ( ) becomes "0", a device specified in ( ) turns ON.
This function block resets the count value of a device specified in ( ) when a device specified in ( ) turns ON.
This function block sets the value stored in ( ) to a device specified in ( ) when a device specified in ( ) turns ON.
Cautions

1) Use the function having ",_E" in its name to connect a bus.
2) Expression of function blocks in each language
   • Set the instance when using a function block.
   • Describe the instance name when programming a function block.

Program example

In this program, the number of times the bit data stored in a device specified in \( s_1 \) turns ON from OFF is counted up (added by "1"). When the value stored in a device specified in \( d_3 \) reaches the value specified in \( n \), a device specified in \( d_1 \) turns ON.

At the same time, the number of times the bit data stored in a device specified in \( s_2 \) turns ON from OFF is counted down (subtracted by "1"). When the value stored in a device specified in \( d_2 \) becomes "0", a device specified in \( d_2 \) turns ON.

1) Function without EN/ENO(CTUD)

[Structured ladder/FBD]

\[
\begin{align*}
\text{CTUD} & \text{ Instance} \\
\text{g_bool1} & \text{CU} & \text{g_bool5} \\
\text{g_bool2} & \text{CD} & \text{g_bool6} \\
\text{g_bool3} & \text{RESET} & \text{g_int2} \\
\text{g_bool4} & \text{LOAD} & \\
\text{g_int1} & \text{PV} & \\
\end{align*}
\]

[ST]

\[
\text{CTUD}\_\text{Instance}(\text{CU}=\text{g_bool1},\text{CD}=\text{g_bool2},\text{RESET}=\text{g_bool3},\text{LOAD}=\text{g_bool4},\text{PV}=\text{g_int1},\text{QU}=\text{g_bool5},\text{QD}=\text{g_bool6},\text{CV}=\text{g_int2});
\]

2) Function with EN/ENO(CTUD_E)

[Structured ladder/FBD]

\[
\begin{align*}
\text{CTUD}_E & \text{ Instance} \\
\text{M0} & \\
\text{g_bool1} & \text{EN} & \text{M10} \\
\text{g_bool2} & \text{CU} & \text{g_bool5} \\
\text{g_bool3} & \text{CD} & \text{g_bool6} \\
\text{g_bool4} & \text{RESET} & \text{g_int2} \\
\text{g_int1} & \text{LOAD} & \\
\text{PV} & \\
\end{align*}
\]

[ST]

\[
\text{CTUD}_E\_\text{Instance}(\text{EN}=\text{M0},\text{CU}=\text{g_bool1},\text{CD}=\text{g_bool2},\text{RESET}=\text{g_bool3},\text{LOAD}=\text{g_bool4},\text{PV}=\text{g_int1},\text{QU}=\text{g_bool5},\text{QD}=\text{g_bool6},\text{CV}=\text{g_int2},\text{ENO}=\text{M10});
\]
14.6 **TP(_E), TP_10(_E) / Pulse timer**

### Outline

This function block keeps ON a signal for specified duration.

#### 1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TP</strong></td>
<td></td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>M0 IN TP M10 Label1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Label1 PT ET Label2</td>
<td></td>
</tr>
<tr>
<td><strong>TP_E</strong></td>
<td></td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X000 IN TP_E EN Q M10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Label1 PT ET Label2</td>
<td></td>
</tr>
<tr>
<td><strong>TP_10</strong></td>
<td></td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>M0 IN TP_10 M10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Label1 PT ET Label2</td>
<td></td>
</tr>
<tr>
<td><strong>TP_10_E</strong></td>
<td></td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X000 IN TP_10_E EN Q M10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Label1 PT ET Label2</td>
<td></td>
</tr>
</tbody>
</table>

#### 2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EN</strong></td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td>ON start input signal</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>PT</strong></td>
<td>ON duration data</td>
<td>Time</td>
</tr>
<tr>
<td><strong>ENO</strong></td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td>Output signal</td>
<td>Bit</td>
</tr>
<tr>
<td><strong>ET</strong></td>
<td>ON duration current value</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

### Explanation of function and operation

1) When a device specified in (S) turns ON, this function block turns ON a device specified in (D1), and keeps it ON for duration specified in (D1).
   - The elapsed time while a device specified in (D1) remains ON is set to a device specified in (D2).
   - A device specified in (D1) remains ON from OFF next time, this function block resets the elapsed time and turns ON again a device specified in (D1).

2) The setting of ON duration data specified in (D).
   - **TP(_E)** : Time can be set in units of 100 ms or more.
   - **TP_10(_E)** : Time can be set in units of 10 ms or more.
Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
3) Expression of function blocks in each language
   • Set the instance when using a function block.
   • Describe the instance name when programming a function block.

Program example

In this program, when bit data stored in a device specified in \( s \) turns ON, bit data stored in a device specified in \( T \) turns ON and remains ON for 10 seconds.

1) Function without EN/ENO(TP,TP_10)

[Structured ladder/FBD]

```plaintext
TP_Instance(IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1);
```

[ST]

```
TP_Instance(IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1);
```

2) Function with EN/ENO(TP_E,TP_10_E)

[Structured ladder/FBD]

```plaintext
TP_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1,ENO:=M10);
```

[ST]

```
TP_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1,ENO:=M10);
```
14.7 TON(_E), TON_10(_E) / On delay timer

Outline
This function block turns ON after specified time.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>TON</td>
<td>M0 - EN IN PT Q M10 Label1</td>
<td>TON(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1, Q:=M10, ET:=Label2);</td>
</tr>
<tr>
<td>TON_E</td>
<td>X000 IN EN IN PT Q M10 Label1</td>
<td>TON_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0, PT:=Label1, Q:=M10, ET:=Label2);</td>
</tr>
<tr>
<td>TON_10</td>
<td>M0 - IN PT Q M10 Label1</td>
<td>TON_10(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1, Q:=M10, ET:=Label2);</td>
</tr>
<tr>
<td>TON_10_E</td>
<td>X000 IN EN IN PT Q M10 Label1</td>
<td>TON_10_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0, PT:=Label1, Q:=M10, ET:=Label2);</td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>IN</td>
<td>Input signal</td>
<td>Bit</td>
</tr>
<tr>
<td>PT</td>
<td>ON start time data</td>
<td>Time</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>Q</td>
<td>Output signal</td>
<td>Bit</td>
</tr>
<tr>
<td>ET</td>
<td>ON start time current value</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) When a device specified in (s) turns ON, this function block turns ON a device specified in (dt) after the time specified in (t).
   The delay elapsed time until a device specified in (dt) turns ON is set to a device specified in (gt).
   When a device specified in (s) turns OFF, this function block turns OFF a device specified in (dt) and resets the delay elapsed time.

2) The setting of ON duration data specified in (n).
   TON(_E) : Time can be set in units of 100 ms or more.
   TON_10(_E) : Time can be set in units of 10 ms or more.
Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

3) Expression of function blocks in each language
   - Set the instance when using a function block.
   - Describe the instance name when programming a function block.

Program example

In this program, when bit data stored in a device specified in (s) turns ON, bit data stored in a device specified in (GT) turns ON 10 seconds later.

1) Function without EN/ENO(TON,TON_10)

   [Structured ladder/FBD]

   ![Structured ladder/FBD Diagram]

   ST

   TON_Instance(IN:=g Bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1);

2) Function with EN/ENO(TON_E,TON_10_E)

   [Structured ladder/FBD]

   ![Structured ladder/FBD Diagram]

   ST

   TON_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1,ENO:=M10)
14.8 TOF(_E), TOF_10(_E) / Off delay timer

Outline
When the input signal turns OFF, this function block turns OFF the output signal after the specified time.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOF</td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td>M0</td>
<td>PT</td>
<td>Q</td>
</tr>
<tr>
<td>Label1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOF_E</td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td>X000</td>
<td>EN</td>
<td>TOF_E</td>
</tr>
<tr>
<td>M0</td>
<td>IN</td>
<td>Q</td>
</tr>
<tr>
<td>Label1</td>
<td>PT</td>
<td>ET</td>
</tr>
<tr>
<td>TOF_10</td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td>M0</td>
<td>IN</td>
<td>Q</td>
</tr>
<tr>
<td>Label1</td>
<td>PT</td>
<td>ET</td>
</tr>
<tr>
<td>TOF_10_E</td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td>X000</td>
<td>EN</td>
<td>TOF_10_E</td>
</tr>
<tr>
<td>M0</td>
<td>IN</td>
<td>Q</td>
</tr>
<tr>
<td>Label1</td>
<td>PT</td>
<td>ET</td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>IN</td>
<td>Input signal</td>
<td>Bit</td>
</tr>
<tr>
<td>PT</td>
<td>OFF start time data</td>
<td>Time</td>
</tr>
<tr>
<td>ENO</td>
<td>Execution status</td>
<td>Bit</td>
</tr>
<tr>
<td>Q</td>
<td>Output signal</td>
<td>Bit</td>
</tr>
<tr>
<td>ET</td>
<td>OFF duration current value</td>
<td>Time</td>
</tr>
</tbody>
</table>

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) When a device specified in \( s \) turns ON, this function block turns ON a device specified in \( d_1 \).
   When a device specified in \( s \) turns OFF from ON, this function block turns OFF a device specified in \( d_1 \) after the time specified in \( n \).
   The elapsed time until a device specified in \( d_1 \) turns OFF is set to a device specified in \( d_2 \).
   When a device specified in \( s \) turns ON again, this function block turns ON a device specified in \( d_1 \) and resets the elapsed time.

2) The setting of ON duration data specified in \( n \).
   TOF(_E) : Time can be set in units of 100 ms or more.
   TOF_10(_E) : Time can be set in units of 10 ms or more.
Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.
3) Expression of function blocks in each language
   • Set the instance when using a function block.
   • Describe the instance name when programming a function block.

Program example

In this program, when bit data stored in a device specified in \( s \) turns ON, bit data stored in a device specified in \( s \) turns ON. When bit data stored in a device specified in \( s \) turns OFF, bit data stored in a device specified in \( s \) turns OFF 10 seconds later.

1) Function without EN/ENO(ToF, ToF_10)

[Structured ladder/FBD]

```
TOF_Instance

<table>
<thead>
<tr>
<th>( \text{g Bool1} )</th>
<th>ToF</th>
<th>( \text{Q Bool2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{T#10s} )</td>
<td>PT</td>
<td>( \text{g Time1} )</td>
</tr>
</tbody>
</table>
```

[ST]

```
TOF_Instance(IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1);
```

2) Function with EN/ENO(ToF_E, ToF_10_E)

[Structured ladder/FBD]

```
TOF_E_Instance

| \( M0 \) | ToF_E | \( \text{EN} \) | \( \text{ENO} \) |
|----------|-------|----------------|
| \( \text{g Bool1} \) | EN    | \( \text{Q} \)  | \( \text{g Bool2} \) |
| \( \text{T#10s} \)   | PT    | \( \text{ET} \)  | \( \text{g Time1} \) |
```

[ST]

```
TOF_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1,ENO:=M10);
```
14.9 COUNTER_FB_M / Counter function blocks

Outline

This counter starts counting when the condition turns ON from OFF and generates an output when counting up to the set value.
A counter initial value can be set.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTER_FB_M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coil</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>Preset</td>
<td>Counter set value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ValueIn</td>
<td>Counter initial value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ValueOut</td>
<td>Counter current value</td>
<td>ANY16</td>
</tr>
<tr>
<td>Status</td>
<td>Counter output contact</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Function and operation explanation

1) The counter starts counting when detecting the rising edge (from OFF to ON) of the input argument Coil. It does not start counting if the Coil remains ON. The counter starts counting from the value of input argument ValueIn. When the input argument Preset value is reached, the output argument Status turns ON. The current count value is stored in the output argument ValueOut.

[Structured ladder/FBD]

COUNTER_FB_M_Instance(Coil:= Var_M0, Preset:=10, ValueIn:=1, ValueOut:=Var_D10, Status:=Var_M10);

[ST]

COUNTER_FB_M_Instance(Coil:= Var_M0, Preset:=10, ValueIn:=1, ValueOut:=Var_D10, Status:=Var_M10);

*1. Var_D10 is a global label and is defined as D10.

*2. Var_M10 is a global label and is defined as M10.
2) When resetting the current value of the counter, reset input variable coil.

[Structured ladder/FBD]

```
COUNTER_FB_M_Instance
Var_M0  COUNTER_FB_M  Var_D10
10     Preset           Var_M10
1      ValueIn

M15
EN   RST
ENO  d  COUNTER_FB_M_Instance.Coil
```

[ST]
```
COUNTER_FB_M_Instance(Coil:= Var_M0,Preset:=10,ValueIn:=1,
ValueOut:=Var_D10,Status:=Var_M10);
RST(M15,COUNTER_FB_M_Instance.Coil);
```

Cautions

1) Expression in each language of function block
   • Set the instance when using the function block.
   • Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the counter numbers are allocated automatically.
14.10 TIMER_10_FB_M / Timer function blocks

Outline

This function block generates an output when the condition continues for the specified time.
The initial value and setting value of the timer is multiplied by 10 ms.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_10_FB_M</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Instance name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMER_10_FB_M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Coil</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ValueOut</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Preset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ValueIn</td>
<td></td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coil</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>Preset</td>
<td>Timer set value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ValueIn</td>
<td>Initial timer value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ValueOut</td>
<td>Current timer value</td>
<td>ANY16</td>
</tr>
<tr>
<td>Status</td>
<td>Timer output contact</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Function and operation explanation

1) When the execution condition of the input argument Coil turns ON, counting the current value starts.
The timer starts counting from "ValueIn × 10 ms". When it counts up to "Preset × 10 ms", the output argument Status turns ON.
The current measurement value is outputted into ValueOut.

2) When the execution condition of the input argument Coil turns OFF, the current value takes on the value of ValueIn and the output argument Status also turns OFF.

[Structured ladder/FBD]

```
TIMER_10_FB_M_Instance
Var_M0  Var_D10-1
     ValueOut
Var_M10  Var_D10-2
     Status

TIMER_10_FB_M_Instance(Coil:= Var_M0,Preset:= 10,
ValueIn:= 1,ValueOut:= Var_D10,Status:=Var_M10);
```

*1. Var_D10 is a global label and is defined as D10.
*2. Var_M10 is a global label and is defined as M10.

Cautions

1) Expression in each language of function block
   - Set the instance when using the function block.
   - Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the timer numbers are allocated automatically.
Outline
This function block counts the period of time while the condition is satisfied, and generates an output when the timer counts up the specified time.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Structured ladder/FBD</td>
</tr>
<tr>
<td>TIMER_CONT_FB_M</td>
<td>TIMER_CONT_FB_M(Coil, Preset, ValueIn, ValueOut, Status);</td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coil</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>Preset</td>
<td>Timer set value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ValueIn</td>
<td>Initial timer value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ValueOut</td>
<td>Current timer value</td>
<td>ANY16</td>
</tr>
<tr>
<td>Status</td>
<td>Timer output contact</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Function and operation explanation

1) This is a retentive timer that counts the time when the variable is ON. It starts counting the current value when the execution condition of the input argument Coil turns ON.
   The timer starts counting from "ValueIn × 1 ms or 100 ms". When it counts up to "Preset × 1 ms or 100 ms", the output argument Status turns ON.
   The current measurement value is outputted into ValueOut.
   The magnification of ValueIn and Preset (1 ms or 100 ms) is specified depending on the allocation device (retentive timer) during compiling.
   The allocation device (retentive timer)
   - Except for FX3S
     T246 to 249 :1ms
     T250 to 255 :100ms
   - For FX3S
     T128 to T131 :1ms
     T132 to T137 :100ms

2) The condition of measurement ValueOut and output argument ON/OFF status is maintained even if the execution condition of the input argument Coil turns OFF.
   When the execution condition of the input argument Coil turns ON, the timer resume counting from the measurement it holds.

```
[Structured ladder/FBD]

TIMER_CONT_FB_M_Instance(Coil:= Var_M0, Preset:= 200, ValueIn:= 0, ValueOut:= Var_D10, Status:=Var_M10);
```

*1. Var_D10 is a global label and is defined as D10.
*2. Var_M10 is a global label and is defined as M10.
3) When resetting the current value of the retentive timer, reset input variable coil.

   [Structured ladder/FBD]

   TIMER_CONT_FB_M_Instance

   Var_M0 Coil ValueOut Var_D10
   Preset Status Var_M10
   ValueIn

   M15

   RST EN ENO

   d TIMER_CONT_FB_M_Instance.Coil

   [ST]

   TIMER_CONT_FB_M_Instance(Coil:= Var_M0,Preset:=10,ValueIn:=1,
   ValueOut:=Var_D10,Status:=Var_M10);

   RST(M15,TIMER_CONT_FB_M_Instance.Coil);

Cautions

1) Expression in each language of function block
   • Set the instance when using the function block.
   • Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the timer numbers are allocated automatically.
14.12 TIMER_100_FB_M / Timer function blocks

Outline

This function block generates an output when the condition continues for the specified time. The initial value and setting value of the timer is multiplied by 100 ms.

1. Format

<table>
<thead>
<tr>
<th>Function name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_100_FB_M</td>
<td>TIMER_100_FB_M(Coil,Preset,ValueIn,ValueOut,Status);</td>
<td></td>
</tr>
</tbody>
</table>

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coil</td>
<td>Execution condition</td>
<td>Bit</td>
</tr>
<tr>
<td>Preset</td>
<td>Timer set value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>ValueIn</td>
<td>Initial timer value</td>
<td>Word [signed]</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ValueOut</td>
<td>Current timer value</td>
<td>ANY16</td>
</tr>
<tr>
<td>Status</td>
<td>Timer output contact</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Function and operation explanation

1) When the execution condition of the input argument Coil turns ON, counting the current value starts. The timer starts counting from "ValueIn × 100 ms". When it counts up to "Preset × 100 ms", the output argument Status turns ON. The current measurement value is outputted into ValueOut.

2) When the execution condition of the input argument Coil turns OFF, the current value takes on the value of ValueIn and the output argument Status also turns OFF.

```
[Structured ladder/FBD]
TIMER_100_FB_M_Instance(Coil:= Var_M0,Preset:= 10,ValueIn:= Var_D10,ValueOut:= Var_M10,Status:=Var_M10);
```

Timing chart

```
Value of Var_D10

Var_M0

ON

Var_M10

ON

900ms
```

*C1. Var_D10 is a global label and is defined as D10.

*C2. Var_M10 is a global label and is defined as M10.

Cautions

1) Expression in each language of function block
   • Set the instance when using the function block.
   • Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the timer numbers are allocated automatically.
## 15. Operator

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Addition</td>
<td>Section 15.1</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtraction</td>
<td>Section 15.2</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiplication</td>
<td>Section 15.3</td>
</tr>
<tr>
<td>DIV</td>
<td>Division</td>
<td>Section 15.4</td>
</tr>
<tr>
<td>MOD</td>
<td>Modulus operation</td>
<td>Section 15.5</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
<td>Section 15.6</td>
</tr>
<tr>
<td>AND</td>
<td>Logical product</td>
<td>Section 15.7</td>
</tr>
<tr>
<td>OR</td>
<td>Logical sum</td>
<td>Section 15.8</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive logical sum</td>
<td>Section 15.9</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical negation</td>
<td>Section 15.10</td>
</tr>
<tr>
<td>GT</td>
<td>Comparison</td>
<td>Section 15.11</td>
</tr>
<tr>
<td>GE</td>
<td>Comparison</td>
<td>Section 15.12</td>
</tr>
<tr>
<td>EQ</td>
<td>Comparison</td>
<td>Section 15.13</td>
</tr>
<tr>
<td>LE</td>
<td>Comparison</td>
<td>Section 15.14</td>
</tr>
<tr>
<td>LT</td>
<td>Comparison</td>
<td>Section 15.15</td>
</tr>
<tr>
<td>NE</td>
<td>Comparison</td>
<td>Section 15.16</td>
</tr>
</tbody>
</table>
15.1 ADD / Addition

Outline
This operator performs addition using two values (A + B = C), and outputs the operation result.

1. Format

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Data for addition or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output variable</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

Explanation of function and operation

1) This function performs addition (s1 + s2 + ... + sn) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in s1 to sn, and outputs the operation result to a device specified in d. Example: When the data type is word [signed]

\[
\begin{align*}
\text{(s1)} \text{ (Word [signed data])} + \text{(s2)} \text{ (Word [signed data])} & \rightarrow \text{(d)} \text{ (Word [signed data])} \\
1234 + 5678 & \rightarrow 6912
\end{align*}
\]

2) The number of pins for s□ can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions
Refer to Section 7.1.

Program example
In this program, addition is performed using double word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d.

[Structured ladder/FBD]

```
D0   ADD   d
D10  s1   s2
```

[ST]

\[
g\_\text{dint3}:=(g\_\text{dint1})+(g\_\text{dint2});
\]
15.2 SUB / Subtraction

Outline
This operator performs subtraction using two values (A - B = C), and outputs the operation result.

1. Format

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Data to be subtracted or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td></td>
<td>Data for subtraction or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output variable</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

Explanation of function and operation
This function performs subtraction (\(s_1 - s_2\)) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in \(s_1\) and \(s_2\), and outputs the operation result to a device specified in \(d\) using the data type of data stored in devices specified in \(s_1\) and \(s_2\).
Example: When the data type is word [signed]

\[
\begin{array}{c}
\text{12345} - 6789 = 5556
\end{array}
\]

Cautions
Refer to Section 7.2.

Program example
In this program, subtraction is performed using word [signed] data stored in devices specified in \(s_1\) and \(s_2\), and the operation result is output to a device specified in \(d\).

[Structured ladder/FBD]

```
SUB
```

[ST]

\[
g_{int3} := (g_{int1}) - (g_{int2});
\]
15.3 MUL / Multiplication

Outline
This operator performs multiplication using two or more values \(A \times B = C\), and outputs the operation result.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL</td>
<td>[ D0 \times s1 \times s2 = d]</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Data for multiplication or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output variable</td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

Explanation of function and operation

1) This function performs multiplication \((s\times_{\ldots} \times s)\) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in \((s)\) to \((s)\) and the operation result is output to a device specified in \((s)\) using the data type of data stored in devices specified in \((s)\) to \((s)\).

Example: When the data type is word [signed]

```
100 \times 15 \rightarrow 1500
```

2) The number of pins for \((s)\) can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions
Refer to Section 7.3.

Program example

In this program, multiplication is performed using double word [signed] data stored in devices specified in \((s)\) and \((s)\) and the operation result is output to a device specified in \((s)\).

[Structured ladder/FBD]

```
g_{\text{dint}1} \rightarrow \text{MUL} \rightarrow g_{\text{dint}3}
```

[ST]

```
g_{\text{dint}3}:=(g_{\text{dint}1})\times(g_{\text{dint}2});
```
15.4 DIV / Division

Outline
This operator performs division using two values (A / B = C … remainder), and outputs the quotient.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV</td>
<td></td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Data to be divided, or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output variable</td>
<td>Data for division (divisor), or word device which stores such data</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td></td>
<td>Word device which will store the operation result</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

Explanation of function and operation
This function performs division (s1 / s2) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.

Example: When the data type is word [signed]

\[
\begin{array}{c}
\text{5} \\
\text{2}
\end{array} / \begin{array}{c}
\text{2}
\end{array} \rightarrow \begin{array}{c}
\text{(Quotient)}
\end{array}
\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>(Word [signed] data)</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>(Word [signed] data)</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>(Word [signed] data)</td>
<td></td>
</tr>
</tbody>
</table>

Cautions
Refer to Section 7.4.

Error
Refer to Section 7.4.

Program example
In this program, division is performed using double word [signed] data stored in devices specified in g_dint1 and g_dint2, and the operation result is output to a device specified in g_dint3 using the data type of data stored in devices specified in g_dint1 and g_dint2.

[Structured ladder/FBD]

```
g_dint1, g_dint2 ——— DIV ——— g_dint3
```

[ST]

```
g_dint3 := (g_dint1) / (g_dint2);
```
15.5 MOD / Modulus operation

Outline
This operator performs division using two values \((A / B = C \ldots\text{remainder})\), and outputs the remainder.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD</td>
<td>The MOD operator is not available in Structured ladder/FBD language.</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Data to be divided, or word device which stores such data</td>
<td>ANY_INT</td>
</tr>
<tr>
<td></td>
<td>Data for division (divisor), or word device which stores such data</td>
<td>ANY_INT</td>
</tr>
<tr>
<td>Output variable</td>
<td>Word device which will store the operation result</td>
<td>ANY_INT</td>
</tr>
</tbody>
</table>

Explanation of function and operation
This function performs division \((s_1 / s_2)\) using word [signed]/double word [signed] data stored in devices specified in \(s_1\) and \(s_2\), and outputs the remainder to a device specified in \(d\) using the data type of data stored in devices specified in \(s_1\) and \(s_2\).
Example: When the data type is word [signed]

\[
\begin{array}{cccc}
5 & / & 2 & \rightarrow \\
\text{\(s_1\) (Word [signed] data)} & \text{\(s_2\) (Word [signed] data)} & \text{Not output} & \text{\(d\) (Word [signed] data)} \\
\end{array}
\]

Cautions
Refer to Section 7.5.

Error
Refer to Section 7.5.

Program examples
In this program, division is performed using double word [signed] data stored in devices specified in \(s_1\) and \(s_2\), and the remainder is output to a device specified in \(d\) using the data type of data stored in devices specified in \(s_1\) and \(s_2\).

\[
\text{g_dint3:=g_dint1 MOD g_dint2;}
\]
15.6 ** / Exponentiation

Outline

This operator obtains raised result, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>The &quot;**&quot; operator is not available in Structured ladder FBD language.</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>s1</td>
<td>Data to be raised, or word device which stores such data</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>Power data, or word device which stores such data</td>
</tr>
<tr>
<td>Output variable</td>
<td>d</td>
<td>Word device which will store the operation result</td>
</tr>
</tbody>
</table>

Explanation of function and operation

This function raises float (single precision) data stored in a device specified in (s1) (to the power of the value stored in a device specified in (s2)), and outputs the operation result to a device specified in (d).

Cautions

Refer to Section 7.6.

Error

Refer to Section 7.6.

Program examples

In this program, the value stored in a device specified in (s1) is raised to the power of the value stored in a device specified in (s2), and the operation result is output to a device specified in (d) using the data type of data stored in a device specified in (s1).

[ST]
\[
g_{\text{real2}}:=\text{EXPT}(g_{\text{real1}},g_{\text{int1}});\]
15.7 AND / Logical product

Outline
This operator obtains the logical product of two or more bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>M0 AND s1, M10 AND s2</td>
<td>s1 AND s2; Example: M20 := M0 AND M10;</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Device used to obtain the logical product</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>Output variable</td>
<td>Device which will store the operation result</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

Explanation of function and operation
1) This function obtains the logical product using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in s1 to s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 to s2.

Example: When the data type is word [unsigned]/bit string [16-bit]

<table>
<thead>
<tr>
<th>s1</th>
<th>1 1 1 1 1 1 0 0 0 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical product</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>0 0 0 1 0 0 0 1 1 0 1 0 0</td>
</tr>
<tr>
<td>d</td>
<td>0 0 0 1 0 0 0 0 0 0 1 0 0</td>
</tr>
</tbody>
</table>

2) The number of pins for s□ can be changed in the range of 2 to 28.

Cautions
Refer to Section 9.1.

Program examples
In this program, the logical product is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d using the data type of data stored in devices specified in s1 and s2.

[Structured ladder/FBD]

```
g_word1 AND g_word2
```

[ST]

```
g_word3 := (g_word1) AND (g_word2);
```

or

```
g_word3 := (g_word1) & (g_word2);
```
15.8 OR / Logical sum

Outline

This operator obtains the logical sum of two or more bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>!M0 s1 OR !M10 s2 !d</td>
</tr>
<tr>
<td></td>
<td>s1 OR s2; Example: M20 := M0 OR M10;</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>to</td>
<td>Device used to obtain the logical sum</td>
</tr>
<tr>
<td>Output variable</td>
<td></td>
<td>Device which will store the operation result</td>
</tr>
</tbody>
</table>

Explanation of function and operation

1) This function obtains the logical sum using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in $s_1$ to $s_2$, and outputs the operation result to a device specified in $d$ using the data type of data stored in devices specified in $s_1$ to $s_2$.

Example: When the data type is word [unsigned]/bit string [16-bit]

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>Logical sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1110011100011111</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$s_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>110010010110110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011110011100</td>
</tr>
</tbody>
</table>

2) The number of pins for $s_2$ can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

Refer to Section 9.2.

Program examples

In this program, the logical sum is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in $s_1$ and $s_2$, and the operation result is output to a device specified in $d$ using the data type of data stored in devices specified in $s_1$ and $s_2$.

[Structured ladder/FBD]

```plaintext
M0 s1 OR M10 s2 d
s1 OR s2; Example: M20 := M0 OR M10;
```

[ST]

```plaintext
g_word3 := (g_word1) OR (g_word2);
```
15.9 XOR / Exclusive logical sum

Outline

This operator obtains the logical sum of two or more bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>M0 XOR s1 M10 s2 d</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Device used to obtain the exclusive logical sum</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>Output</td>
<td>Device which will store the operation result</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

Explanation of function and operation

1) This function obtains the exclusive logical sum using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in to , and outputs the operation result to a device specified using the data type of data stored in devices specified in to .

Example: When the data type is word [unsigned]/bit string [16-bit]

\[
\begin{align*}
\text{s1} &: \quad 1010101010101010 \\
\text{s2} &: \quad 1011000101011010 \\
\text{d}  &: \quad 1011000101011010
\end{align*}
\]

The number of pins for can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

2) The number of pins for can be changed in the range of 2 to 28.

3) If there are 3 or more , the exclusive logical sum is obtained using the "exclusive logical sum of " and " and " and " and ... In this way, the exclusive logical sum is obtained the required number of times for all input labels ... Example: When the data type is bit

\[
\begin{align*}
\text{s1} &: \quad \text{FALSE} \\
\text{s2} &: \quad \text{TRUE} \\
\text{s3} &: \quad \text{TRUE} \\
\text{s4} &: \quad \text{TRUE} \\
\text{s5} &: \quad \text{TRUE} \\
\end{align*}
\]

After that, the exclusive logical sum is obtained the required number of times.
Cautions

Refer to Section 9.3.

Program examples

In this program, the exclusive logical sum is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in \( s_1 \) and \( s_2 \), and the operation result is output to a device specified in \( d \) using the data type of data stored in devices specified in \( s_1 \) and \( s_2 \).

[Structured ladder/FBD]

\[
g_{\text{word}1} \leftarrow \text{XOR} \rightarrow g_{\text{word}2} \rightarrow g_{\text{word}3}
\]

[ST]

\[
g_{\text{word}3} = (g_{\text{word}1}) \oplus (g_{\text{word}2});
\]
15.10 NOT / Logical negation

Outline

This operator obtains the logical negation of bits, and outputs it.

1. Format

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>The NOT operator is not available in Structured ladder/FBD language.</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Device used to obtain the logical negation</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>Output variable</td>
<td>Device which will store the operation result</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

Explanation of function and operation

This function obtains the logical negation using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in a device specified in \( s \), and outputs the operation result to a device specified in \( d \) using the data type of data stored in a device specified in \( s \).

Example: When the data type is word [unsigned]/bit string [16-bit]

\[
\begin{array}{cccccccc}
\text{g_word1} & \text{g_word2} & \text{Logical negation} \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Cautions

Refer to Section 9.4.

Program examples

In this program, the logical negation is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in a device specified in \( s \), and the operation result is output to a device specified in \( d \) using the data type of data stored in a device specified in \( s \).

[ST]

g_word2 := NOT(g_word1);
15.11 GT / Comparison

Outline

This operator compares data with regard to "> (larger)".

1. Format

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

Explanation of function and operation

1) This function compares the contents of devices specified in □ to □, and outputs the operation result expressed as the bit type data to a device specified in □.
   This function executes comparison \[ s_1 > s_2 \] & \[ s_2 > s_3 \] & ... & \[ s_{n-1} > s_n \].
   a) This function outputs "TRUE" when all comparison results are "s□ > s□".
   b) This function outputs "FALSE" when any comparison result is "s□ ≤ s□".

2) The number of pins for □ can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

Refer to Section 11.1.

Program examples

In this program, the contents of devices specified in □ and □ are compared, and the operation result is output to a device specified in □.

[Structured ladder/FBD]

```
    g_int1 -- GT -- g_bool2
    g_int2
```

[ST]

```
g_bool2:=(g_int1)>(g_int2);
```
Outline

This operator compares data with regard to “≥ (larger or equal)”.

1. Format

In explanation of operators, the input variable is described as “s□” and the output variable is described as “d”.

2. Set data

2) The number of pins for s□ can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Explanation of function and operation

1) This function compares the contents of devices specified in s□ to s□, and outputs the operation result expressed as the bit type data to a device specified in d.

This function executes comparison [s□ ≥ s□] & [s□ ≥ s□] & ... & [s□ ≥ s□].

a) This function outputs "TRUE" when all comparison results are "≥ s□".

b) This function outputs "FALSE" when any comparison result is "< s□".

2) The number of pins for s□ can be changed in the range of 2 to 28.

Cautions

Refer to Section 11.2.

Program examples

In this program, the contents of devices specified in s□ and s□ are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]

```
<table>
<thead>
<tr>
<th>g_int1</th>
<th>GE</th>
<th>g_bool2</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_int2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

[ST]

```
g_bool2:=(g_int1)>=g_int2;
```
15.13 EQ / Comparison

Outline
This operator compares data with regard to "= (equal)".

1. Format

In the explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

The number of pins for "□" can be changed in the range of 2 to 28.

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

Explanation of function and operation

1) This function compares the contents of devices specified in "□□" to "□□", and outputs the operation result expressed as the bit type data to a device specified in "□□".
   This function executes comparison [s1 = s2] & [s2 = s3] & ... & [sn-1 = sn].
   a) This function outputs "TRUE" when all comparison results are "s1 = s2 = s3 = ... = sn".
   b) This function outputs "FALSE" when any comparison result is "s1 ≠ s2 ≠ s3 ≠ ... ≠ sn".
2) The number of pins for "□" can be changed in the range of 2 to 28.
   → Refer to Section 3. Function Construction

Cautions
Refer to Section 11.3.

Program examples
In this program, the contents of devices specified in "□□" and "□□" are compared, and the operation result is output to a device specified in "□□".

[Structured ladder/FBD]

```
g_int1
EQ
```

[ST]

```
g_bool2:=(g_int1)=(g_int2);
```
15.14 LE / Comparison

Outline
This operator compares data with regard to "≤ (smaller or equal)".

1. Format
In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE</td>
<td>s1 &lt;= s2; Example: M0:=D0&lt;=D10;</td>
</tr>
</tbody>
</table>

1) This function compares the contents of devices specified in to , and outputs the operation result expressed as the bit type data to a device specified in .

2) The number of pins for can be changed in the range of 2 to 28.

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>s1 to s28</td>
<td>Compared data, or word device which stores such data</td>
</tr>
<tr>
<td>Output variable</td>
<td>d</td>
<td>Device which will store the comparison result</td>
</tr>
</tbody>
</table>

Explanation of function and operation

1) This function compares the contents of devices specified in to , and outputs the operation result expressed as the bit type data to a device specified in .

This function executes comparison [s1 ≤ s2] & [s2 ≤ s3] & … & [s[n-1] ≤ s[n]].

a) This function outputs "TRUE" when all comparison results are "s[n] ≤ s[n]".

b) This function outputs "FALSE" when any comparison result is "s[n] > s[n]".

Cautions
Refer to Section 11.4.

Program examples
In this program, the contents of devices specified in and are compared, and the operation result is output to a device specified in .

[Structured ladder/FBD]

[ST]

```plaintext
LE

M0:=D0<=D10;
```

```plaintext
M0:=D0<=D10;
```

```plaintext
M0:=D0<=D10;
```

```plaintext
M0:=D0<=D10;
```
15.15 LT / Comparison

Outline

This operator compares data with regard to "< (smaller)".

1. Format

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D0 \(\text{s1} < \text{d} \) M0
D10 \(\text{s2} \)

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>Compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>Output variable</td>
<td>Device which will store the comparison result</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Explanation of function and operation

1) This function compares the contents of devices specified in \( \text{s1} \) to \( \text{s2} \), and outputs the operation result expressed as the bit type data to a device specified in \( \text{d} \).

This function executes comparison \( \text{s1} < \text{s2} \) \& \( \text{s2} < \text{s3} \) \& … \& \( \text{s_n-1} < \text{s_n} \).

a) This function outputs "TRUE" when all comparison results are "\( \text{s_n} < \text{s_n} \)".

b) This function outputs "FALSE" when any comparison result is "\( \text{s_n} \geq \text{s_n} \)".

2) The number of pins for \( \text{s} \) can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

Refer to Section 11.5.

Program examples

In this program, the contents of devices specified in \( \text{s1} \) and \( \text{s2} \) are compared, and the operation result is output to a device specified in \( \text{d} \).

[Structured ladder/FBD]

```
g_int1 LT g_int2 g_bool2
```

[ST]

```
g_bool2:=(g_int1)<(g_int2);
```
15.16 NE / Comparison

Outline
This operator compares data with regard to "≠ (unequal)".

1. Format
In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Expression in each language</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE</td>
<td>D0 s1 NE d M0</td>
</tr>
<tr>
<td></td>
<td>D10 s2</td>
</tr>
</tbody>
</table>

In explanation of operators, the input variable is described as "s□" and the output variable is described as "d".

2. Set data

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input variable</td>
<td>[s1 to s2] compared data, or word device which stores such data</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>Output variable</td>
<td>[d] device which will store the comparison result</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Explaination of function and operation
This function compares the contents of devices specified in (s1) and (s2), and outputs the operation result expressed as the bit type data to a device specified in (d).

This function executes comparison \([s1] ≠ [s2]\).

a) This function outputs "TRUE" when in the case of "[s1] ≠ [s2]"

b) This function outputs "FALSE" when in the case of "[s1] = [s2]"

Cautions
Refer to Section 11.6.

Program examples
In this program, the contents of devices specified in (s1) and (s2) are compared, and the operation result is output to a device specified in (d).

[Structured ladder/FBD]
```
g_int1
g_int2
   NE
```

[ST]
g_bool2:=(g_int1)<>(g_int2);
The table below shows the correspondence between devices and addresses.

<table>
<thead>
<tr>
<th>Device</th>
<th>Notation</th>
<th>Example of correspondence between device and address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input relay</td>
<td>X</td>
<td>%IXn</td>
</tr>
<tr>
<td>Output relay</td>
<td>Y</td>
<td>%QXn</td>
</tr>
<tr>
<td>Auxiliary relay</td>
<td>M</td>
<td>%MX0.n</td>
</tr>
<tr>
<td>Timer Contact</td>
<td>TS</td>
<td>%MX3.n</td>
</tr>
<tr>
<td>Timer Coil</td>
<td>TC</td>
<td>%MX5.n</td>
</tr>
<tr>
<td>Timer Current value</td>
<td>TN</td>
<td>%MW3.n</td>
</tr>
<tr>
<td>Counter Contact</td>
<td>CS</td>
<td>%MX4.n</td>
</tr>
<tr>
<td>Counter Coil</td>
<td>CC</td>
<td>%MX6.n</td>
</tr>
<tr>
<td>Counter Current value</td>
<td>CN</td>
<td>%MW4.n</td>
</tr>
<tr>
<td>Data register</td>
<td>D</td>
<td>%MW0.n</td>
</tr>
<tr>
<td>Intelligent function unit device</td>
<td>G</td>
<td>Ux</td>
</tr>
<tr>
<td>Extension register</td>
<td>R</td>
<td>Rn</td>
</tr>
<tr>
<td>Extension file register</td>
<td>ER</td>
<td>ERn</td>
</tr>
<tr>
<td>Pointer</td>
<td>P</td>
<td>Pn</td>
</tr>
<tr>
<td>Interrupt pointer</td>
<td>I</td>
<td>In</td>
</tr>
<tr>
<td>Nesting</td>
<td>N</td>
<td>Nn</td>
</tr>
<tr>
<td>Index register</td>
<td>Z</td>
<td>Zn</td>
</tr>
<tr>
<td>State</td>
<td>S</td>
<td>Sn</td>
</tr>
</tbody>
</table>
## Appendix B: Function/Operator List [by Type/in Alphabetic Order]

### Appendix B-1 [By type]

1. **Type Conversion Functions**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_INT(E)</td>
<td>Converts bit data into word [signed] data.</td>
<td>40</td>
</tr>
<tr>
<td>BOOL_TO_DINT(E)</td>
<td>Converts bit data into double word [signed] data.</td>
<td>42</td>
</tr>
<tr>
<td>BOOL_TO_STR(E)</td>
<td>Converts bit data into string data.</td>
<td>44</td>
</tr>
<tr>
<td>BOOL_TO_WORD(E)</td>
<td>Converts bit data into word [unsigned]/bit string [16-bit] data.</td>
<td>46</td>
</tr>
<tr>
<td>BOOL_TO_DWORD(E)</td>
<td>Converts bit data into double word [unsigned]/bit string [32-bit] data.</td>
<td>48</td>
</tr>
<tr>
<td>BOOL_TO_TIME(E)</td>
<td>Converts bit data into time data.</td>
<td>50</td>
</tr>
<tr>
<td>INT_TO_DINT(E)</td>
<td>Converts word [signed] data into double word [signed] data.</td>
<td>52</td>
</tr>
<tr>
<td>DINT_TO_INT(E)</td>
<td>Converts double word [signed] data into word [signed] data.</td>
<td>54</td>
</tr>
<tr>
<td>INT_TO_BOOL(E)</td>
<td>Converts word [signed] data into bit data.</td>
<td>56</td>
</tr>
<tr>
<td>DINT_TO_BOOL(E)</td>
<td>Converts double word [signed] data into bit data.</td>
<td>58</td>
</tr>
<tr>
<td>INT_TO_REAL(E)</td>
<td>Converts word [signed] data into float (single precision) data.</td>
<td>60</td>
</tr>
<tr>
<td>DINT_TO_REAL(E)</td>
<td>Converts double word [signed] data into float (single precision) data.</td>
<td>62</td>
</tr>
<tr>
<td>INT_TO_STR(E)</td>
<td>Converts word [signed] data into string data.</td>
<td>64</td>
</tr>
<tr>
<td>DINT_TO_STR(E)</td>
<td>Converts double word [signed] data into string data.</td>
<td>66</td>
</tr>
<tr>
<td>INT_TO_TIME(E)</td>
<td>Converts word [signed] data into word [unsigned]/bit string [16-bit] data.</td>
<td>68</td>
</tr>
<tr>
<td>DINT_TO_TIME(E)</td>
<td>Converts double word [signed] data into double word [unsigned]/bit string [32-bit] data.</td>
<td>70</td>
</tr>
<tr>
<td>INT_TO_BCD(E)</td>
<td>Converts word [signed] data into BCD data.</td>
<td>72</td>
</tr>
<tr>
<td>DINT_TO_BCD(E)</td>
<td>Converts double word [signed] data into double word [unsigned]/bit string [32-bit] data.</td>
<td>74</td>
</tr>
<tr>
<td>INT_TO_BCDINT(E)</td>
<td>Converts word [signed] data into BCD data.</td>
<td>76</td>
</tr>
<tr>
<td>DINT_TO_BCDINT(E)</td>
<td>Converts double word [signed] data into double word [unsigned]/bit string [32-bit] data.</td>
<td>78</td>
</tr>
<tr>
<td>INT_TO_TIME(E)</td>
<td>Converts word [signed] data into time data.</td>
<td>80</td>
</tr>
<tr>
<td>DINT_TO_TIME(E)</td>
<td>Converts double word [signed] data into time data.</td>
<td>82</td>
</tr>
<tr>
<td>REAL_TO_INT(E)</td>
<td>Converts float (single precision) data into word [signed] data.</td>
<td>84</td>
</tr>
<tr>
<td>REAL_TO_DINT(E)</td>
<td>Converts float (single precision) data into double word [signed] data.</td>
<td>86</td>
</tr>
</tbody>
</table>

### Appendix B-2 [By type]

2. **Integer Comparison Functions**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>Greater than</td>
<td>91</td>
</tr>
<tr>
<td>LE</td>
<td>Less than or equal to</td>
<td>93</td>
</tr>
<tr>
<td>EQ</td>
<td>Equal to</td>
<td>95</td>
</tr>
</tbody>
</table>

### Appendix B-3 [By type]

3. **Logical Functions**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>Logical OR</td>
<td>97</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND</td>
<td>99</td>
</tr>
<tr>
<td>NOT</td>
<td>Logical NOT</td>
<td>101</td>
</tr>
</tbody>
</table>

### Appendix B-4 [By type]

4. **Mathematical Functions**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Addition</td>
<td>103</td>
</tr>
<tr>
<td>SUB</td>
<td>Subtraction</td>
<td>105</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiplication</td>
<td>107</td>
</tr>
<tr>
<td>DIV</td>
<td>Division</td>
<td>109</td>
</tr>
</tbody>
</table>

### Appendix B-5 [By type]

5. **String Functions**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR_TO_BOOL(E)</td>
<td>Converts string data into bit data.</td>
<td>111</td>
</tr>
<tr>
<td>STR_TO_INT(E)</td>
<td>Converts string data into word [signed] data.</td>
<td>113</td>
</tr>
<tr>
<td>STR_TO_DINT(E)</td>
<td>Converts string data into double word [signed] data.</td>
<td>115</td>
</tr>
<tr>
<td>STR_TO_REAL(E)</td>
<td>Converts string data into float (single precision) data.</td>
<td>117</td>
</tr>
<tr>
<td>STR_TO_BCDINT(E)</td>
<td>Converts BCD data into double word [signed] data.</td>
<td>119</td>
</tr>
<tr>
<td>STR_TO_BCD(E)</td>
<td>Converts BCD data into BCD data.</td>
<td>121</td>
</tr>
<tr>
<td>TIME_TO_BOOL(E)</td>
<td>Converts time data into bit data.</td>
<td>123</td>
</tr>
<tr>
<td>TIME_TO_INT(E)</td>
<td>Converts time data into word [signed] data.</td>
<td>125</td>
</tr>
<tr>
<td>TIME_TO_DINT(E)</td>
<td>Converts time data into double word [signed] data.</td>
<td>127</td>
</tr>
<tr>
<td>TIME_TO_BCDINT(E)</td>
<td>Converts time data into BCD data.</td>
<td>129</td>
</tr>
<tr>
<td>TIME_TO_BCD(E)</td>
<td>Converts time data into double word [signed] data.</td>
<td>131</td>
</tr>
<tr>
<td>TIME_TO_STR(E)</td>
<td>Converts time data into string data.</td>
<td>133</td>
</tr>
<tr>
<td>TIME_TO_WORD(E)</td>
<td>Converts time data into word [unsigned]/bit string [16-bit] data.</td>
<td>135</td>
</tr>
</tbody>
</table>
Appendix B: Application Function List

2. Standard Functions Of One Numeric Variable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(_E)</td>
<td>Obtains the absolute value.</td>
<td>158</td>
</tr>
</tbody>
</table>

3. Standard Arithmetic Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_E</td>
<td>Adds data. (Number of pins variable)</td>
<td>161</td>
</tr>
<tr>
<td>SUB_E</td>
<td>Subtracts data.</td>
<td>163</td>
</tr>
<tr>
<td>MUL_E</td>
<td>Multiplies data. (Number of pins variable)</td>
<td>165</td>
</tr>
<tr>
<td>DIV_E</td>
<td>Divides data. (Number of pins variable)</td>
<td>167</td>
</tr>
<tr>
<td>MOD(_E)</td>
<td>Divides data. (Number of pins variable)</td>
<td>169</td>
</tr>
<tr>
<td>EXPT(_E)</td>
<td>Obtains the raised result.</td>
<td>171</td>
</tr>
<tr>
<td>MOVE(_E)</td>
<td>Transfers data.</td>
<td>173</td>
</tr>
</tbody>
</table>

4. Standard Bit Shift Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHL(_E)</td>
<td>Shifts bits leftward.</td>
<td>176</td>
</tr>
<tr>
<td>SHR(_E)</td>
<td>Shifts bits rightward.</td>
<td>178</td>
</tr>
</tbody>
</table>

5. Standard Bitwise Boolean Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND(_E)</td>
<td>Obtains the logical product.</td>
<td>181</td>
</tr>
<tr>
<td>OR(_E)</td>
<td>Obtains the logical sum.</td>
<td>183</td>
</tr>
<tr>
<td>XOR(_E)</td>
<td>Obtains the exclusive logical sum.</td>
<td>185</td>
</tr>
<tr>
<td>NOT(_E)</td>
<td>Obtains the logical not.</td>
<td>187</td>
</tr>
</tbody>
</table>

6. Standard Selection Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL(_E)</td>
<td>Selects data in accordance with the input condition.</td>
<td>190</td>
</tr>
<tr>
<td>MAXIMUM(_E)</td>
<td>Searches the maximum value.</td>
<td>192</td>
</tr>
<tr>
<td>MINIMUM(_E)</td>
<td>Searches the minimum value.</td>
<td>194</td>
</tr>
<tr>
<td>LIMITATION(_E)</td>
<td>Judges whether data is located within the range between the upper limit value and the lower limit value.</td>
<td>196</td>
</tr>
<tr>
<td>MUX(_E)</td>
<td>Selects data, and outputs it.</td>
<td>198</td>
</tr>
</tbody>
</table>

7. Standard Comparison Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT(_E)</td>
<td>Compares data with regard to &quot;(larger)&quot;.</td>
<td>201</td>
</tr>
<tr>
<td>GE(_E)</td>
<td>Compares data with regard to &quot; (larger or equal)&quot;.</td>
<td>203</td>
</tr>
<tr>
<td>EQ(_E)</td>
<td>Compares data with regard to &quot;= (equal)&quot;.</td>
<td>205</td>
</tr>
<tr>
<td>LE(_E)</td>
<td>Compares data with regard to &quot;≤ (smaller or equal)&quot;.</td>
<td>207</td>
</tr>
<tr>
<td>LT(_E)</td>
<td>Compares data with regard to &quot;&lt; (smaller)&quot;.</td>
<td>209</td>
</tr>
<tr>
<td>NE(_E)</td>
<td>Compares data with regard to &quot;≠ (unequal)&quot;.</td>
<td>211</td>
</tr>
</tbody>
</table>

8. Standard Character String Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID(_E)</td>
<td>Obtains a character string from a specified position.</td>
<td>214</td>
</tr>
<tr>
<td>CONCAT(_E)</td>
<td>Connects character strings.</td>
<td>217</td>
</tr>
<tr>
<td>INSERT(_E)</td>
<td>Inserts a character string.</td>
<td>219</td>
</tr>
<tr>
<td>DELETE(_E)</td>
<td>Deletes a character string.</td>
<td>222</td>
</tr>
<tr>
<td>REPLACE(_E)</td>
<td>Replaces a character string.</td>
<td>224</td>
</tr>
<tr>
<td>FIND(_E)</td>
<td>Searches a character string.</td>
<td>227</td>
</tr>
</tbody>
</table>
## 9. Functions Of Time Data Types

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TIME (_E)</td>
<td>Adds time data.</td>
<td>231</td>
</tr>
<tr>
<td>SUB_TIME (_E)</td>
<td>Subtracts time data.</td>
<td>233</td>
</tr>
<tr>
<td>MUL_TIME (_E)</td>
<td>Multiplies time data.</td>
<td>235</td>
</tr>
<tr>
<td>DIV_TIME (_E)</td>
<td>Divides time data.</td>
<td>237</td>
</tr>
</tbody>
</table>

## 10. Standard Function Blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TRIG (_E)</td>
<td>Detects the rising edge of a signal, and outputs pulse signal.</td>
<td>240</td>
</tr>
<tr>
<td>F_TRIG (_E)</td>
<td>Detects the falling edge of a signal, and outputs pulse signal.</td>
<td>242</td>
</tr>
<tr>
<td>CTU(_E)</td>
<td>Counts up the number of times of rising of a signal.</td>
<td>244</td>
</tr>
<tr>
<td>CTD(_E)</td>
<td>Counts down the number of times of rising of a signal.</td>
<td>246</td>
</tr>
<tr>
<td>CTUD(_E)</td>
<td>Counts up/down the number of times of rising of a signal.</td>
<td>248</td>
</tr>
<tr>
<td>TP(_E)</td>
<td>Keeps ON a signal during specified time duration.</td>
<td>250</td>
</tr>
<tr>
<td>TP_10(_E)</td>
<td>Keeps OFF a signal during specified time duration.</td>
<td>252</td>
</tr>
<tr>
<td>TON(_E)</td>
<td>Turns OFF the output signal at specified time after the input signal turned OFF.</td>
<td>254</td>
</tr>
<tr>
<td>TOF(_E)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNTER_FB_M</td>
<td>Counter drive</td>
<td>256</td>
</tr>
<tr>
<td>TIMER_10_FB_M</td>
<td>10ms timer drive</td>
<td>258</td>
</tr>
<tr>
<td>TIMER_Counter_FB_M</td>
<td>Retentive timer drive</td>
<td>259</td>
</tr>
<tr>
<td>TIMER_100_FB_M</td>
<td>100ms timer drive</td>
<td>261</td>
</tr>
</tbody>
</table>

## 12. Operator (Logical operations)

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND &amp;</td>
<td>Obtains the logical product. (Number of pins variable)</td>
<td>269</td>
</tr>
<tr>
<td>OR OR</td>
<td>Obtains the logical sum. (Number of pins variable)</td>
<td>270</td>
</tr>
<tr>
<td>XOR XOR</td>
<td>Obtains the exclusive logical sum. (Number of pins variable)</td>
<td>271</td>
</tr>
<tr>
<td>NOT</td>
<td>Obtains the logical not.</td>
<td>273</td>
</tr>
</tbody>
</table>

## 13. Operator (Comparison operations)

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT &gt;</td>
<td>Compares data with regard to “&gt; (larger).” (Number of pins variable)</td>
<td>274</td>
</tr>
<tr>
<td>GE &gt;=</td>
<td>Compares data with regard to “≥ (larger or equal).” (Number of pins variable)</td>
<td>275</td>
</tr>
<tr>
<td>EQ =</td>
<td>Compares data with regard to “= (equal).” (Number of pins variable)</td>
<td>276</td>
</tr>
<tr>
<td>LE &lt;=</td>
<td>Compares data with regard to “≤ (smaller or equal).” (Number of pins variable)</td>
<td>277</td>
</tr>
<tr>
<td>LT &lt;</td>
<td>Compares data with regard to “&lt; (smaller).” (Number of pins variable)</td>
<td>278</td>
</tr>
<tr>
<td>NE &lt;&gt;</td>
<td>Compares data with regard to “≠ (unequal).”</td>
<td>279</td>
</tr>
</tbody>
</table>

## 11. Operator (Arithmetic operations)

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD +</td>
<td>Adds data. (Number of pins variable)</td>
<td>263</td>
</tr>
<tr>
<td>SUB -</td>
<td>Subtracts data.</td>
<td>264</td>
</tr>
<tr>
<td>MUL *</td>
<td>Multiplies data. (Number of pins variable)</td>
<td>265</td>
</tr>
<tr>
<td>DIV /</td>
<td>Divides data (, and outputs the quotient).</td>
<td>266</td>
</tr>
<tr>
<td>MOD</td>
<td>Divides data (, and outputs the remainder).</td>
<td>267</td>
</tr>
<tr>
<td>**</td>
<td>Obtains the raised result</td>
<td>268</td>
</tr>
</tbody>
</table>
## Appendix B-2 [In alphabetic order]

### Functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(E)</td>
<td>Obtains the absolute value.</td>
<td>158</td>
</tr>
<tr>
<td>ADD_TIME(_E)</td>
<td>Adds time data.</td>
<td>231</td>
</tr>
<tr>
<td>ADD_E</td>
<td>Adds data. (Number of pins variable)</td>
<td>161</td>
</tr>
<tr>
<td>AND_E</td>
<td>Obtains the logical product. (Number of pins variable)</td>
<td>181</td>
</tr>
<tr>
<td>BCD_TO_DINT(E)</td>
<td>Converts BCD data into double word [signed] data.</td>
<td>124</td>
</tr>
<tr>
<td>BCD_TO_I_NT(E)</td>
<td>Converts BCD data into word [signed] data.</td>
<td>122</td>
</tr>
<tr>
<td>BITARR_T_O_DINT(E)</td>
<td>Converts specified number of bits of a bit array into double word [signed] data or double word [unsigned]/bit string [32-bit] data.</td>
<td>140</td>
</tr>
<tr>
<td>BITARR_T_O_INT(E)</td>
<td>Converts specified number of bits of a bit array into word [signed] data or word [unsigned]/bit string [16-bit] data.</td>
<td>138</td>
</tr>
<tr>
<td>BOOL_TO_DINT(E)</td>
<td>Converts bit data into double word [signed] data.</td>
<td>42</td>
</tr>
<tr>
<td>BOOL_TO_DWORD(E)</td>
<td>Converts bit data into double word [unsigned]/bit string [32-bit] data.</td>
<td>48</td>
</tr>
<tr>
<td>BOOL_TO_INT(E)</td>
<td>Converts bit data into word [signed] data.</td>
<td>40</td>
</tr>
<tr>
<td>BOOL_TO_STR(E)</td>
<td>Converts bit data into string data.</td>
<td>44</td>
</tr>
<tr>
<td>BOOL_TO_TIME(E)</td>
<td>Converts bit data into time data.</td>
<td>50</td>
</tr>
<tr>
<td>BOOL_TO_WORD(E)</td>
<td>Converts bit data into word [unsigned]/bit string [16-bit] data.</td>
<td>46</td>
</tr>
<tr>
<td>CONCAT(E)</td>
<td>Connects character strings. (Number of pins variable)</td>
<td>217</td>
</tr>
<tr>
<td>COUNTER_FB_M</td>
<td>Counter drive</td>
<td>256</td>
</tr>
<tr>
<td>CPY_BITARR(E)</td>
<td>Copies specified number of bits of a bit array.</td>
<td>146</td>
</tr>
<tr>
<td>CPY_BIT_OF_INT(E)</td>
<td>Copies a specified bit of word [signed] data to a specified bit of another word [signed] data.</td>
<td>152</td>
</tr>
<tr>
<td>CTD(E)</td>
<td>Counts down the number of times of rising of a signal.</td>
<td>246</td>
</tr>
<tr>
<td>CTUD(E)</td>
<td>Counts up/down the number of times of rising of a signal.</td>
<td>248</td>
</tr>
<tr>
<td>CTU(E)</td>
<td>Counts up the number of times of rising of a signal.</td>
<td>244</td>
</tr>
<tr>
<td>DELETE(E)</td>
<td>Deletes a character string.</td>
<td>222</td>
</tr>
<tr>
<td>DINT_TO_BCD(E)</td>
<td>Converts double word [signed] data into BCD data.</td>
<td>78</td>
</tr>
<tr>
<td>DINT_TO_BITARR(E)</td>
<td>Outputs low-order “n” bits of double word [signed] data or double word [unsigned]/bit string [32-bit] data to a bit array.</td>
<td>144</td>
</tr>
<tr>
<td>DINT_TO_BOOL(E)</td>
<td>Converts double word [signed] data into bit data.</td>
<td>58</td>
</tr>
<tr>
<td>DINT_TO_DWORD(E)</td>
<td>Converts double word [signed] data into double word [unsigned]/bit string [32-bit] data.</td>
<td>74</td>
</tr>
<tr>
<td>DIV_E</td>
<td>Divides data. (Number of pins variable)</td>
<td>217</td>
</tr>
<tr>
<td>DIV_TIME(_E)</td>
<td>Divides time data.</td>
<td>237</td>
</tr>
<tr>
<td>DIV_E</td>
<td>Divides data (and outputs the quotient).</td>
<td>167</td>
</tr>
<tr>
<td>DWORD_T_O_BOOL(E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into bit data.</td>
<td>93</td>
</tr>
<tr>
<td>DWORD_T_O_DINT(E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into double word [signed] data.</td>
<td>101</td>
</tr>
<tr>
<td>DWORD_T_O_INT(E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into word [signed] data.</td>
<td>99</td>
</tr>
<tr>
<td>DWORD_T_O_TIME(E)</td>
<td>Converts double word [signed] data into double word [signed] data.</td>
<td>109</td>
</tr>
<tr>
<td>DWORD_T_O_WORD(E)</td>
<td>Converts double word [unsigned]/bit string [32-bit] data into word [unsigned]/bit string [16-bit] data.</td>
<td>105</td>
</tr>
<tr>
<td>EQ_E</td>
<td>Compares data with regard to “=” (equal). (Number of pins variable)</td>
<td>205</td>
</tr>
<tr>
<td>EXPT(E)</td>
<td>Obtains the raised result.</td>
<td>171</td>
</tr>
<tr>
<td>FIND(E)</td>
<td>Searches a character string.</td>
<td>227</td>
</tr>
<tr>
<td>F_TRIG(E)</td>
<td>Detects the falling edge of a signal, and outputs pulse signal.</td>
<td>242</td>
</tr>
<tr>
<td>GE_E</td>
<td>Compares data with regard to “&gt;” (larger or equal). (Number of pins variable)</td>
<td>203</td>
</tr>
<tr>
<td>GET_BIT_OF_INT(E)</td>
<td>Reads the value of a specified bit of word [signed] data.</td>
<td>148</td>
</tr>
<tr>
<td>GET_BOOL_ADDR</td>
<td>Outputs start data as bit data.</td>
<td>154</td>
</tr>
<tr>
<td>GET_INT_ADDR</td>
<td>Outputs start data as word [signed] data.</td>
<td>155</td>
</tr>
<tr>
<td>GET_WORD_ADDR</td>
<td>Outputs start data as word [unsigned]/bit string [16-bit] data.</td>
<td>156</td>
</tr>
<tr>
<td>GT_E</td>
<td>Compares data with regard to “&gt;” (larger). (Number of pins variable)</td>
<td>201</td>
</tr>
<tr>
<td>INSERT(E)</td>
<td>Inserts a character string.</td>
<td>219</td>
</tr>
<tr>
<td>INT_TO_BCD(E)</td>
<td>Converts word [signed] data into BCD data.</td>
<td>78</td>
</tr>
<tr>
<td>INT_TO_BITARR(E)</td>
<td>Outputs low-order “n” bits of word [signed] data or word [unsigned]/bit string [16-bit] data to a bit array.</td>
<td>142</td>
</tr>
<tr>
<td>INT_TO_BOOL(E)</td>
<td>Converts word [signed] data into bit data.</td>
<td>56</td>
</tr>
<tr>
<td>INT_TO_DINT(E)</td>
<td>Converts word [signed] data into double word [signed] data.</td>
<td>52</td>
</tr>
</tbody>
</table>
### Application Function List

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_TO_D(WORD(E))</td>
<td>Converts word [signed] data into double word [unsigned/bit string [32-bit] data.</td>
<td>72</td>
</tr>
<tr>
<td>INT_TO_REAL(E)</td>
<td>Converts word [signed] data into float [single precision] data.</td>
<td>60</td>
</tr>
<tr>
<td>INT_TO_STRING(E)</td>
<td>Converts word [signed] data into string data.</td>
<td>64</td>
</tr>
<tr>
<td>INT_TO_TIME(E)</td>
<td>Converts word [signed] data into time data.</td>
<td>80</td>
</tr>
<tr>
<td>INT_TO_WORD(E)</td>
<td>Converts word [signed] data into word [unsigned/bit string [16-bit] data.</td>
<td>68</td>
</tr>
<tr>
<td>LE_E</td>
<td>Compares data with regard to &quot;≤ (smaller or equal)&quot;. (Number of pins variable)</td>
<td>207</td>
</tr>
<tr>
<td>LIMITATION(E)</td>
<td>Judges whether data is located within the range between the upper limit value and the lower limit value.</td>
<td>196</td>
</tr>
<tr>
<td>LT_E</td>
<td>Compares data with regard to &quot;&lt; (smaller)&quot;. (Number of pins variable)</td>
<td>209</td>
</tr>
<tr>
<td>MAXIMUM(E)</td>
<td>Searches the maximum value. (Number of pins variable)</td>
<td>192</td>
</tr>
<tr>
<td>MID(E)</td>
<td>Obtains a character string from a specified position.</td>
<td>214</td>
</tr>
<tr>
<td>MINIMUM(E)</td>
<td>Searches the minimum value. (Number of pins variable)</td>
<td>194</td>
</tr>
<tr>
<td>MOD(E)</td>
<td>Divides data, and outputs the remainder. (Number of pins variable)</td>
<td>169</td>
</tr>
<tr>
<td>MOVE(E)</td>
<td>Transfers data.</td>
<td>173</td>
</tr>
<tr>
<td>MUL_TIME(E)</td>
<td>Multiplies time data.</td>
<td>235</td>
</tr>
<tr>
<td>MUX(E)</td>
<td>Selects data, and outputs it. (Number of pins variable)</td>
<td>198</td>
</tr>
<tr>
<td>NE_E</td>
<td>Compares data with regard to “≠ (unequal)”.</td>
<td>211</td>
</tr>
<tr>
<td>NOT(E)</td>
<td>Obtains the logical not.</td>
<td>187</td>
</tr>
<tr>
<td>OR_E</td>
<td>Obtains the logical sum. (Number of pins variable)</td>
<td>183</td>
</tr>
<tr>
<td>REAL_TO_DINT(E)</td>
<td>Converts float [single precision] data into double word [signed] data.</td>
<td>86</td>
</tr>
<tr>
<td>REAL_TO_INT(E)</td>
<td>Converts float [single precision] data into word [signed] data.</td>
<td>84</td>
</tr>
<tr>
<td>REAL_TO_STRING(E)</td>
<td>Converts float [single precision] data into string data.</td>
<td>88</td>
</tr>
<tr>
<td>REPLACE(E)</td>
<td>Replaces a character string.</td>
<td>224</td>
</tr>
<tr>
<td>R_TRIG(E)</td>
<td>Detects the rising edge of a signal, and outputs pulse signal.</td>
<td>240</td>
</tr>
<tr>
<td>SEL(E)</td>
<td>Selects data in accordance with the input condition.</td>
<td>190</td>
</tr>
<tr>
<td>SET_BIT_OF_INT(E)</td>
<td>Writes a value to a specified bit of word [signed] data.</td>
<td>150</td>
</tr>
<tr>
<td>SHR(E)</td>
<td>Shifts bits rightward.</td>
<td>178</td>
</tr>
<tr>
<td>STR_TO_BOOLEAN(E)</td>
<td>Converts string data into bit data.</td>
<td>91</td>
</tr>
<tr>
<td>STR_TO_DINT(E)</td>
<td>Converts string data into double word [signed] data.</td>
<td>115</td>
</tr>
<tr>
<td>STR_TO_DOUBLE(E)</td>
<td>Converts string data into double word [signed] data.</td>
<td>130</td>
</tr>
<tr>
<td>STR_TO_DWORD(E)</td>
<td>Converts string data into double word [unsigned/bit string [32-bit] data.</td>
<td>103</td>
</tr>
<tr>
<td>STR_TO_INT(E)</td>
<td>Converts string data into word [signed] data.</td>
<td>95</td>
</tr>
<tr>
<td>STR_TO_TIME(E)</td>
<td>Converts string data into time data.</td>
<td>107</td>
</tr>
<tr>
<td>XOR_E</td>
<td>Obtains the exclusive logical sum. (Number of pins variable)</td>
<td>185</td>
</tr>
</tbody>
</table>

---

**Function Name**

- **Appendix B-2** [In alphabetic order]

**Function List**

- **INT_TO_D(WORD(E))**: Converts word [signed] data into double word [unsigned/bit string [32-bit] data.
- **INT_TO_REAL(E)**: Converts word [signed] data into float [single precision] data.
- **INT_TO_STRING(E)**: Converts word [signed] data into string data.
- **INT_TO_TIME(E)**: Converts word [signed] data into time data.
- **INT_TO_WORD(E)**: Converts word [signed] data into word [unsigned/bit string [16-bit] data.
- **LE_E**: Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)
- **LIMITATION(E)**: Judges whether data is located within the range between the upper limit value and the lower limit value.
- **LT_E**: Compares data with regard to "< (smaller)". (Number of pins variable)
- **MAXIMUM(E)**: Searches the maximum value. (Number of pins variable)
- **MID(E)**: Obtains a character string from a specified position.
- **MINIMUM(E)**: Searches the minimum value. (Number of pins variable)
- **MOD(E)**: Divides data, and outputs the remainder. (Number of pins variable)
- **MOVE(E)**: Transfers data.
- **MUL_TIME(E)**: Multiplies time data.
- **MUL(E)**: Multiplies data. (Number of pins variable)
- **MUX(E)**: Selects data, and outputs it. (Number of pins variable)
- **NE_E**: Compares data with regard to “≠ (unequal)".
- **NOT(E)**: Obtains the logical not.
- **OR_E**: Obtains the logical sum. (Number of pins variable)
- **REAL_TO_DINT(E)**: Converts float [single precision] data into double word [signed] data.
- **REAL_TO_INT(E)**: Converts float [single precision] data into word [signed] data.
- **REAL_TO_STRING(E)**: Converts float [single precision] data into string data.
- **REPLACE(E)**: Replaces a character string.
- **R_TRIG(E)**: Detects the rising edge of a signal, and outputs pulse signal.
- **SEL(E)**: Selects data in accordance with the input condition.
- **SET_BIT_OF_INT(E)**: Writes a value to a specified bit of word [signed] data.
- **SHR(E)**: Shifts bits rightward.
- **STR_TO_BOOLEAN(E)**: Converts string data into bit data.
- **STR_TO_DINT(E)**: Converts string data into double word [signed] data.
- **STR_TO_REAL(E)**: Converts string data into word [signed] data.
- **STR_TO_DWORD(E)**: Converts string data into double word [unsigned/bit string [32-bit] data.
- **STR_TO_DWORD(E)**: Converts string data into double word [unsigned/bit string [32-bit] data.
- **STR_TO_INT(E)**: Converts string data into word [signed] data.
- **STR_TO_TIME(E)**: Converts string data into time data.
- **SUB_TIME(E)**: Subtracts time data.
- **SUB_E**: Subtracts data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_BYTE(E)**: Converts time data into byte data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_BYTE(E)**: Converts time data into byte data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_BYTE(E)**: Converts time data into byte data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_BYTE(E)**: Converts time data into byte data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_BYTE(E)**: Converts time data into byte data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_BYTE(E)**: Converts time data into byte data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_BYTE(E)**: Converts time data into byte data.
- **TIME_TO_BOOLEAN(E)**: Converts time data into bit data.
- **TIME_TO_DINT(E)**: Converts time data into double word [signed] data.
- **TIME_TO_DWORD(E)**: Converts time data into double word [unsigned/bit string [32-bit] data.
- **TIME_TO_INT(E)**: Converts time data into word [signed] data.
- **TIME_TO_STRING(E)**: Converts time data into string data.
<table>
<thead>
<tr>
<th>Operator name</th>
<th>Function</th>
<th>Ref. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Adds data. (Number of pins variable)</td>
<td>263</td>
</tr>
<tr>
<td>-</td>
<td>Subtracts data.</td>
<td>264</td>
</tr>
<tr>
<td>*</td>
<td>Multiplies data. (Number of pins variable)</td>
<td>265</td>
</tr>
<tr>
<td>/</td>
<td>Divides data (and outputs the quotient).</td>
<td>266</td>
</tr>
<tr>
<td>**</td>
<td>Obtains the raised result.</td>
<td>268</td>
</tr>
<tr>
<td>&amp;</td>
<td>Obtains the logical product. (Number of pins variable)</td>
<td>269</td>
</tr>
<tr>
<td>&gt;</td>
<td>Compares data with regard to “&gt;” (larger). (Number of pins variable)</td>
<td>274</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Compares data with regard to “≥” (larger or equal). (Number of pins variable)</td>
<td>275</td>
</tr>
<tr>
<td>=</td>
<td>Compares data with regard to “=” (equal). (Number of pins variable)</td>
<td>276</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Compares data with regard to “≤” (smaller or equal). (Number of pins variable)</td>
<td>277</td>
</tr>
<tr>
<td>&lt;</td>
<td>Compares data with regard to “&lt;” (smaller). (Number of pins variable)</td>
<td>278</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Compares data with regard to “≠” (unequal).</td>
<td>279</td>
</tr>
<tr>
<td>A</td>
<td>ADD Adds data. (Number of pins variable)</td>
<td>263</td>
</tr>
<tr>
<td></td>
<td>AND Obtains the logical product. (Number of pins variable)</td>
<td>269</td>
</tr>
<tr>
<td>D</td>
<td>DIV Divides data (and outputs the quotient).</td>
<td>266</td>
</tr>
<tr>
<td>E</td>
<td>EQ Compares data with regard to “≈” (equal). (Number of pins variable)</td>
<td>276</td>
</tr>
<tr>
<td>G</td>
<td>GE Compares data with regard to “≥” (larger or equal). (Number of pins variable)</td>
<td>275</td>
</tr>
<tr>
<td></td>
<td>GT Compares data with regard to “&gt;” (larger). (Number of pins variable)</td>
<td>274</td>
</tr>
<tr>
<td>L</td>
<td>LE Compares data with regard to “≤” (smaller or equal). (Number of pins variable)</td>
<td>277</td>
</tr>
<tr>
<td></td>
<td>LT Compares data with regard to “&lt;” (smaller). (Number of pins variable)</td>
<td>278</td>
</tr>
<tr>
<td>M</td>
<td>MOD Divides data (and outputs the remainder).</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>MUL Multiplies data. (Number of pins variable)</td>
<td>265</td>
</tr>
<tr>
<td>N</td>
<td>NE Compares data with regard to “≠” (unequal).</td>
<td>279</td>
</tr>
<tr>
<td></td>
<td>NOT Obtains the logical not.</td>
<td>273</td>
</tr>
<tr>
<td>O</td>
<td>OR Obtains the logical sum. (Number of pins variable)</td>
<td>270</td>
</tr>
<tr>
<td>S</td>
<td>SUB Subtracts data.</td>
<td>264</td>
</tr>
<tr>
<td>X</td>
<td>XOR Obtains the exclusive logical sum. (Number of pins variable)</td>
<td>271</td>
</tr>
</tbody>
</table>
Warranty

Please confirm the following product warranty details before using this product.

1. Gratis Warranty Term and Gratis Warranty Range
   If any faults or defects (hereinafter "Failure") found to be the responsibility of Mitsubishi occurs during use of the product within the gratis warranty term, the product shall be repaired at no cost via the sales representative or Mitsubishi Service Company. However, if repairs are required onsite at domestic or overseas location, expenses to send an engineer will be solely at the customer's discretion. Mitsubishi shall not be held responsible for any re-commissioning, maintenance, or testing on-site that involves replacement of the failed module.

   [Gratis Warranty Term]
   The gratis warranty term of the product shall be for one year after the date of purchase or delivery to a designated place. Note that after manufacture and shipment from Mitsubishi, the maximum distribution period shall be six (6) months, and the longest gratis warranty term after manufacturing shall be eighteen (18) months. The gratis warranty term of repair parts shall not exceed the gratis warranty term before repairs.

   [Gratis Warranty Range]
   1) The range shall be limited to normal use within the usage state, usage methods and usage environment, etc., which follow the conditions and precautions, etc., given in the instruction manual, user's manual and caution labels on the product.
   2) Even within the gratis warranty term, repairs shall be charged for in the following cases.
      a) Failure occurring from inappropriate storage or handling, carelessness or negligence by the user. Failure caused by the user's hardware or software design.
      b) Failure caused by unapproved modifications, etc., to the product by the user.
      c) When the Mitsubishi product is assembled into a user's device, Failure that could have been avoided if functions or structures, judged as necessary in the legal safety measures the user's device is subject to or as necessary by industry standards, had been provided.
      d) Failure that could have been avoided if consumable parts (battery, backlight, fuse, etc.) designated in the instruction manual had been correctly serviced or replaced.
      e) Relay failure or output contact failure caused by usage beyond the specified Life of contact (cycles).
      f) Failure caused by external irresistible forces such as fires or abnormal voltages, and failure caused by force majeure such as earthquakes, lightning, wind and water damage.
      g) Failure caused by reasons unpredictable by scientific technology standards at time of shipment from Mitsubishi.
      h) Any other failure found not to be the responsibility of Mitsubishi or that admitted not to be so by the user.

2. Onerous repair term after discontinuation of production
   1) Mitsubishi shall accept onerous product repairs for seven (7) years after production of the product is discontinued. Discontinuation of production shall be notified with Mitsubishi Technical Bulletins, etc.
   2) Product supply (including repair parts) is not available after production is discontinued.

3. Overseas service
   Overseas, repairs shall be accepted by Mitsubishi's local overseas FA Center. Note that the repair conditions at each FA Center may differ.

4. Exclusion of loss in opportunity and secondary loss from warranty liability
   Regardless of the gratis warranty term, Mitsubishi shall not be liable for compensation of damages caused by any cause found not to be the responsibility of Mitsubishi, loss in opportunity, lost profits incurred to the user or third person by Failures of Mitsubishi products, special damages and secondary damages whether foreseeable or not, compensation for accidents, and compensation for damages to products other than Mitsubishi products, replacement by the user, maintenance of on-site equipment, start-up test run and other tasks.

5. Changes in product specifications
   The specifications given in the catalogs, manuals or technical documents are subject to change without prior notice.

6. Product application
   1) In using the Mitsubishi MELSEC programmable logic controller, the usage conditions shall be that the application will not lead to a major accident even if any problem or fault should occur in the programmable logic controller device, and that backup and fail-safe functions are systematically provided outside of the device for any problem or fault.
   2) The Mitsubishi programmable logic controller has been designed and manufactured for applications in general industries, etc. Thus, applications in which the public could be affected such as in nuclear power plants and other power plants operated by respective power companies, and applications in which a special quality assurance system is required, such as for Railway companies or Public service purposes shall be excluded from the programmable logic controller applications.

   In addition, in certain cases, some applications may be possible, providing the user consults their local Mitsubishi representative outlining the special requirements of the project, and providing that all parties concerned agree to the special circumstances, solely at the users discretion.
## Revision History

<table>
<thead>
<tr>
<th>Date of preparation</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
</table>
| 7/2009              | B        | • Equivalent circuits are deleted.  
                      • Following instructions are not supported in FX0, FX0S and FX0N PLCs.  
                      CTD(_E), CTU(_E), CTUD(_E), TOF(_E), TON(_E), TP(_E)  
                      • Function blocks (SR(_E), RS(_E)) are deleted. |
| 2/2010              | C        | • Manual name of a related manual was changed.  
                      • Operators are added. |
| 9/2010              | D        | • Following functions are added.  
                      BITARR_TO_INT(_E), BITARR_TO_DINT(_E), INT_TO_BITARR(_E),  
                      DINT_TO_BITARR(_E), CPY_BITARR(_E), GET_BIT_OF_INT(_E),  
                      SET_BIT_OF_INT(_E), CPY_BIT_OF_INT(_E), GET_BOOL_ADDR,  
                      GET_INT_ADDR and GET_WORD_ADDR  
                      • Appendix B is added. |
| 7/2011              | E        | • FBD language is added. |
| 2/2012              | F        | • FX3GC is added. |
| 5/2012              | G        | The reset method of the counter value and timer value of a function block  
                      (COUNTER_FB_M, TIMER_CONT_FB_M) is added. |
| 2/2013              | H        | • Function blocks are added.  
                      TON_10, TON_10_E, TOF_10, TOF_10_E, TP_10, TP_10_E |
| 5/2013              | J        | • FX3S is added. |
| 4/2015              | K        | • A part of the cover design is changed. |
## FXCPU

**Structured Programming Manual**

### Application Functions

<table>
<thead>
<tr>
<th>MODEL</th>
<th>FX-KP-OK-E</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL CODE</td>
<td>09R927</td>
</tr>
</tbody>
</table>

MITSUBISHI ELECTRIC CORPORATION
HEAD OFFICE: TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN

JY997D34801K (MEE)  
Effective April 2015  
Specifications are subject to change without notice.