MITSUBISHI
ELECTRIC

PROGRAMMABLE CONTROLLERS
MELSEC-F

FXCPU

Structured Programming Manual

Application Functions

FX

FXCPU Structured Programming Manual

[Application Functions]

Manual number	JY997D34801
Manual revision	K
Date	4/2015

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.

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 15

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 23

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 29

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_STR(_E) / Float (single precision) data → string 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

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

14.9 COUNTER_FB_M / Counter function blocks 256
14.10 TIMER_10_FB_M / Timer function blocks 258
14.11 TIMER_CONT_FB_M / Timer function blocks 259
14.12 TIMER_100_FB_M / Timer function blocks 261

15. Operator 262

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

Appendix A: Correspondence between Devices and Addresses 280

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

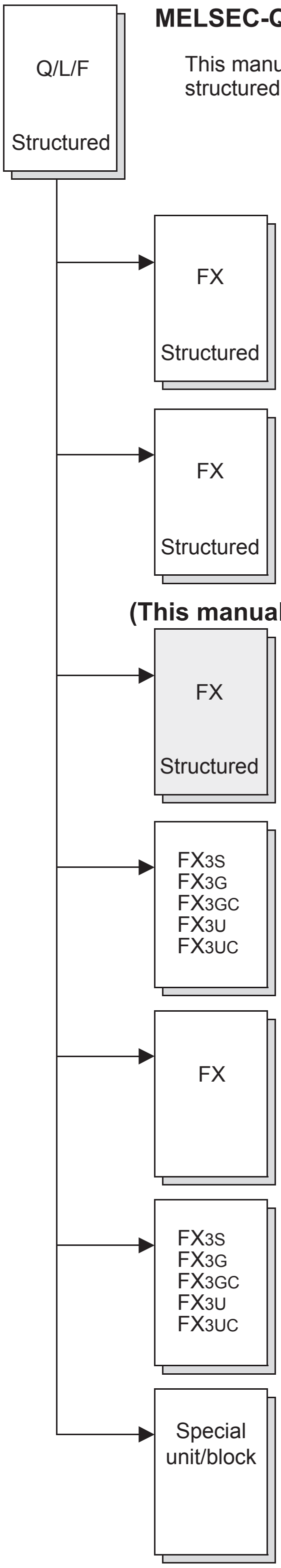
Appendix B-1 [By type] 282
Appendix B-2 [In alphabetic order] 285

Warranty 289
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



MELSEC-Q/L/F Structured Programming Manual (Fundamentals) (Additional Manual)

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

FXCPU Structured Programming Manual [Device & Common] (Additional Manual)

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

FXCPU Structured Programming Manual [Basic & Applied Instruction]

(Additional Manual)

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

FXCPU Structured Programming Manual [Application Functions]

(Additional Manual)

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

FX3S/FX3G/FX3GC/FX3U/FX3UC User's Manual- Analog Control Edition

(Additional Manual)

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.

FX Series User's Manual -Data Communication Edition (Additional Manual)

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.

FX3S/FX3G/FX3GC/FX3U/FX3UC Series User's Manual -Positioning Edition

(Additional Manual)

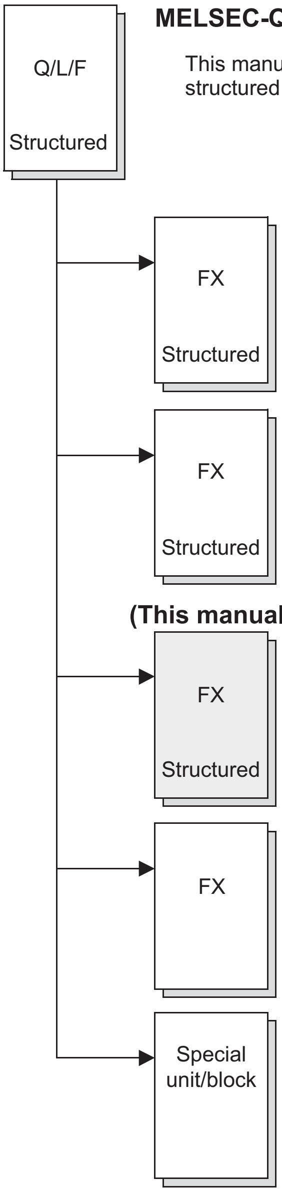
This manual explains details of wiring, instructions and operations of the positioning function built in FX3S/FX3G/FX3GC/FX3U/FX3UC PLC main units.
Explanation of instructions and instructions used in program examples are expressed for GX Developer.

Individual manuals (Manual supplied with product or additional 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.

*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.

FXCPU Structured Programming Manual [Device & Common] (Additional Manual)

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

FXCPU Structured Programming Manual [Basic & Applied Instruction] (Additional Manual)

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

FXCPU Structured Programming Manual [Application Functions] (Additional Manual)

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

FX Series User's Manual -Data Communication Edition (Additional Manual)

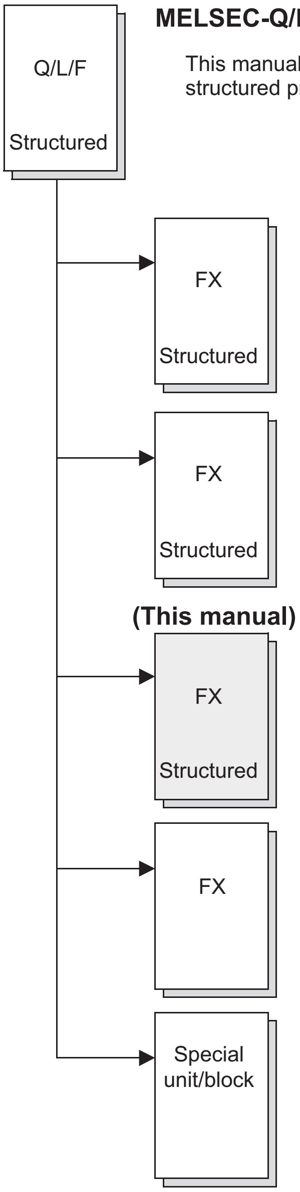
This manual explains details of simple N:N link, parallel link, computer link, no-protocol communication (RS instruction), programming communication and inverter communication for FX PLCs.
Explanation of instructions and instructions used in program examples are expressed for GX Developer and FX-PCS/WIN.

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.

FXCPU Structured Programming Manual [Device & Common] (Additional Manual)

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

FXCPU Structured Programming Manual [Basic & Applied Instruction] (Additional Manual)

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

FXCPU Structured Programming Manual [Application Functions] (Additional Manual)

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

FX Series User's Manual -Data Communication Edition (Additional Manual)

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 [*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.

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]

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
MELSEC-Q/L/F Structured Programming Manual (Fundamentals)	SH-080782	Additional Manual	Programming methods, specifications, functions, etc. required to create structured programs	13JW06
FXCPU Structured Programming Manual [Device & Common]	JY997D26001	Additional Manual	Devices, parameters, etc. provided in structured projects of GX Works2	09R925
FXCPU Structured Programming Manual [Basic & Applied Instruction]	JY997D34701	Additional Manual	Sequence instructions provided in structured projects of GX Works2	09R926
FXCPU Structured Programming Manual [Application Functions]	JY997D34801	Additional Manual	Application functions provided in structured projects of GX Works2	09R927

FX3S/FX3G/FX3GC/FX3U/FX3UC PLCs

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
PLC main unit				
FX3U Series Hardware Manual	JY997D18801	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3U extracted from the FX3U Series User’s Manual - Hardware Edition. For detailed explanation, refer to the FX3U Series User’s Manual - Hardware Edition.	-
FX3U Series User's Manual- Hardware Edition	JY997D16501	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3U PLC main unit.	09R516
FX3UC (D, DS, DSS) Series Hardware Manual	JY997D28601	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3UC (D, DS, DSS) extracted from the FX3UC Series User’s Manual - Hardware Edition. For detailed explanation, refer to the FX3UC Series User’s Manual - Hardware Edition.	-
FX3UC-32MT-LT-2 Hardware Manual	JY997D31601	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3UC-32MT-LT-2 extracted from the FX3UC Series User’s Manual - Hardware Edition. For detailed explanation, refer to the FX3UC Series User’s Manual - Hardware Edition.	-
FX3UC Series User's Manual - Hardware Edition	JY997D28701	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3UC PLC main unit.	09R519
FX3G Series Hardware Manual	JY997D33401	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3G extracted from the FX3G Series User’s Manual - Hardware Edition. For detailed explanation, refer to the FX3G Series User’s Manual - Hardware Edition.	-
FX3G Series User's Manual- Hardware Edition	JY997D31301	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3G PLC main unit.	09R521
FX3GC Series Hardware Manual	JY997D45201	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3GC extracted from the FX3GC Series User's Manual - Hardware Edition. For detailed explanation, refer to the FX3GC Series User's Manual - Hardware Edition.	-
FX3GC Series User's Manual- Hardware Edition	JY997D45401	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3GC PLC main unit.	09R533

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
PLC main unit				
FX3S Series Hardware Manual	JY997D48301	Supplied with product	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.	-
FX3S Series User's Manual- Hardware Edition	JY997D48601	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3S PLC main unit.	09R535
Programming				
FX3S/FX3G/FX3GC/FX3U/FX3UC User's Manual- Analog Control Edition	JY997D16701	Additional Manual	Detaileds about the analog special function block (FX3U-4AD, FX3U-4DA, FX3UC-4AD) and analog special adapter (FX3U-****-ADP).	09R619
FX Series User's Manual -Data Communication Edition	JY997D16901	Additional Manual	Details about simple N : N link, parallel link, computer link and no-protocol communication (RS instruction and FX2N-232IF).	09R715
FX3S/FX3G/FX3GC/FX3U/FX3UC Series User's Manual- MODBUS Serial Communication Edition	JY997D26201	Additional Manual	Explains the MODBUS serial communication network in FX3S/FX3G/FX3GC/FX3U/FX3UC PLCs.	09R626
FX3S/FX3G/FX3GC/FX3U/FX3UC Series User's Manual -Positioning Edition	JY997D16801	Additional Manual	Details about the positioning function built in the FX3S/FX3G/FX3GC/FX3U/FX3UC Series.	09R620
FX3U-CF-ADP User's Manual	JY997D35401	Additional Manual	Describes details of the FX3U-CF-ADP CF card special adapter.	09R720

FX1S/FX1N/FX1NC PLCs
FX2N/FX2NC PLCs [whose production is finished]

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
PLC main unit				
FX1S HARDWARE MANUAL	JY992D83901	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1S PLC main unit.	-
FX1N HARDWARE MANUAL	JY992D89301	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1N PLC main unit.	-
FX2N HARDWARE MANUAL	JY992D66301	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX2N PLC main unit.	09R508
FX1NC HARDWARE MANUAL	JY992D92101	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1NC PLC main unit. (Japanese only)	09R505
FX2NC HARDWARE MANUAL	JY992D76401	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX2NC PLC main unit.	09R509
Programming				
FX Series User's Manual -Data Communication Edition	JY997D16901	Additional Manual	Details about simple N : N link, parallel link, computer link and no-protocol communication (RS instruction and FX2N-232IF).	09R715

FX0S/FX0/FX0N/FXU/FX2C PLCs [whose production is finished]

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
PLC main unit				
FX0/FX0N HARDWARE MANUAL	JY992D47501	Supplied with product	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX0/FX0N PLC main unit.	-
FX0S HARDWARE MANUAL	JY992D55301	Supplied with product	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX0S PLC main unit.	-
FX/FX2C HARDWARE MANUAL	JY992D47401	Supplied with product	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FXU/FX2C PLC main unit.	-
Programming				
FX Series User's Manual -Data Communication Edition	JY997D16901	Additional Manual	Details about simple N : N link, parallel link, computer link and no-protocol communication (RS instruction and FX2N-232IF).	09R715

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

Abbreviation/generic name	Name
PLCs	
FX3U Series or FX3U PLC	Generic name of FX3U Series PLCs
FX3UC Series or FX3UC PLC	Generic name of FX3UC Series PLCs
FX3G Series or FX3G PLC	Generic name of FX3G Series PLCs
FX3GC Series or FX3GC PLC	Generic name of FX3GC Series PLCs
FX3S Series or FX3S PLC	Generic name of FX3S Series PLCs
FX2N Series or FX2N PLC	Generic name of FX2N Series PLCs
FX2NC Series or FX2NC PLC	Generic name of FX2NC Series PLCs
FX1N Series or FX1N PLC	Generic name of FX1N Series PLCs
FX1NC Series or FX1NC PLC	Generic name of FX1NC Series PLCs These products can only used in Japan.
FX1S Series or FX1S PLC	Generic name of FX1S Series PLCs
FXU Series or FXU PLC	Generic name of FXU(FX, FX2) Series PLCs
FX2C Series or FX2C PLC	Generic name of FX2C Series PLCs
FX0N Series or FX0N PLC	Generic name of FX0N Series PLCs
FX0S Series or FX0S PLC	Generic name of FX0S Series PLCs
FX0 Series or FX0 PLC	Generic name of FX0 Series PLCs
Special adapters	
CF card special adapter	Generic name of CF card special adapters
CF-ADP	FX3U-CF-ADP
Ethernet adapter	Abbreviated of FX3U-ENET-ADP
Programming language	
ST	Abbreviation of structured text language
Structured ladder	Abbreviation of ladder diagram language
FBD	Abbreviation of function block diagram language
Manuals	
Q/L/F Structured Programming Manual (Fundamentals)	Abbreviation of MELSEC-Q/L/F Structured Programming Manual [Fundamentals]
FX Structured Programming Manual [Device & Common]	Abbreviation of FXCPU Structured Programming Manual [Device & Common]
FX Structured Programming Manual [Basic & Applied Instruction]	Abbreviation of FXCPU Structured Programming Manual [Basic & Applied Instruction]
FX Structured Programming Manual [Application Functions]	Abbreviation of FXCPU Structured Programming Manual [Application Functions]
COMMUNICATION CONTROL EDITION	Abbreviation of FX Series User's Manual-DATA COMMUNICATION CONTROL EDITION
ANALOG CONTROL EDITION	Abbreviation of FX3S/FX3G/FX3GC/FX3U/FX3UC Series User's Manual-ANALOG CONTROL EDITION
POSITIONING CONTROL EDITION	Abbreviation of FX3S/FX3G/FX3GC/FX3U/FX3UC Series User's Manual-POSITIONING CONTROL EDITION

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.

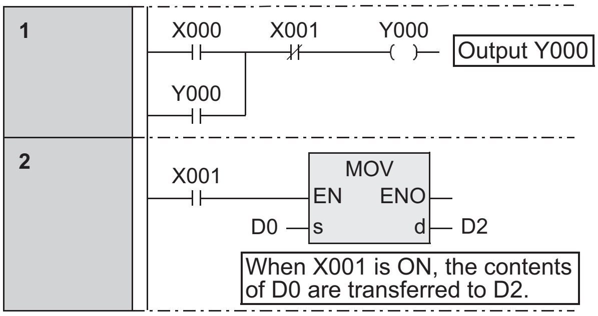
1.1.2 Programming languages

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

Graphic languages

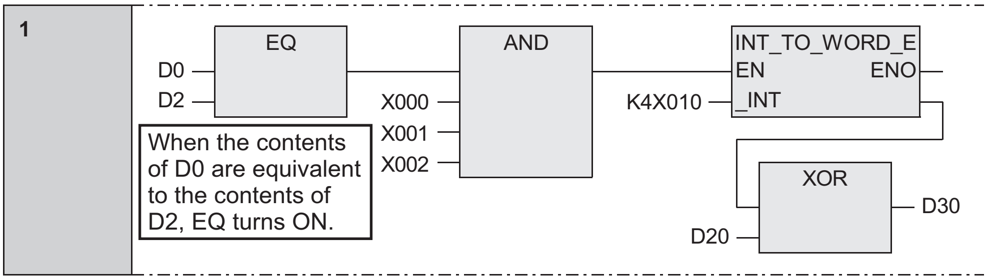
1. Structured ladder language

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

PLC Series	Software package name (model name)	GX Works2 version
FX3U•FX3UC	GX Works2 (SW1DNC-GXW2-E)	Ver. 1.08J or later
FX3G		
FX2N•FX2NC		
FX1N•FX1NC		
FX1S		
FXU/FX2C		
FX0N		
FX0•FX0S		
FX3GC		Ver.1.77F or later
FX3S		Ver.1.492N or later

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)**

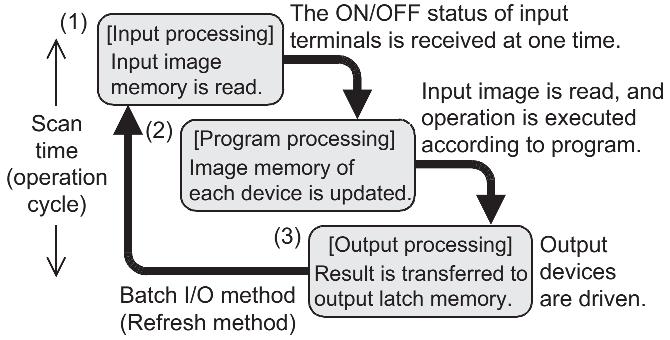
Refer to the following programming manual for detailed operations of and cautions on devices and parameters:

→ **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.

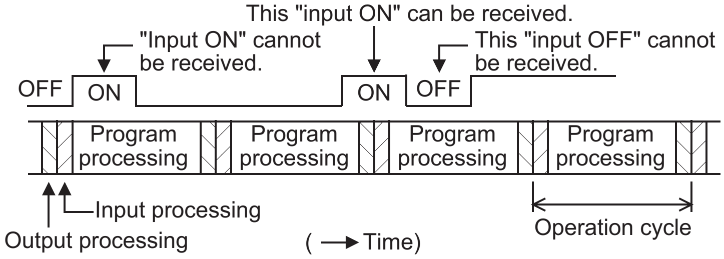
2. **Short pulses cannot be received.**

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

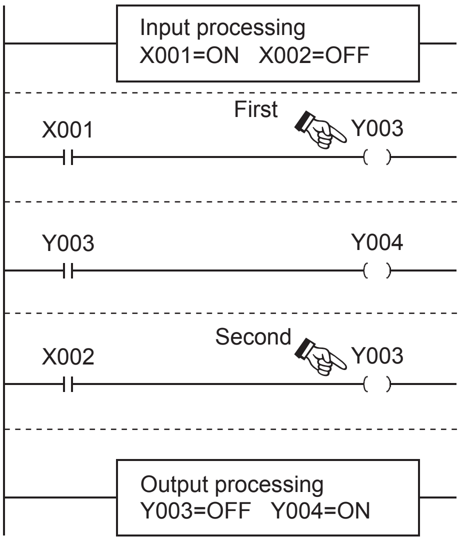
When a coil (output variable) is used twice (double coils) in another program block to be executed or in the same program block, the PLC gives priority to the latter coil.

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.

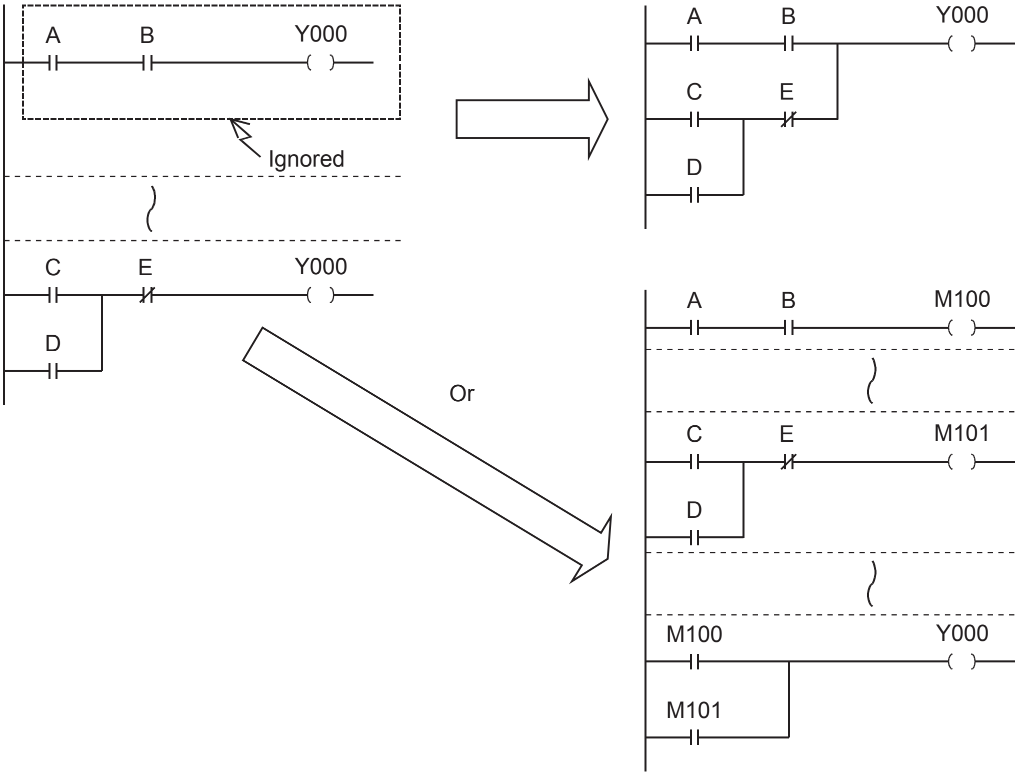
In the second coil Y003, however, the image memory is set to OFF because the input X002 is OFF.

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.

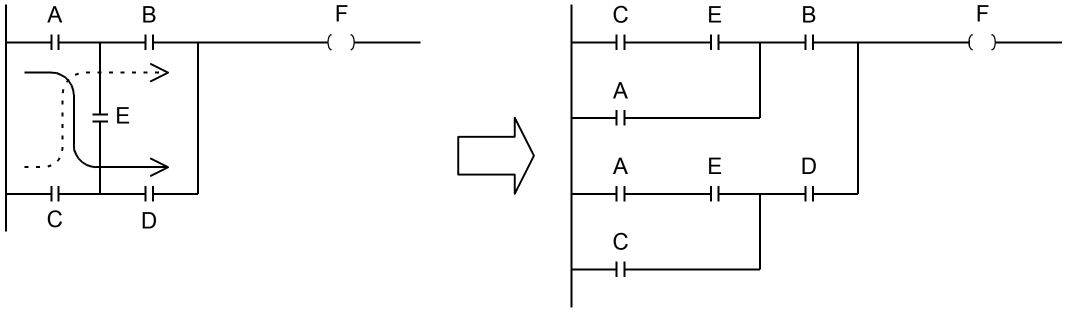


The SET and RST instructions or jump instruction can be used instead, or a same output coil can be programmed at each state using step ladder instructions STL and RET.
When you use the step ladder instructions STL and RET, note that the PLC regards it as double coils if you program, inside the state, 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

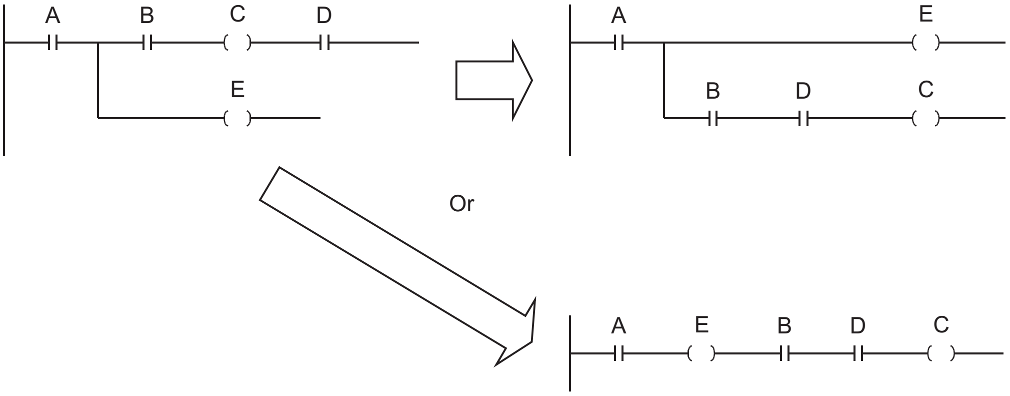
1. Bridge circuit

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
M8328:Instruction non-execution flag*1
M8329:Instruction execution abnormal complete flag*2
M8304:Zero flag*2
M8306:Carry flag*2

*1. Supported only in FX3U/FX3UC PLCs.

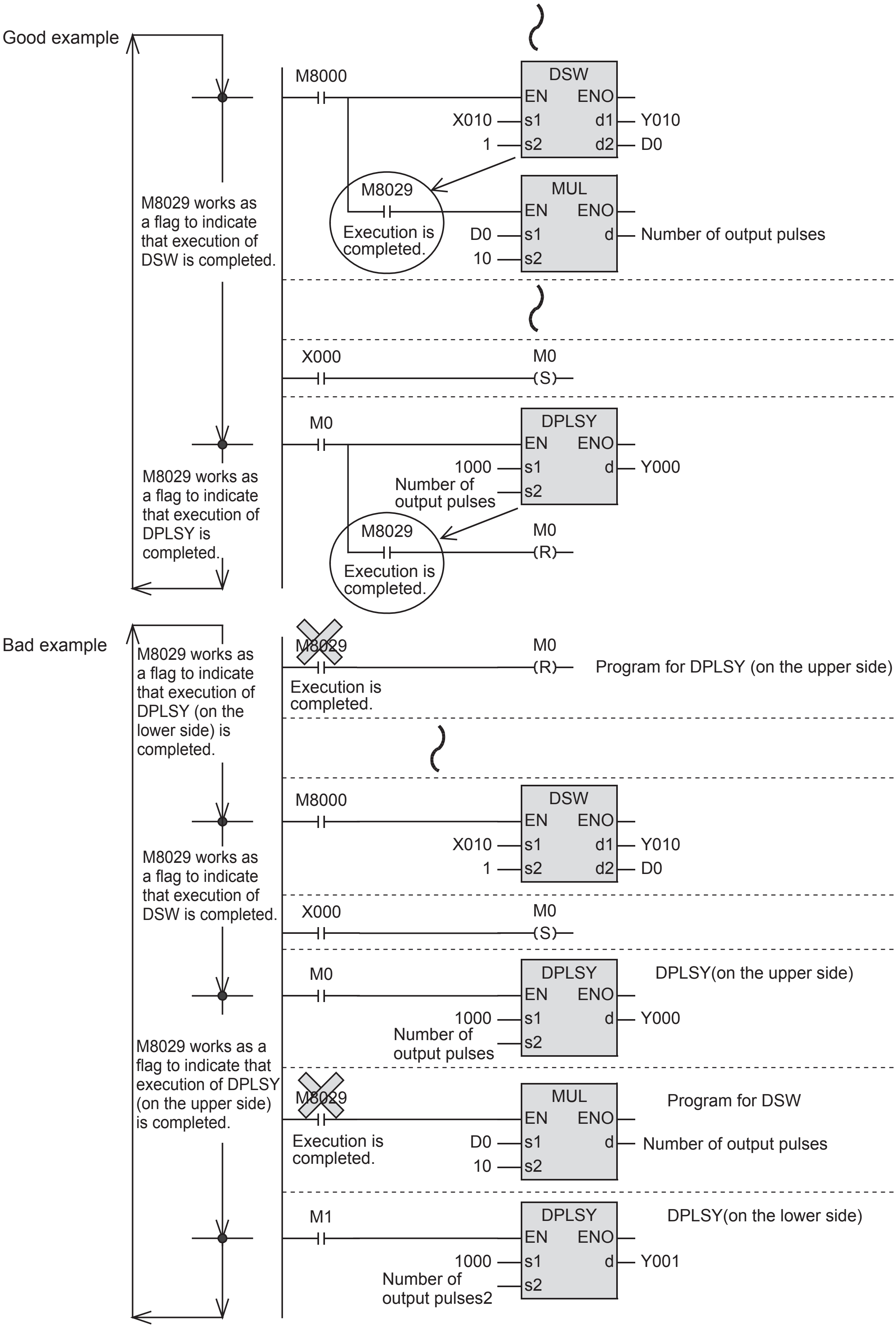
*2. Supported only in FX3S/FX3G/FX3GC/FX3U/FX3UC PLCs.

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.

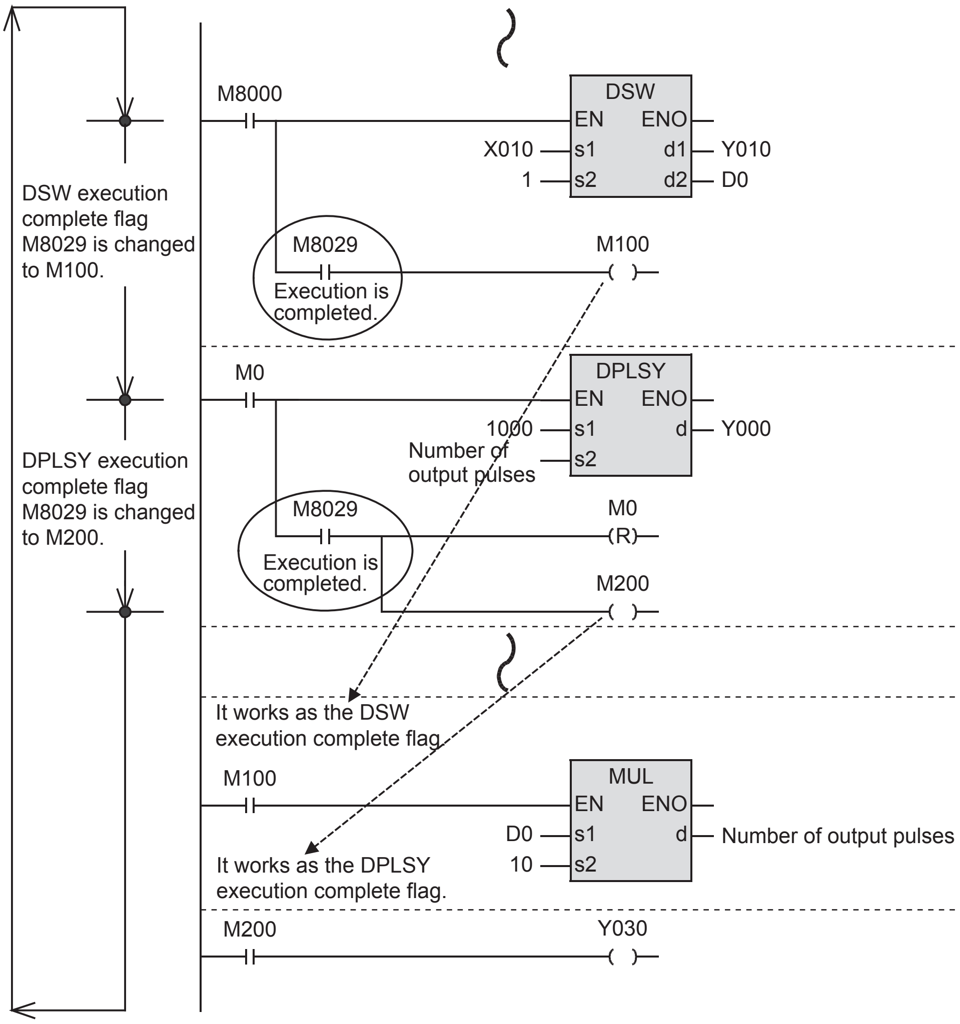
Refer to the next page when you would like to use the flag M8029 in any position other than the position just under the corresponding sequence instruction.



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

Error flag	Device which stores error code	Device which stores error occurrence step	
		FX0S/FX0/FX0N/FX1S/FX1N/FX1NC/ FXU/FX2C/FX2N/FX2NC/FX3S/FX3G/FX3GC	FX3U/FX3UC
M8067	D8067	D8069[*1]	D8315, D8314

*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

Error flag	Device which stores error code	Device which stores error occurrence step	
		FX0S/FX0/FX0N/FX1S/FX1N/FX1NC/ FXU/FX2C/FX2N/FX2NC/FX3S/FX3G/FX3GC	FX3U/FX3UC
M8068	-	D8068[*2]	D8313, D8312

*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

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
BOOL_TO_INT(_E)	Converts bit data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.1
BOOL_TO_DINT(_E)	Converts bit data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.2
BOOL_TO_STR(_E)	Converts bit data into string data.	✓									Section 5.3
BOOL_TO_WORD(_E)	Converts bit data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.4
BOOL_TO_DWORD (_E)	Converts bit data into double word [unsigned]/bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.5
BOOL_TO_TIME(_E)	Converts bit data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.6
INT_TO_DINT(_E)	Converts word [signed] data into double word [signed] data	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.7
DINT_TO_INT(_E)	Converts double word [signed] data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.8
INT_TO_BOOL(_E)	Converts word [signed] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.9
DINT_TO_BOOL(_E)	Converts double word [signed] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.10
INT_TO_REAL(_E)	Converts word [signed] data into float (single precision) data.	✓	*1	✓	✓						Section 5.11
DINT_TO_REAL(_E)	Converts double word [signed] data into float (single precision) data.	✓	*1	✓	✓						Section 5.12
INT_TO_STR(_E)	Converts word [signed] data into string data.	✓									Section 5.13
DINT_TO_STR(_E)	Converts double word [signed] data into string data.	✓									Section 5.14
INT_TO_WORD(_E)	Converts word [signed] data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.15
DINT_TO_WORD(_E)	Converts double word [signed] data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.16
INT_TO_DWORD(_E)	Converts word [signed] data into double word [unsigned]/bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.17
DINT_TO_DWORD (_E)	Converts double word [signed] data into double word [unsigned]/bit string[32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.18
INT_TO_BCD(_E)	Converts word [signed] data into BCD data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.19
DINT_TO_BCD(_E)	Converts double word [signed] data into BCD data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.20
INT_TO_TIME(_E)	Converts word [signed] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.21
DINT_TO_TIME(_E)	Converts double word [signed] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.22
REAL_TO_INT(_E)	Converts float (single precision) data into word [signed] data.	✓	*1	✓	✓						Section 5.23
REAL_TO_DINT(_E)	Converts float (single precision) data into double word [signed] data.	✓	*1	✓	✓						Section 5.24
REAL_TO_STR(_E)	Converts float (single precision) data into string data.	✓									Section 5.25
WORD_TO_BOOL(_E)	Converts word [unsigned]/bit string [16-bit] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.26

*1. The function is provided in the FX3G Series Ver.1.10 or later.

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
DWORD_TO_BOOL (_E)	Converts double word [unsigned]/bit string [32-bit] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.27
WORD_TO_INT(_E)	Converts word [unsigned]/bit string [16-bit] data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.28
WORD_TO_DINT(_E)	Converts word [unsigned]/bit string [16-bit] data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.29
DWORD_TO_INT(_E)	Converts double word [unsigned]/bit string [32-bit] data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.30
DWORD_TO_DINT (_E)	Converts double word [unsigned]/bit string [32-bit] data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.31
WORD_TO_DWORD (_E)	Converts word [unsigned]/bit string [16-bit] data into double word [unsigned]/bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.32
DWORD_TO_WORD (_E)	Converts double word [unsigned]/bit string [32-bit] data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.33
WORD_TO_TIME(_E)	Converts word [unsigned]/bit string [16-bit] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.34
DWORD_TO_TIME (_E)	Converts double word [unsigned]/bit string [32-bit] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.35
STR_TO_BOOL(_E)	Converts string data into bit data.	✓									Section 5.36
STR_TO_INT(_E)	Converts string data into word [signed] data.	✓									Section 5.37
STR_TO_DINT(_E)	Converts string data into double word [signed] data.	✓									Section 5.38
STR_TO_REAL(_E)	Converts string data into float (single precision) data.	✓									Section 5.39
STR_TO_TIME(_E)	Converts string data into time data.	✓									Section 5.40
BCD_TO_INT(_E)	Converts BCD data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.41
BCD_TO_DINT(_E)	Converts BCD data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.42
TIME_TO_BOOL(_E)	Converts time data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.43
TIME_TO_INT(_E)	Converts time data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.44
TIME_TO_DINT(_E)	Converts time data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.45
TIME_TO_STR(_E)	Converts time data into string data.	✓									Section 5.46
TIME_TO_WORD(_E)	Converts time data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.47
TIME_TO_DWORD (_E)	Converts time data into double word [unsigned]/ bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.48
BITARR_TO_INT(_E)	Converts specified number of bits of a bit array into word [signed] data or word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.49
BITARR_TO_DINT(_E)	Converts specified number of bits of a bit array into double word [signed] data or double word [unsigned]/bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.50
INT_TO_BITARR(_E)	Outputs low-order "n" bits of word [signed] data or word [unsigned]/bit string [16-bit] data to a bit array.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.51
DINT_TO_BITARR(_E)	Outputs low-order "n" bits of double word [signed] data or double word [unsigned]/bit string [32-bit] data to a bit array.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.52
CPY_BITARR(_E)	Copies specified number of bits of a bit array.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.53
GET_BIT_OF_INT(_E)	Reads a value of a specified bit of word [signed] data.	✓									Section 5.54
SET_BIT_OF_INT(_E)	Writes a value to a specified bit of word [signed] data.	✓									Section 5.55
CPY_BIT_OF_INT(_E)	Copies a specified bit of word [signed] data to a specified bit of another word [signed] data.	✓									Section 5.56

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
GET_BOOL_ADDR	Outputs start data as bit data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.57
GET_INT_ADDR	Outputs start data as word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.58
GET_WORD_ADDR	Outputs start data as word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 5.59

2.2 Standard Functions Of One Numeric Variable

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
ABS(_E)	Obtains the absolute value.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 6.1

2.3 Standard Arithmetic Functions

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
ADD_E	Adds data. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 7.1
SUB_E	Subtracts data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 7.2
MUL_E	Multiplies data. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 7.3
DIV_E	Divides data (and outputs the quotient).	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 7.4
MOD(_E)	Divides data (and outputs the remainder).	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 7.5
EXPT(_E)	Obtains the raised result.	✓									Section 7.6
MOVE(_E)	Transfers data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 7.7

2.4 Standard Bit Shift Functions

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
SHL(_E)	Shifts bits leftward.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 8.1
SHR(_E)	Shifts bits rightward.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 8.2

2.5 Standard Bitwise Boolean Functions

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
AND_E	Obtains the logical product. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 9.1
OR_E	Obtains the logical sum. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 9.2
XOR_E	Obtains the exclusive logical sum. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 9.3
NOT_E	Obtains the logical not.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 9.3

2.6 Standard Selection Functions

Function name	Function	Applicable PLC									Reference
		FX3U(C)	**FX3G(C)**	**FX3S**	**FX2N(C)**	**FX1N(C)**	**FX1S**	**FXU/FX2C**	**FX0N**	**FX0(S)**	
SEL(_E)	Selects data in accordance with the input condition.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 10.1
MAXIMUM(_E)	Searches the maximum value. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 10.2
MINIMUM(_E)	Searches the minimum value. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 10.3
LIMITATION(_E)	Judges whether data is located within the range between the upper limit value and the lower limit value.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 10.4
MUX(_E)	Selects data, and outputs it. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 10.5

2.7 Standard Comparison Functions

Function name	Function	Applicable PLC									Reference
		FX3U(C)	**FX3G(C)**	**FX3S**	**FX2N(C)**	**FX1N(C)**	**FX1S**	**FXU/FX2C**	**FX0N**	**FX0(S)**	
GT_E	Compares data with regard to "> (larger)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 11.1
GE_E	Compares data with regard to "≥ (larger or equal)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 11.2
EQ_E	Compares data with regard to "= (equal)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 11.3
LE_E	Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 11.4
LT_E	Compares data with regard to "< (smaller)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 11.5
NE_E	Compares data with regard to "≠ (unequal)".	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 11.6

2.8 Standard Character String Functions

Function name	Function	Applicable PLC									Reference
		FX3U(C)	**FX3G(C)**	**FX3S**	**FX2N(C)**	**FX1N(C)**	**FX1S**	**FXU/FX2C**	**FX0N**	**FX0(S)**	
MID(_E)	Obtains a character string from a specified position.	✓									Section 12.1
CONCAT(_E)	Connects character strings. (Number of pins variable)	✓									Section 12.2
INSERT(_E)	Inserts a character string.	✓									Section 12.3
DELETE(_E)	Deletes a character string.	✓									Section 12.4
REPLACE(_E)	Replaces a character string.	✓									Section 12.5
FIND(_E)	Searches a character string.	✓									Section 12.6

2.9 Functions Of Time Data Types

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
ADD_TIME(_E)	Adds time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 13.1
SUB_TIME(_E)	Subtracts time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 13.2
MUL_TIME(_E)	Multiplies time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 13.3
DIV_TIME(_E)	Divides time data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 13.4

2.10 Standard Function Blocks

Function name	Function	Applicable PLC									Reference
		FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
R_TRIG(_E)	Detects the rising edge of a signal, and outputs pulse signal.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 14.1
F_TRIG(_E)	Detects the falling edge of a signal, and outputs pulse signal.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 14.2
CTU(_E)	Counts up the number of times of rising of a signal.	✓	✓	✓	✓	✓	✓	✓			Section 14.3
CTD(_E)	Counts down the number of times of rising of a signal.	✓	✓	✓	✓	✓	✓	✓			Section 14.4
CTUD(_E)	Counts up/down the number of times of rising of a signal.	✓	✓	✓	✓	✓	✓	✓			Section 14.5
TP(_E)	Keeps ON a signal during specified time duration.	✓	✓	✓	✓	✓	✓	✓			Section 14.6
TP_10(_E)		✓	✓		✓	✓		✓			
TON(_E)	Keeps OFF a signal during specified time duration.	✓	✓	✓	✓	✓	✓	✓			Section 14.7
TON_10(_E)		✓	✓		✓	✓		✓			
TOF(_E)	Turns OFF the output signal at specified time after the input signal turned OFF.	✓	✓	✓	✓	✓	✓	✓			Section 14.8
TOF_10(_E)		✓	✓		✓	✓		✓			
COUNTER_FB_M	Counter drive	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 14.9
TIMER_10_FB_M	10ms timer drive	✓	✓		✓	✓		✓			Section 14.10
TIMER_CONT_FB_M	Retentive timer drive	✓	✓	✓	✓	✓		✓			Section 14.11
TIMER_100_FB_M	100ms timer drive	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 14.12

2.11 Operator

2.11.1 Arithmetic operations

Operator name		Function	Applicable PLC									Reference
Structured ladder /FBD	**ST**		**FX3U(C)**	**FX3G(C)**	**FX3S**	**FX2N(C)**	**FX1N(C)**	**FX1S**	**FXU/FX2C**	**FX0N**	**FX0(S)**	
ADD	+	Adds data. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.1
SUB	-	Subtracts data.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.2
MUL	*	Multiplies data. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.3
DIV	/	Divides data (, and outputs the quotient).	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.4
-	MOD	Divides data (, and outputs the remainder).	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.5
-	**	Obtains the raised result.	✓									Section 15.6

2.11.2 Logical operations

Operator name		Function	Applicable PLC									Reference
Structured ladder /FBD	**ST**		**FX3U(C)**	**FX3G(C)**	**FX3S**	**FX2N(C)**	**FX1N(C)**	**FX1S**	**FXU/FX2C**	**FX0N**	**FX0(S)**	
AND	& AND	Obtains the logical product. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.7
OR	OR	Obtains the logical sum. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.8
XOR	XOR	Obtains the exclusive logical sum. (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.9
-	NOT	Obtains the logical not.	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.10

2.11.3 Comparison operations

Operator name		Function	Applicable PLC									Reference
Structured ladder /FBD	**ST**		**FX3U(C)**	**FX3G(C)**	**FX3S**	**FX2N(C)**	**FX1N(C)**	**FX1S**	**FXU/FX2C**	**FX0N**	**FX0(S)**	
GT	>	Compares data with regard to "> (larger)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.11
GE	>=	Compares data with regard to "≥ (larger or equal)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.12
EQ	=	Compares data with regard to "= (equal)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.13
LE	<=	Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.14
LT	<	Compares data with regard to "< (smaller)". (Number of pins variable)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.15
NE	<>	Compares data with regard to "≠ (unequal)".	✓	✓	✓	✓	✓	✓	✓	✓	✓	Section 15.16

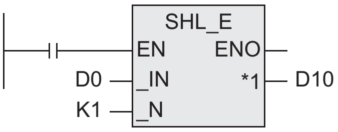
3. Function Construction

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.



_IN (s) : An argument whose contents do not change even if the function is executed is called "source", and expressed in this symbol.

*1 (d) : An argument whose contents change when the function is executed is called "destination", and expressed in this symbol.

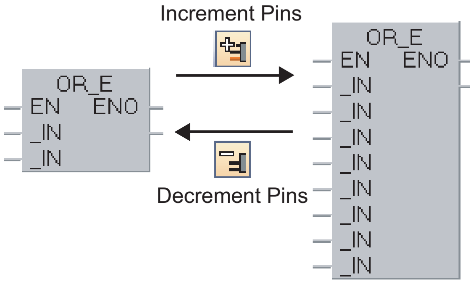
K1 (n) : Arguments not regarded as source or destination are expressed in "m", "n", etc.

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.
  → **FX Structured Programming Manual [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.
  → **Refer to Section 2. Function List**
- Changing the number of pins



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.

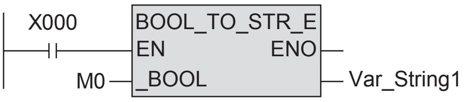
Class	Description	Applicable program component		
		Program block	Function	Function block
VAR_GLOBAL	Common label available in all program components	✓		✓
VAR_GLOBAL_CONSTANT	Common constant available in all program components	✓		✓
VAR	Label available within declared program components, and not available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

Label definition

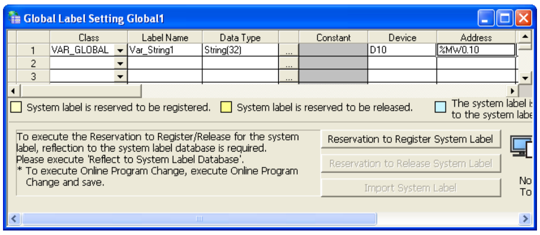
It is necessary to define a label to use the label.
An error will occur when a program in which labels are not defined is converted (compiled).

- 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".



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



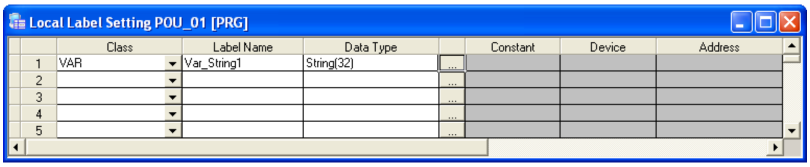
Global Label Setting Global1

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR_GLOBAL	Var_String1	String(32)	...		D10	%MW0.10
2				...			
3				...			

System label is reserved to be registered. System label is reserved to be released. The system label i to the system labe

To execute the Reservation to Register/Release for the system label, reflection to the system label database is required.
Please execute 'Reflect to System Label Database'.
* To execute Online Program Change, execute Online Program Change and save.

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



Local Label Setting POU_01 [PRG]

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR	Var_String1	String(32)	...			
2				...			
3				...			
4				...			
5				...			

Constant description method

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

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

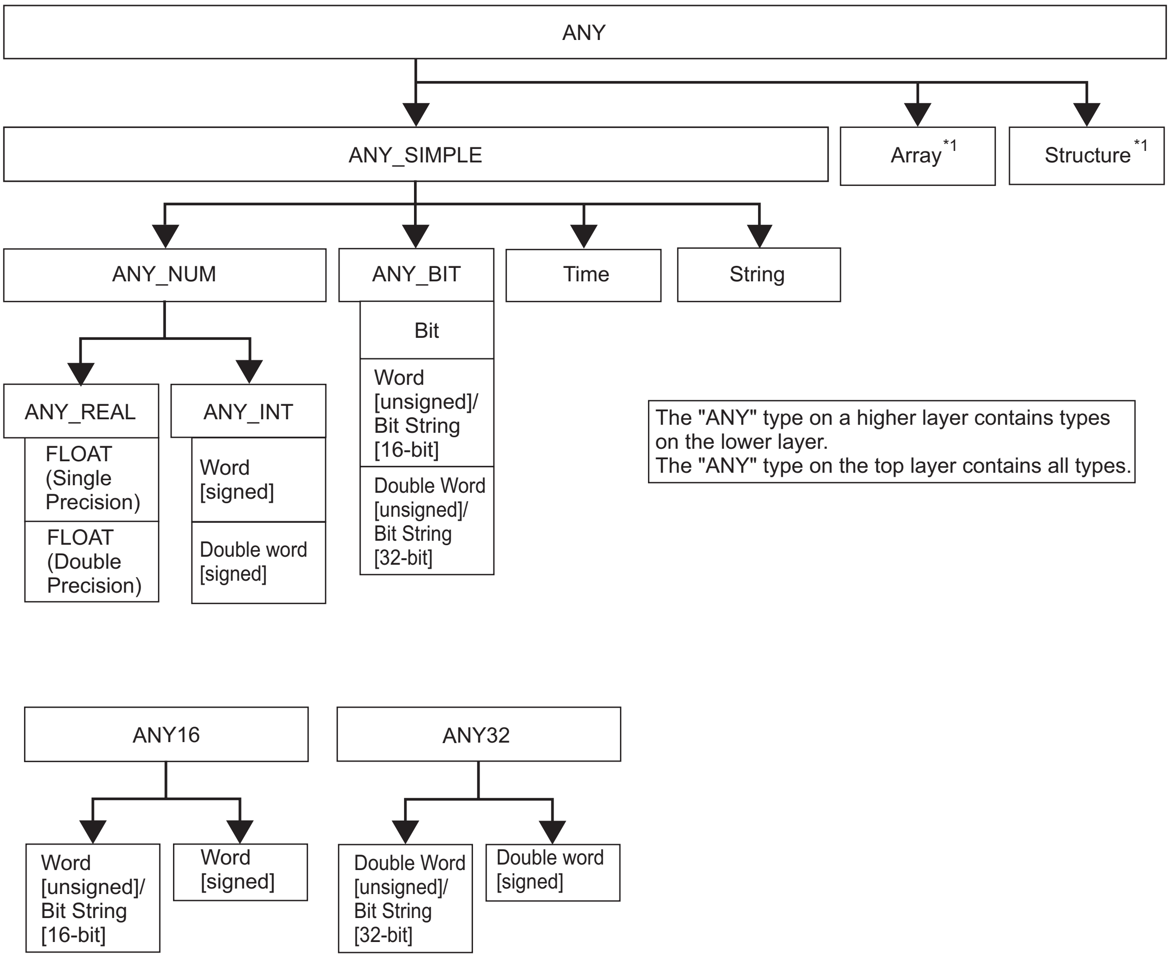
Data type

The label data type is basic or universal.

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

Data type	Description	Value range	Bit length
Bit	Boolean data	0(FALSE), 1(TRUE)	1 bit
Word [signed]	Integer	-32768 to 32767	16 bits
Double Word [signed]	Double precision integer	-2147483648 to 2147483647	32 bits
Word [unsigned]/Bit String [16-bit]	16-bit data	0 to 65535	16 bits
Double Word [unsigned]/Bit String [32-bit]	32-bit data	0 to 4294967295	32 bits
FLOAT (Single Precision)	Real number	E ±1.175495^{-38} to E ±3.402823^{+38} (Number of significant figures: 6)	32 bits
String	Character string	(50 characters maximum)	Variable
Time	Time value	T#-24d-0h31m23s648.00ms to T#24d20h31m23s647.00ms	32 bits

- The universal data type indicates data type of a label which combines several basic data types. The data type name begins with "ANY".



---

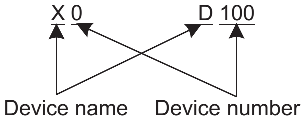
[1] Refer to the following manual for details.
→ Q/L/F Structured Programming Manual (Fundamentals)

3.3 Device and Address

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

Device method

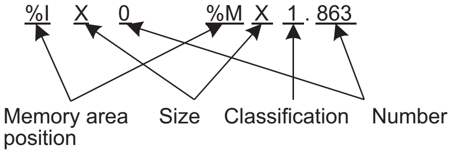
In this method, a device is described using the device name and device number.



Address method

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

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



- 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:

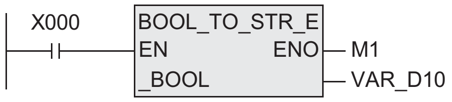
→ **7.3 Appendix A**

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.

EN	ENO	Operation result
TRUE(Executes operation.)	TRUE(Operation error did not occur.)	Operation output value
	FALSE(Operation error occurred.)	Indefinite value
FALSE(Stops operation.)	FALSE	Indefinite value

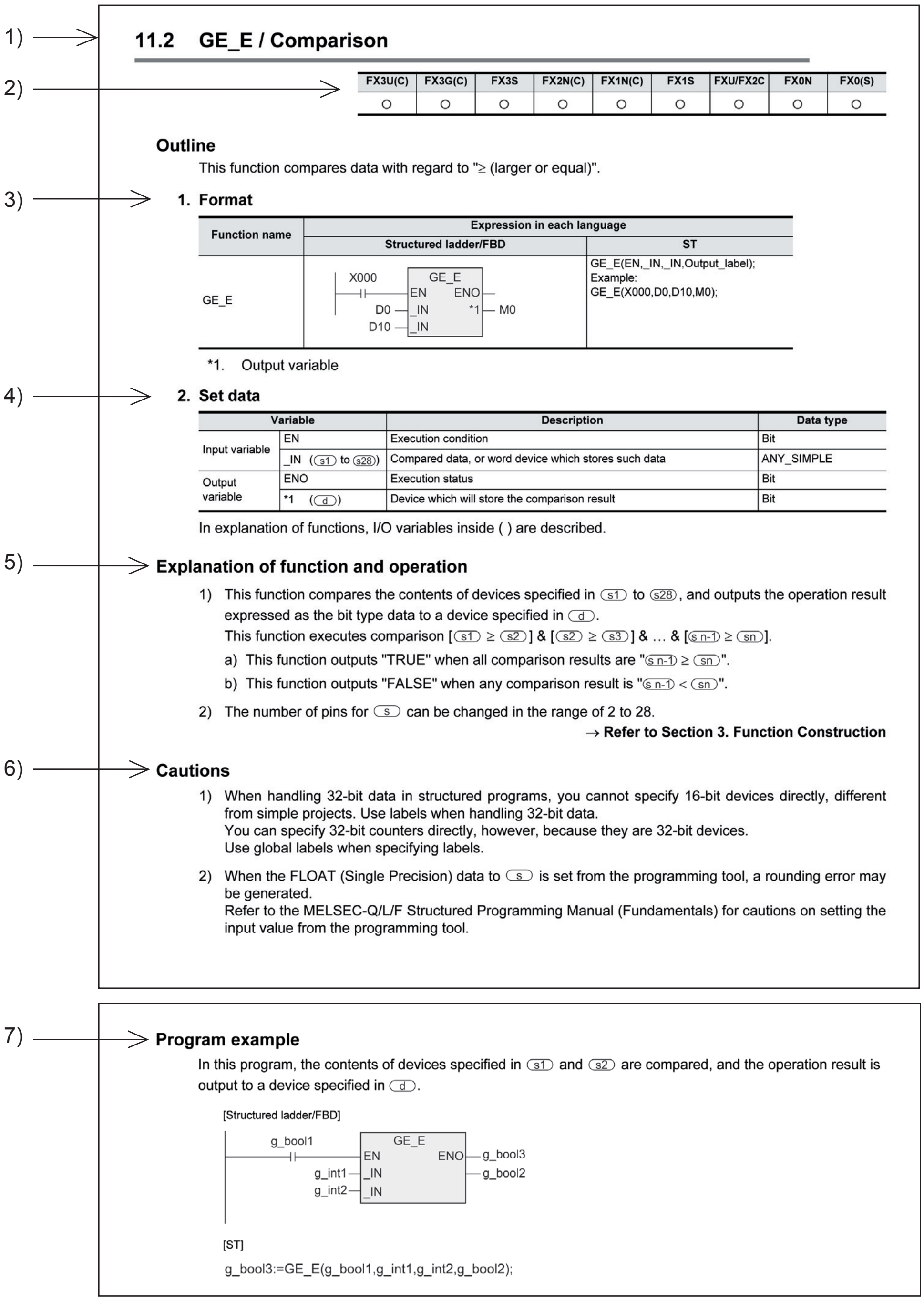


In the above example, the function "BOOL_TO_STR_E"
is executed only when X000 is "TRUE".
When the function is executed normally, "TRUE" is output to M1.

MEMO

4. How to Read Explanation of Functions

Function explanation pages have the following configuration.



1) **11.2 GE_E / Comparison**

2)

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≥ (larger or equal)".

3) **1. Format**

Function name	Expression in each language	
	Structured ladder/FBD	ST
GE_E	X000 GE_E EN ENO D0 _IN *1 M0 D10 _IN	GE_E(EN,_IN,_IN,Output_label); Example: GE_E(X000,D0,D10,M0);

*1. Output variable

4) **2. Set data**

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

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

5) **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 d.
This function executes comparison [s1 ≥ s2] & [s2 ≥ s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

6) **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.

7) **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]

[ST]

```
g_bool3:=GE_E(g_bool1,g_int1,g_int2,g_bool2);
```

* The above page is prepared for explanation, and is different from the actual page.

1) Indicates the chapter/section/subsection number and instruction name.

2) Indicates PLCs which support the function.

Item	Description
○	The PLC Series supports the function from its first product.
△	The supporting status varies on the version. Applicable versions are explained in "Cautions".
×	The PLC Series does not support the function.

3) Indicates the expression of each function.

Item	Description
Structured ladder /FBD	Indicates the instruction expression in the structured ladder language adopted as the representative.
ST	Indicates the instruction expression in the ST language.

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)

Function name	Function	Reference
BOOL_TO_INT(_E)	Bit data → word [signed] data conversion	Section 5.1
BOOL_TO_DINT(_E)	Bit data → double word [signed] data conversion	Section 5.2
BOOL_TO_STR(_E)	Bit data → string data conversion	Section 5.3
BOOL_TO_WORD(_E)	Bit data → word [unsigned]/bit string [16-bit] data conversion	Section 5.4
BOOL_TO_DWORD(_E)	Bit data → double word [unsigned]/bit string [32-bit] data conversion	Section 5.5
BOOL_TO_TIME(_E)	Bit data → time data conversion	Section 5.6
INT_TO_DINT(_E)	Word [signed] data → double word [signed] data conversion	Section 5.7
DINT_TO_INT(_E)	Double word [signed] data → word [signed] data conversion	Section 5.8
INT_TO_BOOL(_E)	Word [signed] data → bit data conversion	Section 5.9
DINT_TO_BOOL(_E)	Double word [signed] data → bit data conversion	Section 5.10
INT_TO_REAL(_E)	Word [signed] data → float (single precision) data conversion	Section 5.11
DINT_TO_REAL(_E)	Double word [signed] data → float (single precision) data conversion	Section 5.12
INT_TO_STR(_E)	Word [signed] data → string data conversion	Section 5.13
DINT_TO_STR(_E)	Double word [signed] data → string data conversion	Section 5.14
INT_TO_WORD(_E)	Word [signed] data → word [unsigned]/bit string [16-bit] data conversion	Section 5.15
DINT_TO_WORD(_E)	Double word [signed] data → word [unsigned]/bit string [16-bit] data conversion	Section 5.16
INT_TO_DWORD(_E)	Word [signed] data → double word [unsigned]/bit string [32-bit] data conversion	Section 5.17
DINT_TO_DWORD(_E)	Double word [signed] data → double word [unsigned]/bit string [32-bit] data conversion	Section 5.18
INT_TO_BCD(_E)	Word [signed] data → BCD data conversion	Section 5.19
DINT_TO_BCD(_E)	Double word [signed] data → BCD data conversion	Section 5.20
INT_TO_TIME(_E)	Word [signed] data → time data conversion	Section 5.21
DINT_TO_TIME(_E)	Double word [signed] data → time data conversion	Section 5.22
REAL_TO_INT(_E)	Float (single precision) data → word [signed] data conversion	Section 5.23
REAL_TO_DINT(_E)	Float (single precision) data → double word [signed] data conversion	Section 5.24
REAL_TO_STR(_E)	Float (single precision) data → string data conversion	Section 5.25
WORD_TO_BOOL(_E)	Word [unsigned]/bit string [16-bit] data → bit data conversion	Section 5.26
DWORD_TO_BOOL(_E)	Double word [unsigned]/bit string [32-bit] data → bit data conversion	Section 5.27
WORD_TO_INT(_E)	Word [unsigned]/bit string [16-bit] data → word [signed] data conversion	Section 5.28
WORD_TO_DINT(_E)	Word [unsigned]/bit string [16-bit] data → double word [signed] data conversion	Section 5.29
DWORD_TO_INT(_E)	Double word [unsigned]/bit string [32-bit] data → Word [signed] data conversion	Section 5.30
DWORD_TO_DINT(_E)	Double word [unsigned]/bit string [32-bit] data → double word [signed] data conversion	Section 5.31
WORD_TO_DWORD(_E)	Word [unsigned]/bit string [16-bit] data → double word [unsigned]/bit string [32-bit] data conversion	Section 5.32
DWORD_TO_WORD(_E)	Double word [unsigned]/bit string [32-bit] data → word [unsigned]/bit string [16-bit] data conversion	Section 5.33
WORD_TO_TIME(_E)	Word [unsigned]/bit string [16-bit] data → time data conversion	Section 5.34
DWORD_TO_TIME(_E)	Double word [unsigned]/bit string [32-bit] data → time data conversion	Section 5.35
STR_TO_BOOL(_E)	String data → bit data conversion	Section 5.36
STR_TO_INT(_E)	String data → word [signed] data conversion	Section 5.37
STR_TO_DINT(_E)	String data → double word [signed] data conversion	Section 5.38
STR_TO_REAL(_E)	String data → float (single precision) data conversion	Section 5.39
STR_TO_TIME(_E)	String data → time data conversion	Section 5.40
BCD_TO_INT(_E)	BCD data → word [signed] data conversion	Section 5.41
BCD_TO_DINT(_E)	BCD data → double word [signed] data conversion	Section 5.42
TIME_TO_BOOL(_E)	Time data → bit data conversion	Section 5.43
TIME_TO_INT(_E)	Time data → word [signed] data conversion	Section 5.44
TIME_TO_DINT(_E)	Time data → double word [signed] data conversion	Section 5.45

Function name	Function	Reference
TIME_TO_STR(_E)	Time data → string data conversion	Section 5.46
TIME_TO_WORD(_E)	Time data → word [unsigned]/bit string [16-bit] data conversion	Section 5.47
TIME_TO_DWORD(_E)	Time data → double word [unsigned]/bit string [32-bit] data conversion	Section 5.48
BITARR_TO_INT(_E)	Bit array → Word [signed] type, word [unsigned]/bit String [16-bit] data conversion	Section 5.49
BITARR_TO_DINT(_E)	Bit array → Double word [signed] type, double word [unsigned]/bit string [32-bit] data conversion	Section 5.50
INT_TO_BITARR(_E)	Word [signed] data, word [unsigned]/bit string [16-bit] data → bit array conversion	Section 5.51
DINT_TO_BITARR(_E)	Double word [signed] data, double word [unsigned]/bit string [32-bit] data → bit array conversion	Section 5.52
CPY_BITARR(_E)	Bit array copy	Section 5.53
GET_BIT_OF_INT(_E)	Specified bit read of word [signed] data	Section 5.54
SET_BIT_OF_INT(_E)	Specified bit write of word [signed] data	Section 5.55
CPY_BIT_OF_INT(_E)	Specified bit copy of word [signed] data	Section 5.56
GET_BOOL_ADDR	Acquisition of start data	Section 5.57
GET_INT_ADDR	Acquisition of start data	Section 5.58
GET_WORD_ADDR	Acquisition of start data	Section 5.59

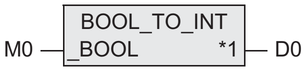
5.1 BOOL_TO_INT(_E) / Bit data → word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BOOL_TO_INT		BOOL_TO_INT(_BOOL); Example: D0:= BOOL_TO_INT(M0);
BOOL_TO_INT_E		BOOL_TO_INT_E(EN,_BOOL, Output_label Example: BOOL_TO_INT_E(X000,M0, D0);

*1. Output variable

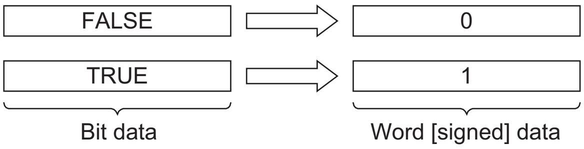
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

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 (s) into word [signed] data, and outputs the data obtained by conversion to a device specified in (d).
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

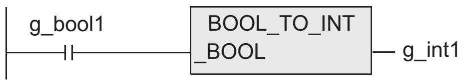
Use the function having "_E" in its name to connect a bus.

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]

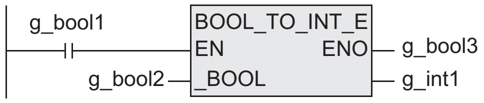


[ST]

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

2) Function with EN/ENO(BOOL_TO_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BOOL_TO_INT_E(g_bool1, g_bool2, g_int1);
```

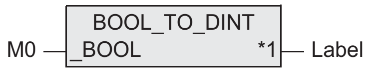
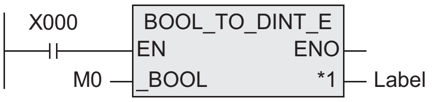
5.2 BOOL_TO_DINT(_E) / Bit data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BOOL_TO_DINT		BOOL_TO_DINT(_BOOL); Example: Label:= BOOL_TO_DINT(M0);
BOOL_TO_DINT_E		BOOL_TO_DINT_E(EN, _BOOL, Output_label); Example: BOOL_TO_DINT_E(X000,M0, Label);

*1. Output variable

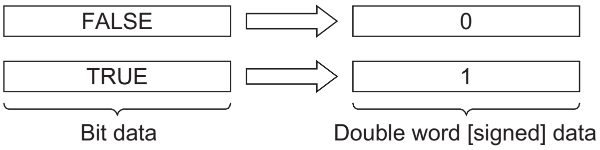
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

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 (s) into double word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

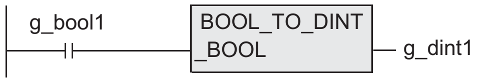
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, 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(BOOL_TO_DINT)

[Structured ladder/FBD]

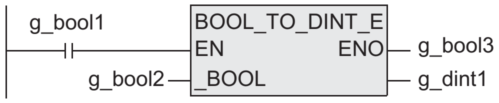


[ST]

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

2) Function with EN/ENO(BOOL_TO_DINT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BOOL_TO_DINT_E(g_bool1, g_bool2, g_dint1);
```

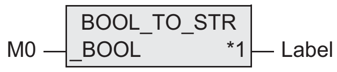
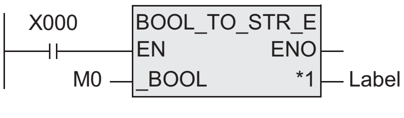
5.3 BOOL_TO_STR(_E) / Bit data → string data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BOOL_TO_STR		BOOL_TO_STR(_BOOL); Example: Label:= BOOL_TO_STR(M0);
BOOL_TO_STR_E		BOOL_TO_STR_E(EN, _BOOL, Output_label); Example: BOOL_TO_STR_E(X000,M0, Label);

*1. Output variable

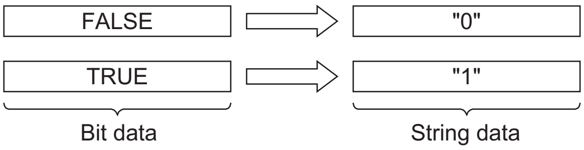
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

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

Explanation of function and operation

This function converts bit data input to a deice specified in s into string 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 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 deice 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(BOOL_TO_STR)

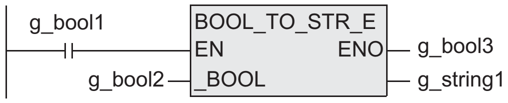
[Structured ladder/FBD]

[ST]

```
g_string1 := BOOL_TO_STR(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_STR_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BOOL_TO_STR_E(g_bool1, g_bool2, g_string1);
```

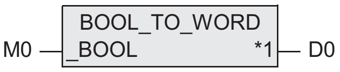
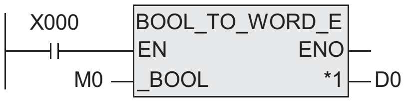
5.4 BOOL_TO_WORD(_E) / Bit data → word [unsigned]/ bit string [16-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BOOL_TO_WORD		BOOL_TO_WORD(_BOOL); Example: D0:= BOOL_TO_WORD(M0);
BOOL_TO_WORD_E		BOOL_TO_WORD_E(EN, _BOOL, Output_label); Example: BOOL_TO_WORD_E(X000, M0,D0);

*1. Output variable

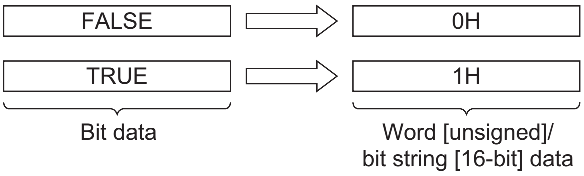
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (ⓢ)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

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

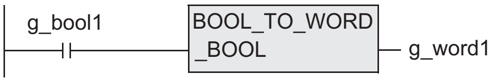
Use the function having "_E" in its name to connect a bus.

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]

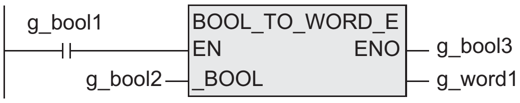


[ST]

```
g_word1 := BOOL_TO_WORD(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_WORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BOOL_TO_WORD_E(g_bool1, g_bool2, g_word1);
```

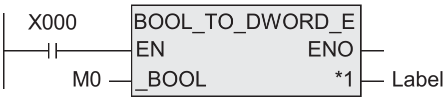
5.5 BOOL_TO_DWORD(_E) / Bit data → double word [unsigned]/ bit string [32-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BOOL_TO_DWORD		BOOL_TO_DWORD(_BOOL); Example: Label:= BOOL_TO_DWORD(M0);
BOOL_TO_DWORD_E		BOOL_TO_DWORD_E(EN, _BOOL, Output_label); Example: BOOL_TO_DWORD_E(X000, M0, Label);

*1. Output variable

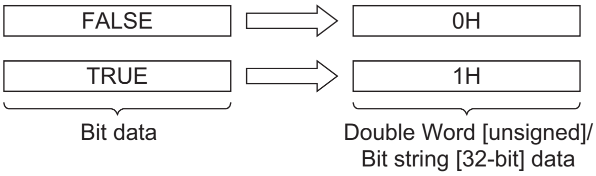
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

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 s into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in d.
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.



Cautions

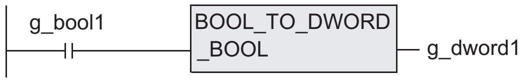
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, bit data stored in a device specified in 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(BOOL_TO_DWORD)

[Structured ladder/FBD]

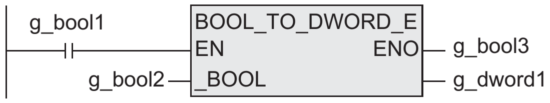


[ST]

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

2) Function with EN/ENO(BOOL_TO_DWORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BOOL_TO_DWORD_E(g_bool1, g_bool2, g_dword1);
```

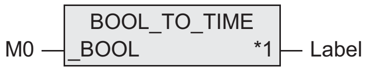
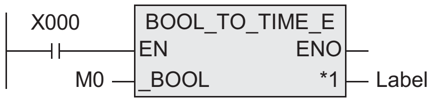
5.6 BOOL_TO_TIME(_E) / Bit data → time data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BOOL_TO_TIME		BOOL_TO_TIME(_BOOL); Example: Label:= BOOL_TO_TIME(M0);
BOOL_TO_TIME_E		BOOL_TO_TIME_E(EN,_BOOL, Output_label); Example: BOOL_TO_TIME_E(X000,M0, Label);

*1. Output variable

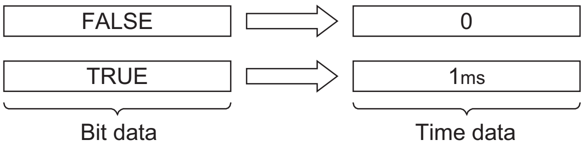
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

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 s into time data, and outputs the data obtained by conversion to a device specified in d.



Cautions

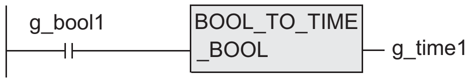
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, 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(BOOL_TO_TIME)

[Structured ladder/FBD]

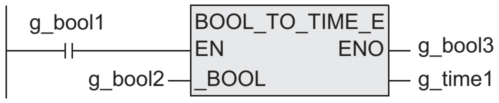


[ST]

```
g_time1 := BOOL_TO_TIME(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BOOL_TO_TIME_E(g_bool1, g_bool2, g_time1);
```

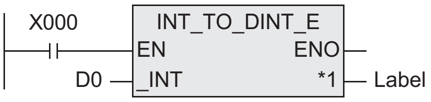
5.7 INT_TO_DINT(_E) / Word [signed] data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_DINT		INT_TO_DINT(_INT); Example: Label:= INT_TO_DINT(D0);
INT_TO_DINT_E		INT_TO_DINT_E(EN,_INT, Output_label); Example: INT_TO_DINT_E(X000,D0, Label);

*1. Output variable

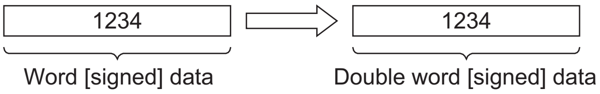
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (ⓢ)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [signed] data after conversion	Double Word [signed]

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 ⓓ.



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 double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(INT_TO_DINT)

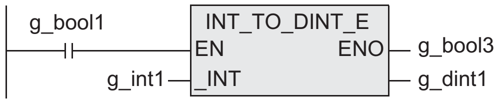
[Structured ladder/FBD]

[ST]

```
g_dint1 := INT_TO_DINT(g_int1);
```

2) Function with EN/ENO(INT_TO_DINT_E)

[Structured ladder/FBD]



[ST]

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

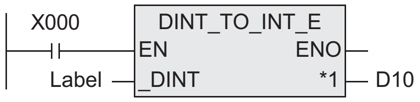
5.8 DINT_TO_INT(_E) / Double word [signed] data → word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_INT		DINT_TO_INT(_DINT); Example: D10:= DINT_TO_INT(Label);
DINT_TO_INT_E		DINT_TO_INT_E(EN,_DINT, Output_label); Example: DINT_TO_INT_E(X000, Label, D10);

*1. Output variable

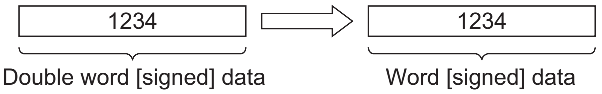
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (ⓢ)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Word [signed] data after conversion	Word [signed]

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

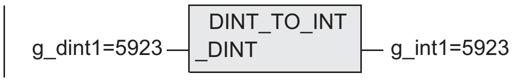
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

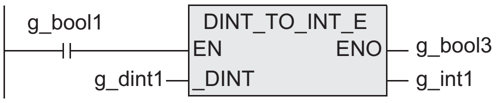


[ST]

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

2) Function with EN/ENO(DINT_TO_INT_E)

[Structured ladder/FBD]



[ST]

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

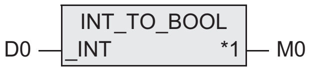
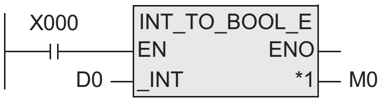
5.9 INT_TO_BOOL(_E) / Word [signed] data → bit data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_BOOL		INT_TO_BOOL(_INT); Example: M0:= INT_TO_BOOL(D0);
INT_TO_BOOL_E		INT_TO_BOOL_E(EN,_INT, Output_label); Example: INT_TO_BOOL_E(X000,D0,M0);

*1. Output variable

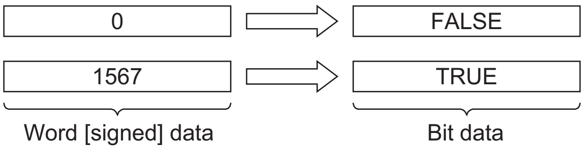
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

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 s into bit data, and outputs the data obtained by conversion to a device specified in d.
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

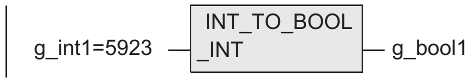
Use the function having "_E" in its name to connect a bus.

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]

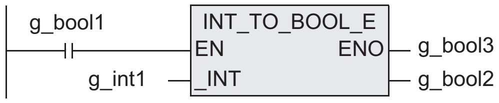


[ST]

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

2) Function with EN/ENO(INT_TO_BOOL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := INT_TO_BOOL_E(g_bool1, g_int1, g_bool2);
```

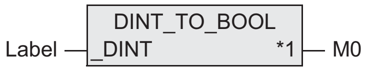
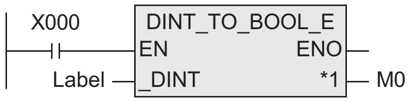
5.10 DINT_TO_BOOL(_E) / Double word [signed] data → bit data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_BOOL		DINT_TO_BOOL(_DINT); Example: M0:= DINT_TO_BOOL(Label);
DINT_TO_BOOL_E		DINT_TO_BOOL_E(EN,_DINT, Output_label); Example: DINT_TO_BOOL_E(X000, Label, M0);

*1. Output variable

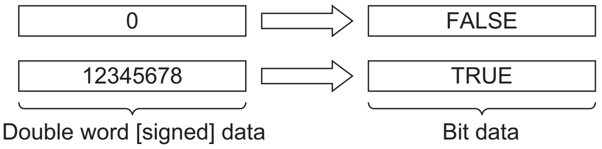
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

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 s into bit data, and outputs the data obtained by conversion to a device specified in d.
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 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(DINT_TO_BOOL)

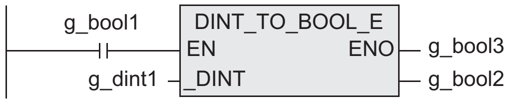
[Structured ladder/FBD]

[ST]

g_bool1 := DINT_TO_BOOL(g_dint1);

2) Function with EN/ENO(DINT_TO_BOOL_E)

[Structured ladder/FBD]



[ST]

g_bool3 := DINT_TO_BOOL_E(g_bool1, g_dint1, g_bool2);

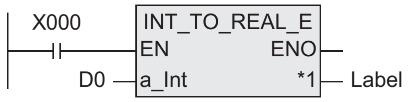
5.11 INT_TO_REAL(_E) / Word [signed] data → float (single precision) data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	○	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_REAL		INT_TO_REAL(a_Int); Example: Label:= INT_TO_REAL(D0);
INT_TO_REAL_E		INT_TO_REAL_E(EN,a_Int, Output_label); Example: INT_TO_REAL_E(X000,D0,Label);

*1. Output variable

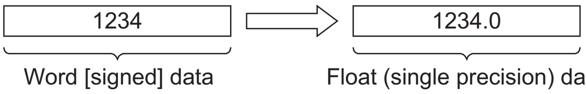
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_Int (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Float (single precision) data after conversion	FLOAT (Single Precision)

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 (s) into float (single precision) data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

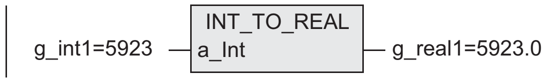
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

[Structured ladder/FBD]

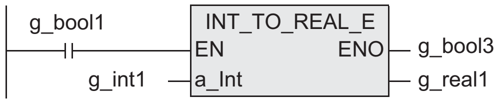


[ST]

```
g_real1 := INT_TO_REAL(g_int1);
```

2) Function with EN/ENO(INT_TO_REAL_E)

[Structured ladder/FBD]



[ST]

```
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

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	○	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_REAL		DINT_TO_REAL(a_Dint); Example: Label2:= DINT_TO_REAL(Label1);
DINT_TO_REAL_E		DINT_TO_REAL_E(EN,a_Dint, Output_label); Example: DINT_TO_REAL_E(X000, Label1, Label2);

*1. Output variable

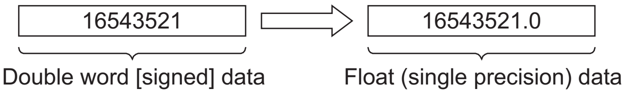
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_Dint (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Float (single precision) data after conversion	FLOAT (Single Precision)

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 (s) into float (single precision) data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

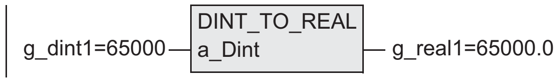
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

[Structured ladder/FBD]

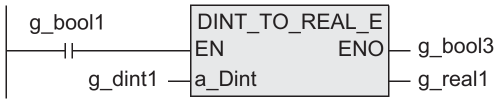


[ST]

```
g_real1 := DINT_TO_REAL(g_dint1);
```

2) Function with EN/ENO(DINT_TO_REAL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DINT_TO_REAL_E(g_bool1, g_dint1, g_real1);
```

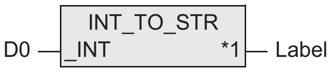
5.13 INT_TO_STR(_E) / Word [signed] data → string data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_STR		INT_TO_STR(_INT); Example: Label:= INT_TO_STR(D0);
INT_TO_STR_E		INT_TO_STR_E(EN,_INT, Output_label); Example: INT_TO_STR_E(X000, D0, Label);

*1. Output variable

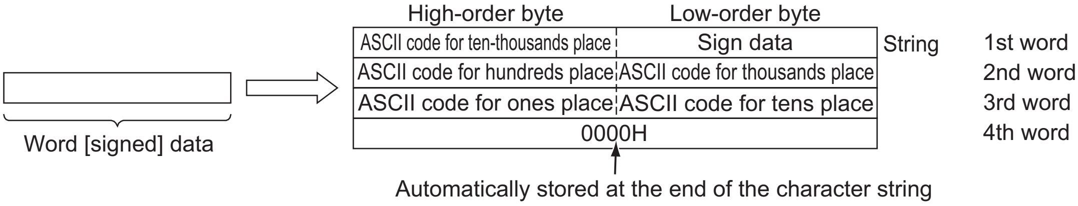
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (ⓢ)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	String data after conversion	String

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

Explanation of function and operation

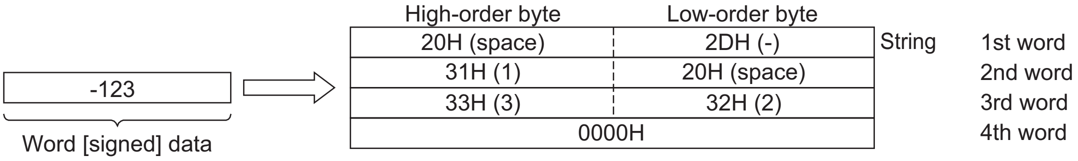
1) This function converts word [signed] data stored in a device specified in ⓢ into string data, and outputs the data obtained by conversion to a device specified in ⓓ.



High-order byte	Low-order byte		
ASCII code for ten-thousands place	Sign data	String	1st word
ASCII code for hundreds place	ASCII code for thousands place		2nd word
ASCII code for ones place	ASCII code for tens place		3rd word
0000H			4th word

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



High-order byte	Low-order byte		
20H (space)	2DH (-)	String	1st word
31H (1)	20H (space)		2nd word
33H (3)	32H (2)		3rd word
0000H			4th word

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.

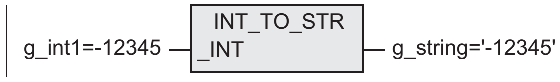
1) When the number of points occupied by the string data storage destination (device specified in d) exceeds the range of the corresponding device
(Error code: K6706)

Program example

In this program, word [signed] 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(INT_TO_STR)

[Structured ladder/FBD]

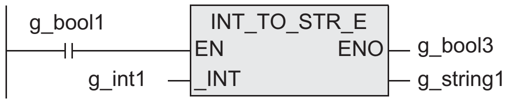


[ST]

```
g_string1 := INT_TO_STR(g_int1);
```

2) Function with EN/ENO(INT_TO_STR_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := INT_TO_STR_E(g_bool1, g_int1, g_string1);
```

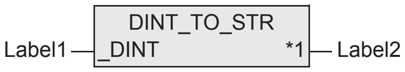
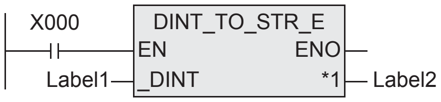
5.14 DINT_TO_STR(_E) / Double word [signed] data → string data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

<table>
<tr><th rowspan="2">Function name</th><th colspan="2">Expression in each language</th></tr>
<tr><th>Structured ladder/FBD</th><th>ST</th></tr>
<tr><td>DINT_TO_STR</td><td> </td><td>DINT_TO_STR(_DINT);
Example:
Label2:=
DINT_TO_STR(Label1);</td></tr>
<tr><td>DINT_TO_STR_E</td><td> </td><td>DINT_TO_STR_E(EN,_DINT,
Output_label);
Example:
DINT_TO_STR_E(X000,
Label1, Label2);</td></tr>
</table>

*1. Output variable

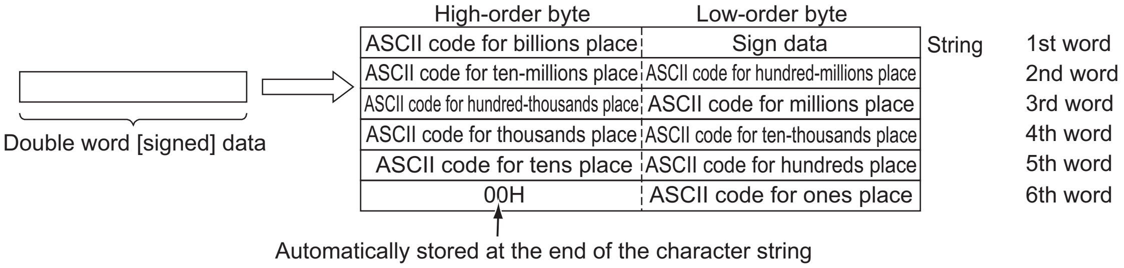
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (ⓢ)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	String data after conversion	String

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

Explanation of function and operation

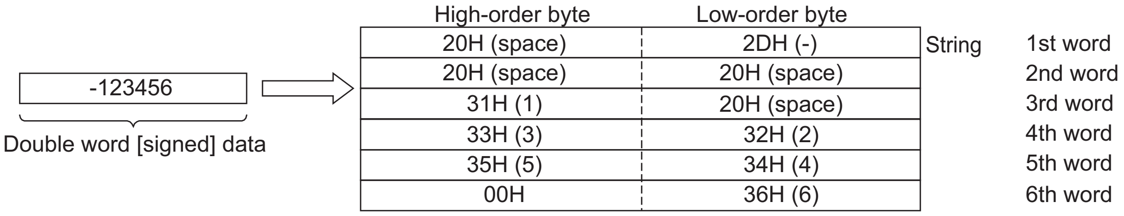
1) This function converts double word [signed] data stored in a device specified in ⓢ into string data, and outputs the data obtained by conversion to a device specified in ⓓ.



High-order byte	Low-order byte		
ASCII code for billions place	Sign data	String	1st word
ASCII code for ten-millions place	ASCII code for hundred-millions place		2nd word
ASCII code for hundred-thousands place	ASCII code for millions place		3rd word
ASCII code for thousands place	ASCII code for ten-thousands place		4th word
ASCII code for tens place	ASCII code for hundreds place		5th word
00H	ASCII code for ones place		6th word

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



High-order byte	Low-order byte		
20H (space)	2DH (-)	String	1st word
20H (space)	20H (space)		2nd word
31H (1)	20H (space)		3rd word
33H (3)	32H (2)		4th word
35H (5)	34H (4)		5th word
00H	36H (6)		6th word

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.

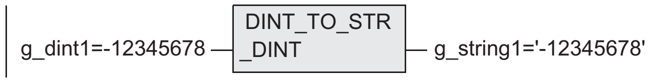
1) When the number of points occupied by the string data storage destination (device specified in d) exceeds the range of the corresponding device
   (Error code: K6706)

Program example

In this program, double word [signed] data stored in a device specified in 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(DINT_TO_STR)

[Structured ladder/FBD]

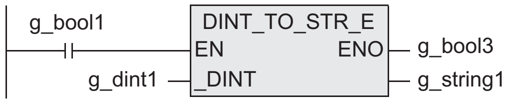


[ST]

```
g_string1 := DINT_TO_STR(g_dint1);
```

2) Function with EN/ENO(DINT_TO_STR_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DINT_TO_STR_E(g_bool1, g_dint1, g_string1);
```

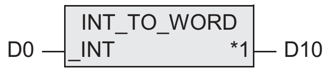
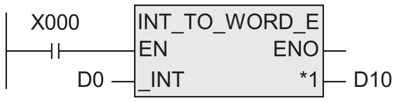
5.15 INT_TO_WORD(_E) / Word [signed] data → word [unsigned]/ bit string [16-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_WORD		INT_TO_WORD(_INT); Example: D10:= INT_TO_WORD(D0);
INT_TO_WORD_E		INT_TO_WORD_E(EN,_INT, Output_label); Example: INT_TO_WORD_E(X000,D0,D10);

*1. Output variable

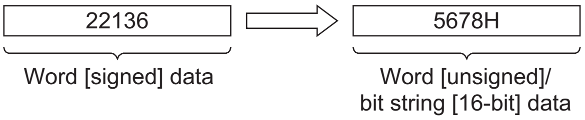
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/Bit String [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

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 s into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

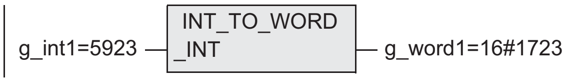
Use the function having "_E" in its name to connect a bus.

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]



[ST]

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

2) Function with EN/ENO(INT_TO_WORD_E)

[Structured ladder/FBD]

[ST]

```
g_bool3 := INT_TO_WORD_E(g_bool1, g_int1, g_word1);
```

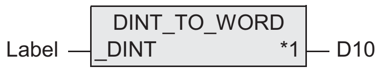
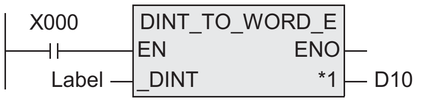
5.16 DINT_TO_WORD(_E) / Double word [signed] data → word [unsigned]/ bit string [16-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_WORD		DINT_TO_WORD(_DINT); Example: D10:= DINT_TO_WORD(Label);
DINT_TO_WORD _E		DINT_TO_WORD_E(EN,_DINT, Output_label); Example: DINT_TO_WORD_E(X000, Label, D10);

*1. Output variable

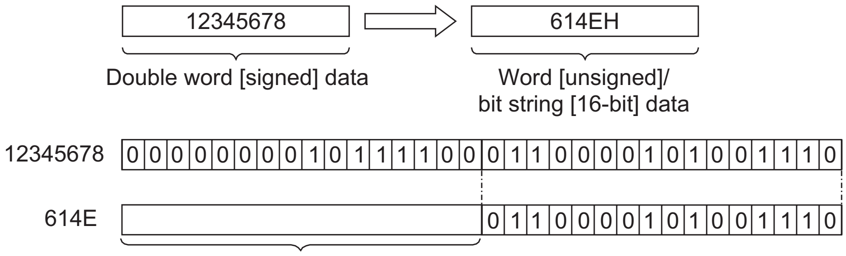
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

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 s into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in d.



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

Cautions

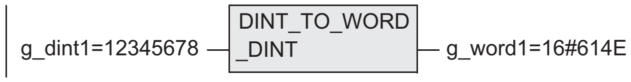
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, double word [signed] data stored in a device specified in 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(DINT_TO_WORD)

[Structured ladder/FBD]

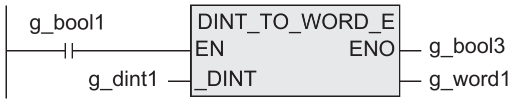


[ST]

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

2) Function with EN/ENO(DINT_TO_WORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DINT_TO_WORD_E(g_bool1, g_dint1, g_word1);
```

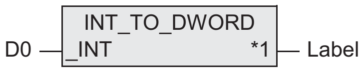
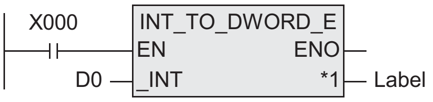
5.17 INT_TO_DWORD(_E) / word [signed] data → double word [unsigned]/ bit string [32-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_DWORD		INT_TO_DWORD(_INT); Example: Label:= INT_TO_DWORD(D0);
INT_TO_DWORD _E		INT_TO_DWORD_E(EN,_INT, Output_label); Example: INT_TO_DWORD_E(X000,D0, Label);

*1. Output variable

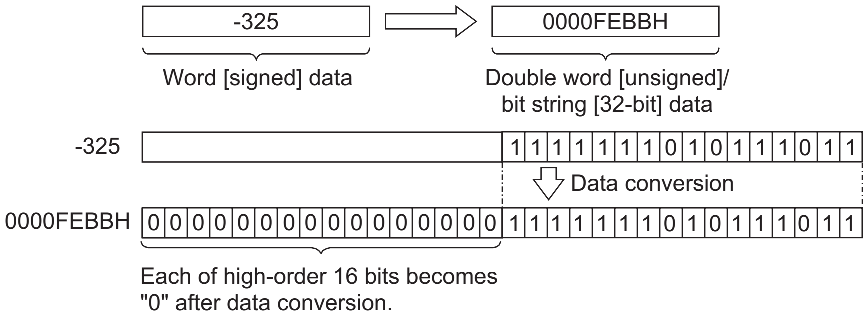
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (ⓢ)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

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 ⓓ.



Cautions

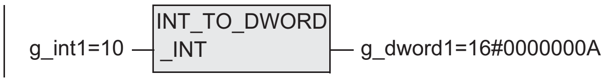
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, word [signed] data stored in a device specified in 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(INT_TO_DWORD)

[Structured ladder/FBD]

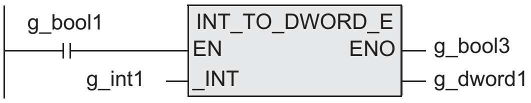


[ST]

```
g_dword1 := INT_TO_DWORD(g_int1);
```

2) Function with EN/ENO(INT_TO_DWORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := INT_TO_DWORD_E(g_bool1, g_int1, g_dword1);
```

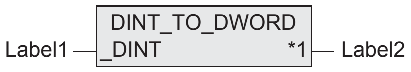
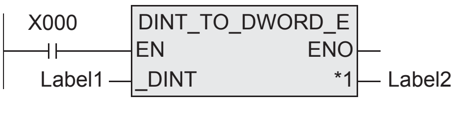
5.18 DINT_TO_DWORD(_E) / Double word [signed] data → double word [unsigned]/bit string [32-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_DWORD		DINT_TO_DWORD(_DINT); Example: Label2:= DINT_TO_DWORD(Label1);
DINT_TO_DWORD_E		DINT_TO_DWORD_E(EN,_DINT, Output_label); Example: DINT_TO_DWORD_E(X000, Label1, Label2);

*1. Output variable

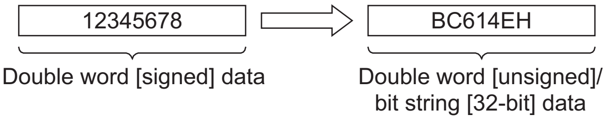
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

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



Cautions

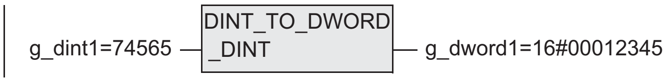
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, double word [signed] 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(DINT_TO_DWORD)

[Structured ladder/FBD]

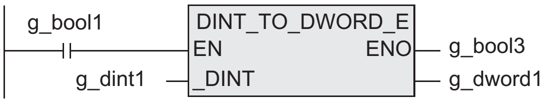


[ST]

```
g_dword1 := DINT_TO_DWORD(g_dint1);
```

2) Function with EN/ENO(DINT_TO_DWORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DINT_TO_DWORD_E(g_bool1, g_dint1, g_dword1);
```

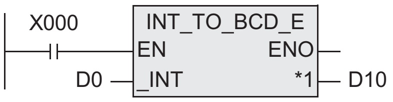
5.19 INT_TO_BCD(_E) / Word [signed] data → BCD data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_BCD		INT_TO_BCD(_INT); Example: D10:= INT_TO_BCD(D0);
INT_TO_BCD_E		INT_TO_BCD_E(EN,_INT, Output_label); Example: INT_TO_BCD_E(X000,D0,D10);

*1. Output variable

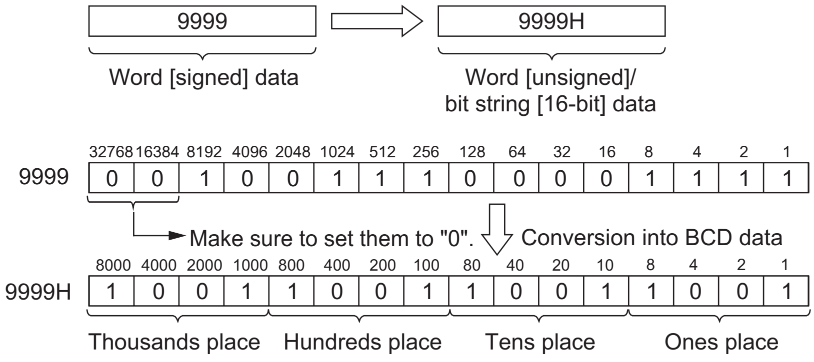
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	BCD data after conversion	Word [unsigned]/ Bit String [16-bit]

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 s into BCD data, and outputs the data obtained by conversion to a device specified in d.



Cautions

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

Error

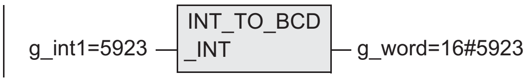
An operation error occurs when the value stored in a device specified in s is outside the range from "0" to "9,999".

Program example

In this program, 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(INT_TO_BCD)

[Structured ladder/FBD]

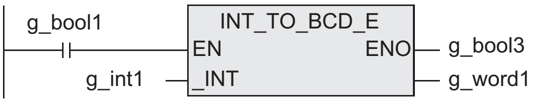


[ST]

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

2) Function with EN/ENO(INT_TO_BCD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := INT_TO_BCD_E(g_bool1, g_int1, g_word1);
```

5.20 DINT_TO_BCD(_E) / Double word [signed] data → BCD data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_BCD	DINT_TO_BCD Label1 — _DINT *1 — Label2	DINT_TO_BCD(_DINT); Example: Label2:= DINT_TO_BCD(Label1);
DINT_TO_BCD_E	X000 DINT_TO_BCD_E —┤├— EN ENO — Label1 — _DINT *1 — Label2	DINT_TO_BCD_E(EN,_DINT, Output_label); Example: DINT_TO_BCD_E(X000, Label1, Label2);

*1. Output variable

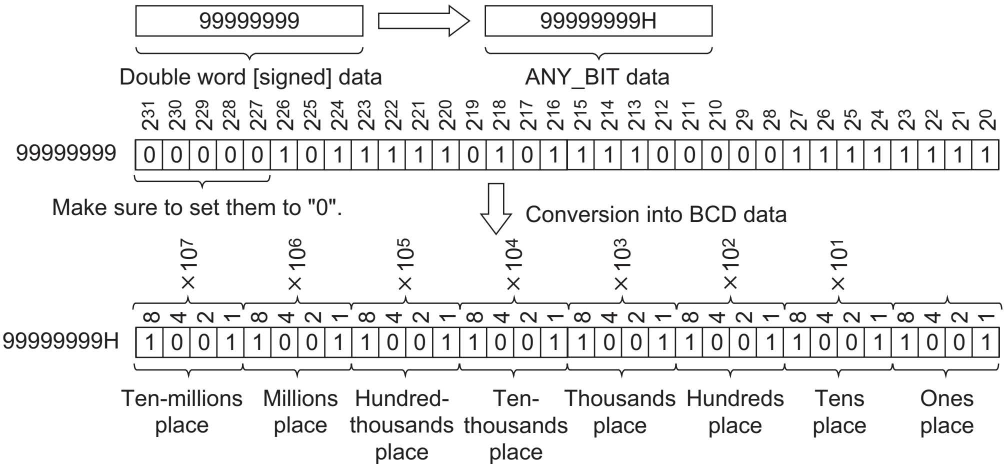
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	BCD data after conversion	ANY_BIT

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 (s) into BCD 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.

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)

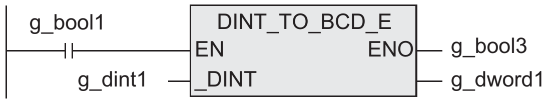
[Structured ladder/FBD]

[ST]

```
g_dword1 := DINT_TO_BCD(g_dint1);
```

2) Function with EN/ENO(DINT_TO_BCD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DINT_TO_BCD_E(g_bool1, g_dint1, g_dword1);
```

5.21 INT_TO_TIME(_E) / Word [signed] data → time data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_TIME		INT_TO_TIME(_INT); Example: Label:= INT_TO_TIME(D0);
INT_TO_TIME_E		INT_TO_TIME_E(EN,_INT, Output_label); Example: INT_TO_TIME_E(X000,D0,Label);

*1. Output variable

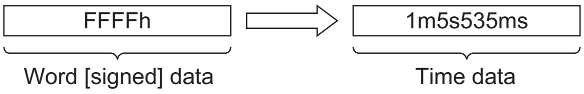
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

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 (s) into time data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

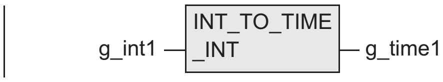
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, 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]

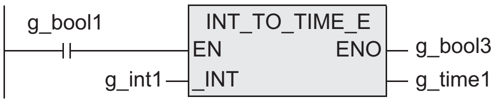


[ST]

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

2) Function with EN/ENO(INT_TO_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := INT_TO_TIME_E(g_bool1, g_int1, g_time1);
```

5.22 DINT_TO_TIME(_E) / Double word [signed] data → time data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_TIME		DINT_TO_TIME(_DINT); Example: Label2:= DINT_TO_TIME(Label1);
DINT_TO_TIME_E		DINT_TO_TIME_E(EN,_DINT, Output_label); Example: DINT_TO_TIME_E(X000, Label1, Label2);

*1. Output variable

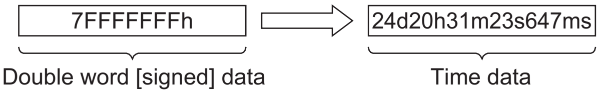
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

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 (s) into time data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

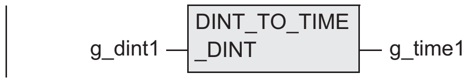
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, double 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(DINT_TO_TIME)

[Structured ladder/FBD]

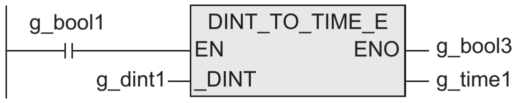


[ST]

```
g_time1 := DINT_TO_TIME(g_dint1);
```

2) Function with EN/ENO(DINT_TO_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DINT_TO_TIME_E(g_bool1, g_dint1, g_time1);
```

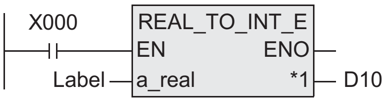
5.23 REAL_TO_INT(_E) / Float (single precision) data → word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	○	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
REAL_TO_INT		REAL_TO_INT(a_real); Example: D10:= REAL_TO_INT(Label);
REAL_TO_INT_E		REAL_TO_INT_E(EN,a_real, Output_label); Example: REAL_TO_INT_E(X000, Label, D10);

*1. Output variable

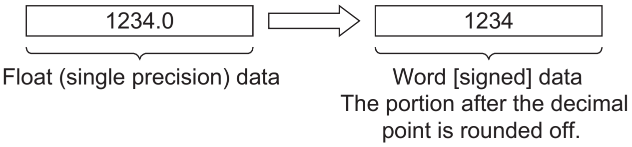
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_real (s)	Conversion source float (single precision) data	FLOAT (Single Precision)
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

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 s into word [signed] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

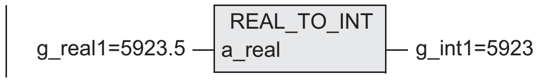
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

[Structured ladder/FBD]

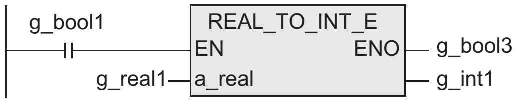


[ST]

```
g_int1 := REAL_TO_INT(g_real1);
```

2) Function with EN/ENO(REAL_TO_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := REAL_TO_INT_E(g_bool1, g_real1, g_int1);
```

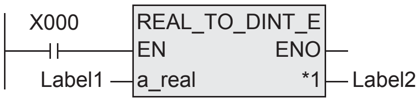
5.24 REAL_TO_DINT(_E) / Float (single precision) data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	○	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
REAL_TO_DINT		REAL_TO_DINT(a_real); Example: Label2:= REAL_TO_DINT(Label1);
REAL_TO_DINT_E		REAL_TO_DINT_E(EN,a_real, Output_label); Example: REAL_TO_DINT_E(X000, Label1, Label2);

*1. Output variable

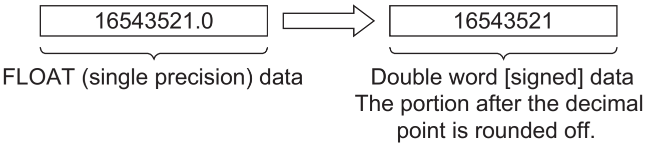
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_real (s)	Conversion source float (single precision) data	FLOAT (Single Precision)
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

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 (s) into double word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

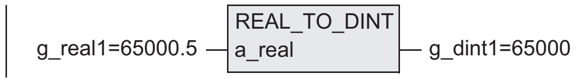
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

[Structured ladder/FBD]

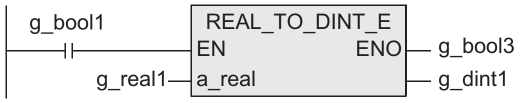


[ST]

```
g_dint1 := REAL_TO_DINT(g_real1);
```

2) Function with EN/ENO(DINT_TO_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := REAL_TO_DINT_E(g_bool1, g_real1, g_dint1);
```

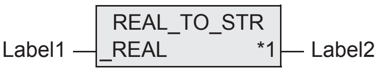
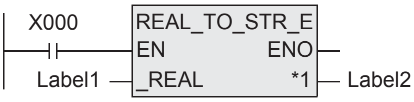
5.25 REAL_TO_STR(_E) / Float (single precision) data → string data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
REAL_TO_STR		REAL_TO_STR(_REAL); Example: Label2:= REAL_TO_STR(Label1);
REAL_TO_STR_E		REAL_TO_STR_E(EN,_REAL, Output_label); Example: REAL_TO_STR_E(X000, Label1, Label2);

*1. Output variable

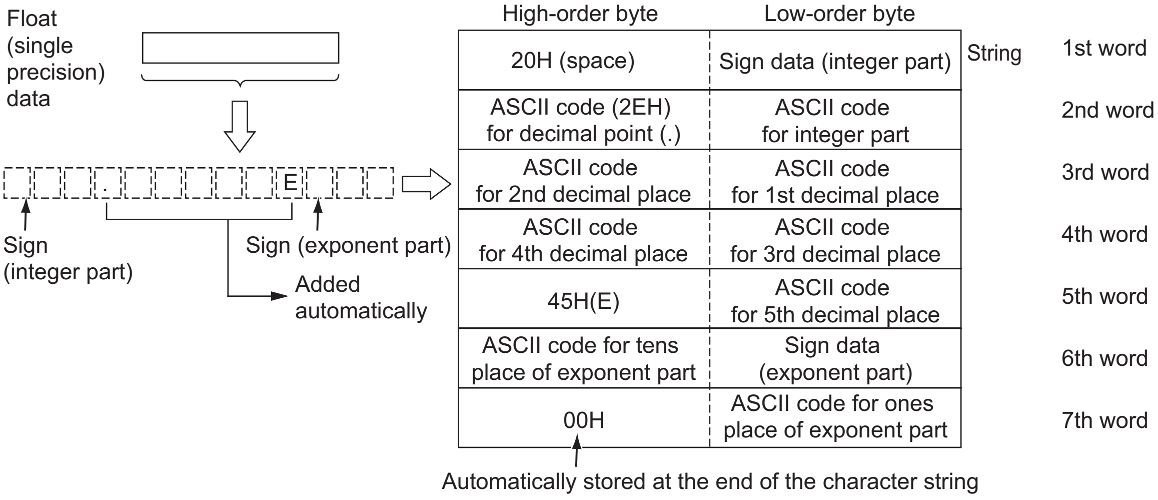
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_REAL (s)	Conversion source float (single precision) data	FLOAT (Single Precision)
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

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 (s) into string (exponent) data, and outputs the data obtained by conversion to a device specified in (d).

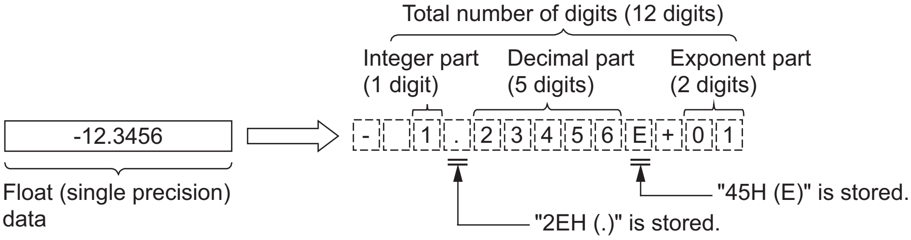


High-order byte	Low-order byte	
20H (space)	Sign data (integer part)	1st word
ASCII code (2EH) for decimal point (.)	ASCII code for integer part	2nd word
ASCII code for 2nd decimal place	ASCII code for 1st decimal place	3rd word
ASCII code for 4th decimal place	ASCII code for 3rd decimal place	4th word
45H(E)	ASCII code for 5th decimal place	5th word
ASCII code for tens place of exponent part	Sign data (exponent part)	6th word
00H	ASCII code for ones place of exponent part	7th word

Automatically stored at the end of the character string

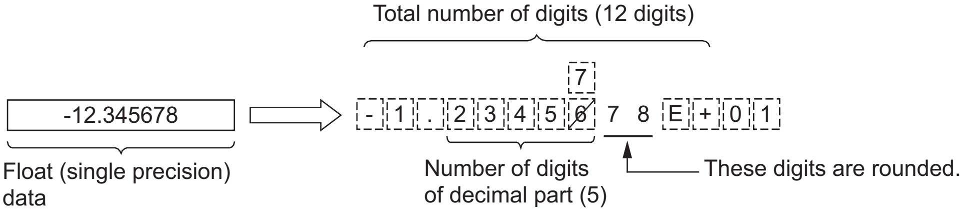
2) The string data obtained by conversion is output to a device specified in d as follows:

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.

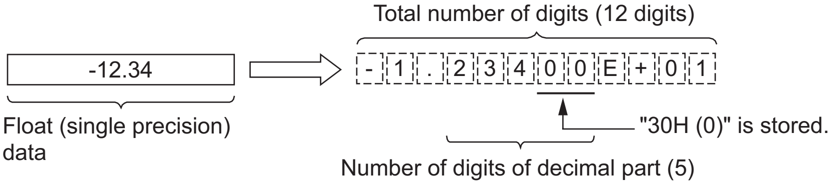


b) In "Sign data (integer part)", "20H (space)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

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

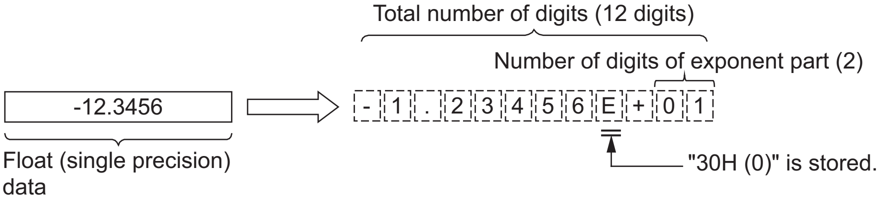


d) "30H (0)" is stored in the decimal part when the number of significant figures is small.



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 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

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

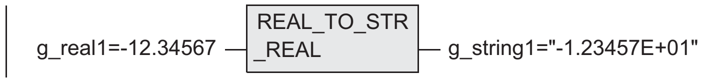
1) When the value stored in a device specified in s is outside the following range:
   $0, \pm 2^{-126} \leq$ (Value of device specified in s) $\leq \pm 2^{128}$
   (Error code: K6706)
2) When the range of a device which will store the character string obtained by conversion (device specified in d) exceeds 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 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(REAL_TO_STR)

[Structured ladder/FBD]

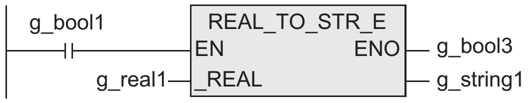


[ST]

```
g_string1 := REAL_TO_STR(g_real1);
```

2) Function with EN/ENO(REAL_TO_STR_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := REAL_TO_STR_E(g_bool1, g_real1, g_string1);
```

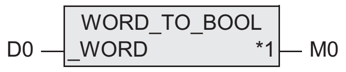
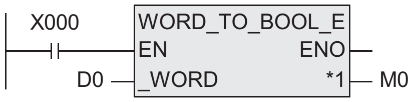
5.26 WORD_TO_BOOL(_E) / Word [unsigned] / bit string [16-bit] data → bit data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
WORD_TO_BOOL		WORD_TO_BOOL(_WORD); Example: M0:= WORD_TO_BOOL(D0);
WORD_TO_BOOL_E		WORD_TO_BOOL_E(EN, _WORD, Output_label); Example: WORD_TO_BOOL_E(X000,D0, M0);

*1. Output variable

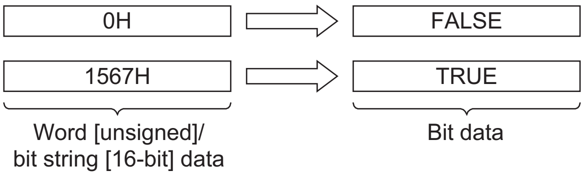
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit String [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

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 (s) into bit data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

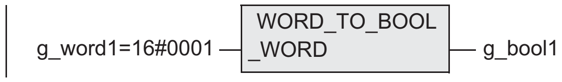
Use the function having "_E" in its name to connect a bus.

Program example

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

1) Function without EN/ENO(WORD_TO_BOOL)

[Structured ladder/FBD]

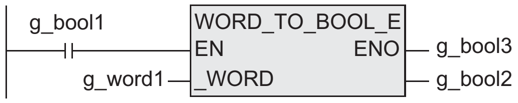


[ST]

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

2) Function with EN/ENO(WORD_TO_BOOL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := WORD_TO_BOOL_E(g_bool1, g_word1, g_bool2);
```

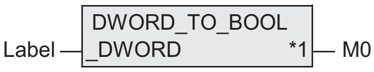
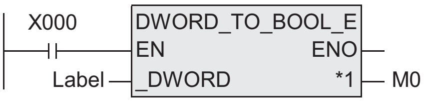
5.27 DWORD_TO_BOOL(_E) / Double word [unsigned]/bit string [32-bit] data → bit data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DWORD_TO_BOOL		DWORD_TO_BOOL(_DWORD); Example: M0:= DWORD_TO_BOOL(Label);
DWORD_TO_BOOL_E		DWORD_TO_BOOL_E(EN, _DWORD, Output_label); Example: DWORD_TO_BOOL_E(X000, Label, M0);

*1. Output variable

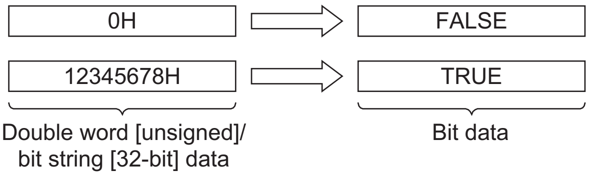
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (s)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

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 (s) into bit data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

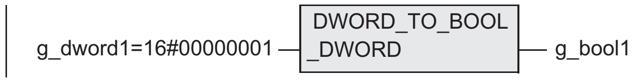
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, double word [unsigned]/bit string [32-bit] 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(DWORD_TO_BOOL)

[Structured ladder/FBD]

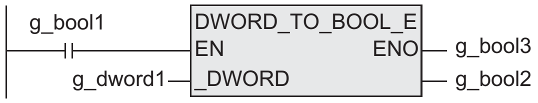


[ST]

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

2) Function with EN/ENO(DWORD_TO_BOOL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DWORD_TO_BOOL_E(g_bool1, g_dword1, g_bool2);
```

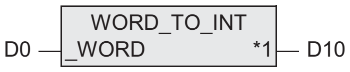
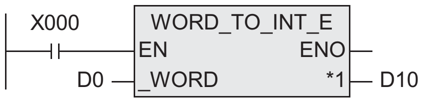
5.28 WORD_TO_INT(_E) / Word [unsigned]/bit string [16-bit] data → word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
WORD_TO_INT		WORD_TO_INT(_WORD); Example: D10:= WORD_TO_INT(D0);
WORD_TO_INT_E		WORD_TO_INT_E(EN,_WORD, Output_label); Example: WORD_TO_INT_E(X000,D0, D10);

*1. Output variable

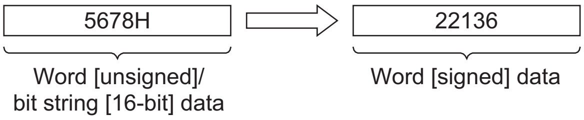
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

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 (s) into word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

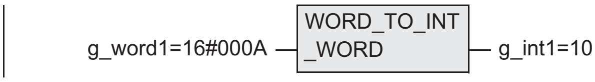
Use the function having "_E" in its name to connect a bus.

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]

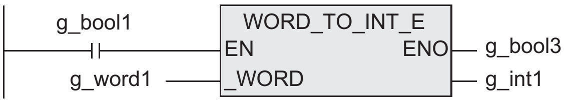


[ST]

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

2) Function with EN/ENO(WORD_TO_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := WORD_TO_INT_E(g_bool1, g_word1, g_int1);
```

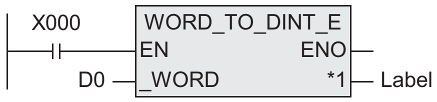
5.29 WORD_TO_DINT(_E) / Word [unsigned]/bit string [16-bit] data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
WORD_TO_DINT		WORD_TO_DINT(_WORD); Example: Label:= WORD_TO_DINT(D0);
WORD_TO_DINT _E		WORD_TO_DINT_E(EN,_WORD, Output_label); Example: WORD_TO_DINT_E(X000,D0, Label);

*1. Output variable

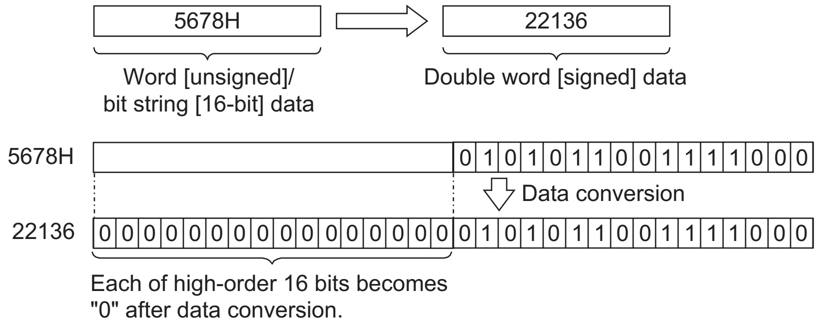
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (ⓢ)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [signed] data after conversion	Double Word [signed]

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 ⓓ.



Cautions

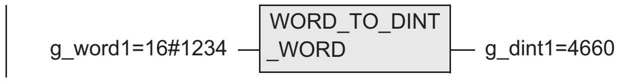
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

[Structured ladder/FBD]

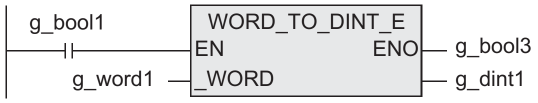


[ST]

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

2) Function with EN/ENO(WORD_TO_DINT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := WORD_TO_DINT_E(g_bool1, g_word1, g_dint1);
```

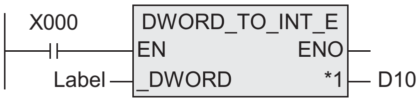
5.30 DWORD_TO_INT(_E) / Double word [unsigned]/bit string [32-bit] data → Word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
DWORD_TO_INT		DWORD_TO_INT(_DWORD); Example: D10:= DWORD_TO_INT(Label);
DWORD_TO_INT _E		DWORD_TO_INT_E(EN, _DWORD, Output_label); Example: DWORD_TO_INT_E(X000,Label, D10);

*1. Output variable

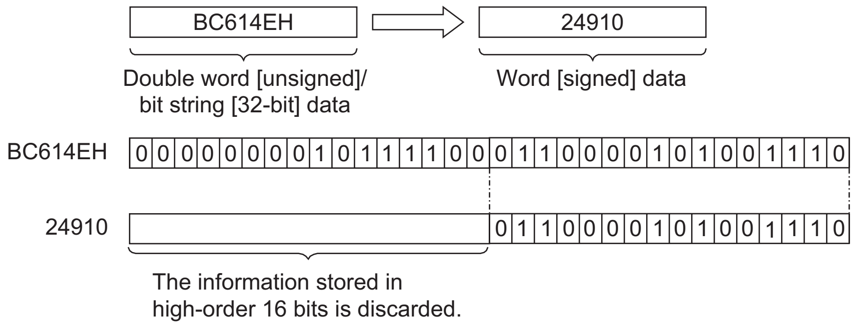
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (s)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

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 s into word [signed] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

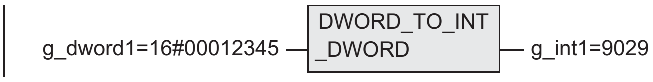
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, double word [unsigned]/bit string [32-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(DWORD_TO_INT)

[Structured ladder/FBD]

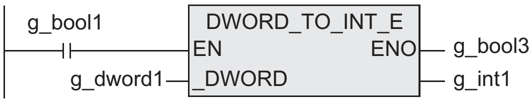


[ST]

```
g_int1 := DWORD_TO_INT(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DWORD_TO_INT_E(g_bool1, g_dword1, g_int1);
```

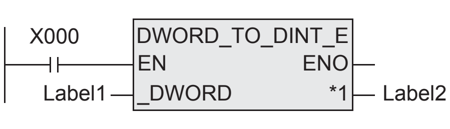
5.31 DWORD_TO_DINT(_E) / Double word [unsigned]/bit string [32-bit] data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
DWORD_TO_DINT		DWORD_TO_DINT(_DWORD); Example: Label2:= DWORD_TO_DINT(Label1);
DWORD_TO_DINT_E		DWORD_TO_DINT_E(EN, _DWORD, Output_label); Example: DWORD_TO_DINT_E(X000, Label1, Label2);

*1. Output variable

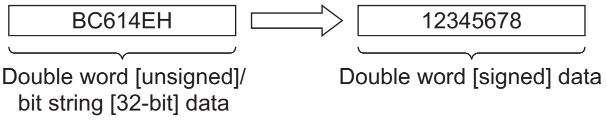
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (ⓢ)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [signed] data after conversion	Double Word [signed]

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

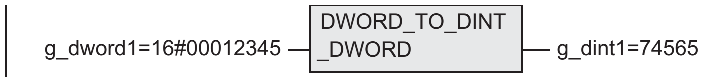
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

1) Function without EN/ENO(DWORD_TO_DINT)

[Structured ladder/FBD]

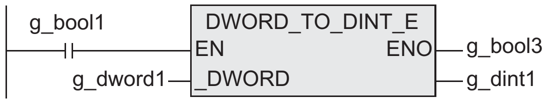


[ST]

```
g_dint1 := DWORD_TO_DINT(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_DINT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DWORD_TO_DINT_E(g_bool1, g_dword1, g_dint1);
```

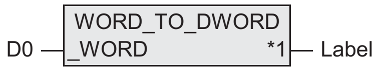
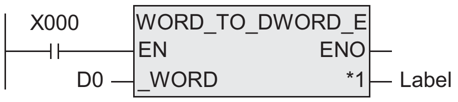
5.32 WORD_TO_DWORD(_E) / Word [unsigned]/bit string [16-bit] data → double word [unsigned]/bit string [32-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
WORD_TO_DWORD		WORD_TO_DWORD(_WORD); Example: Label:= WORD_TO_DWORD(D0);
WORD_TO_DWORD_E		WORD_TO_DWORD_E(EN, _WORD, Output_label); Example: WORD_TO_DWORD_E(X000,D0, Label);

*1. Output variable

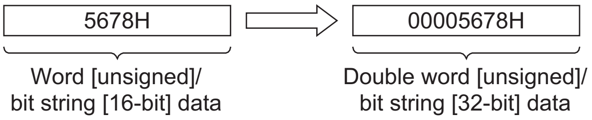
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word[unsigned]/bit string[32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

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 s into double word [unsigned]/bit [32-bit] data, and outputs the data obtained by conversion to a device specified in d. Each of high-order 16 bits becomes "0" after data conversion.



Cautions

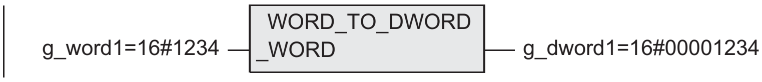
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

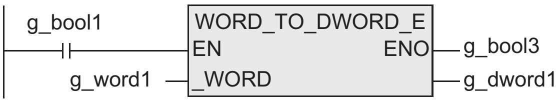


[ST]

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

2) Function with EN/ENO(WORD_TO_DWORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := WORD_TO_DWORD_E(g_bool1, g_word1, g_dword1);
```

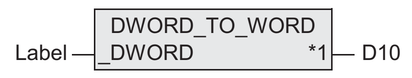
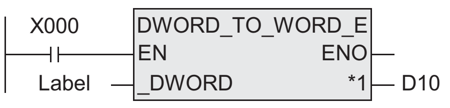
5.33 DWORD_TO_WORD(_E) / Double word [unsigned]/bit string[32-bit] data → word [unsigned]/bit string [16-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
DWORD_TO_WORD		DWORD_TO_WORD(_DWORD); Example: D10:= DWORD_TO_WORD(Label);
DWORD_TO_WORD_E		DWORD_TO_WORD_E(EN, _DWORD, Output_label); Example: DWORD_TO_WORD_E(X000, Label, D10);

*1. Output variable

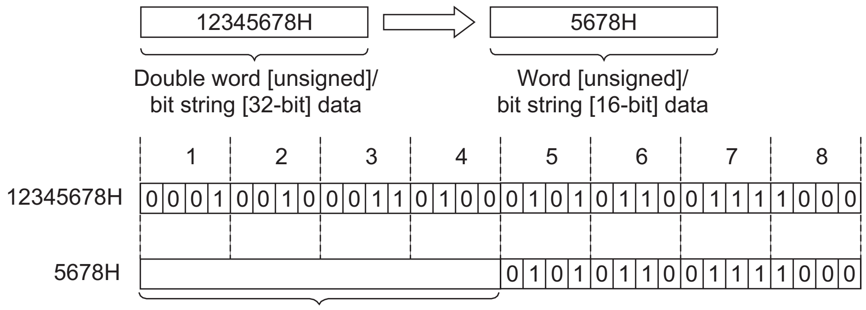
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (s)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned] /Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

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 s into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in d.



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

Cautions

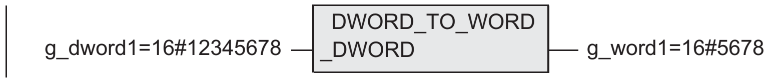
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

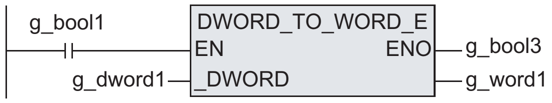


[ST]

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

2) Function with EN/ENO(WORD_TO_DWORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DWORD_TO_WORD_E(g_bool1, g_dword1, g_word1);
```

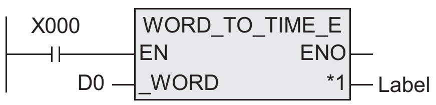
5.34 WORD_TO_TIME(_E) / Word [unsigned]/bit string [16-bit] data → time data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
WORD_TO_TIME		WORD_TO_TIME(_WORD); Example: Label:= WORD_TO_TIME(D0);
WORD_TO_TIME _E		WORD_TO_TIME_E(EN,_WORD, Output_label); Example: WORD_TO_TIME_E(X000,D0, Label);

*1. Output variable

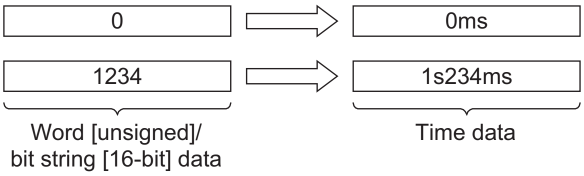
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (ⓢ)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String[16-bit]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Time data after conversion	Time

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 ⓓ.



Cautions

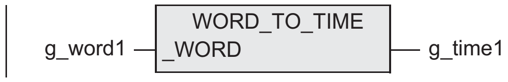
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

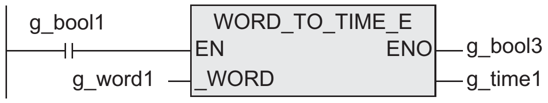


[ST]

```
g_time1 := WORD_TO_TIME(g_word1);
```

2) Function with EN/ENO(WORD_TO_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := WORD_TO_TIME_E(g_bool1, g_word1, g_time1);
```

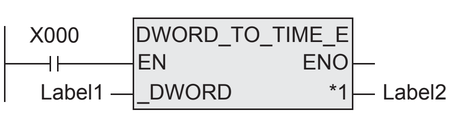
5.35 DWORD_TO_TIME(_E) / Double word [unsigned]/bit string [32-bit] data → time data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DWORD_TO_TIME		DWORD_TO_TIME(_DWORD); Example: Label2:= DWORD_TO_TIME(Label1);
DWORD_TO_TIME_E		DWORD_TO_TIME_E(EN,_ DWORD, Output_label); Example: DWORD_TO_TIME_E(X000, Label1, Label2);

*1. Output variable

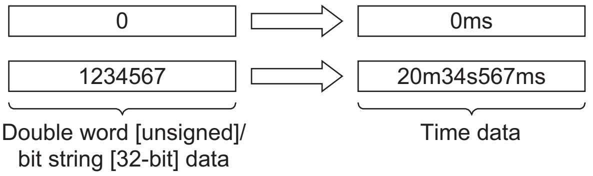
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD ((s))	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 ((d))	Time data after conversion	Time

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 (s) into time data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

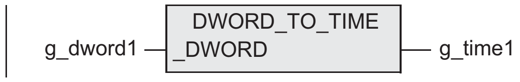
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, double word [unsigned]/bit string [32-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(DWORD_TO_TIME)

[Structured ladder/FBD]

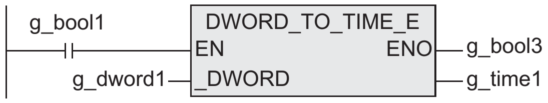


[ST]

```
g_time1 := DWORD_TO_TIME(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := DWORD_TO_TIME_E(g_bool1, g_dword1, g_time1);
```

5.36 STR_TO_BOOL(_E) / String data → bit data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
STR_TO_BOOL		STR_TO_BOOL(_STRING); Example: M0:= STR_TO_BOOL(Label);
STR_TO_BOOL_E		STR_TO_BOOL_E(EN,_STRING, Output_label); Example: STR_TO_BOOL_E(X000, Label, M0);

*1. Output variable

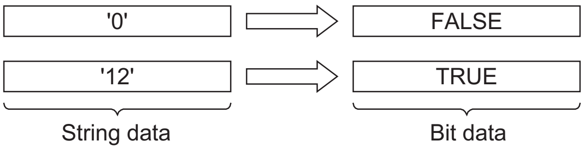
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

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 (s) into bit data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

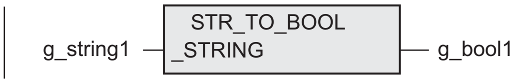
1) Use the function having "_E" in its name to connect a bus.
2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data.
Use global labels when specifying labels.

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]



[ST]

```
g_bool1 := STR_TO_BOOL(g_string1);
```

2) Function with EN/ENO(STR_TO_BOOL_E)

[Structured ladder/FBD]

[ST]

```
g_bool3 := STR_TO_BOOL_E(g_bool1, g_string1, g_bool2);
```

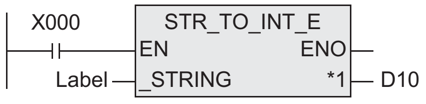
5.37 STR_TO_INT(_E) / String data → word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
STR_TO_INT		STR_TO_INT(_STRING); Example: D10:= STR_TO_INT(Label);
STR_TO_INT_E		STR_TO_INT_E(EN,_STRING, Output_label); Example: STR_TO_INT_E(X000, Label, D10);

*1. Output variable

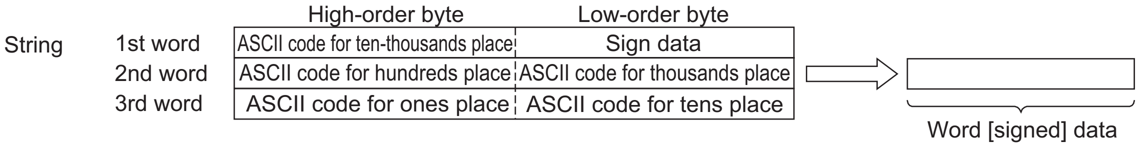
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

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 (s) into word [signed] 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 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)

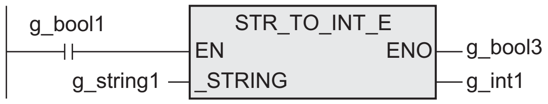
[Structured ladder/FBD]

[ST]

```
g_int1 := STR_TO_INT(g_string1);
```

2) Function with EN/ENO(STR_TO_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := STR_TO_INT_E(g_bool1, g_string1, g_int1);
```

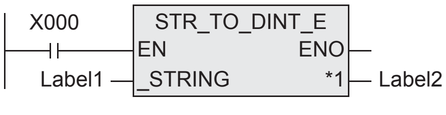
5.38 STR_TO_DINT(_E) / String data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
STR_TO_DINT		STR_TO_DINT(_STRING); Example: Label2:= STR_TO_DINT(Label1);
STR_TO_DINT_E		STR_TO_DINT_E(EN,_STRING, Output_label); Example: STR_TO_DINT_E(X000, Label1, Label2);

*1. Output variable

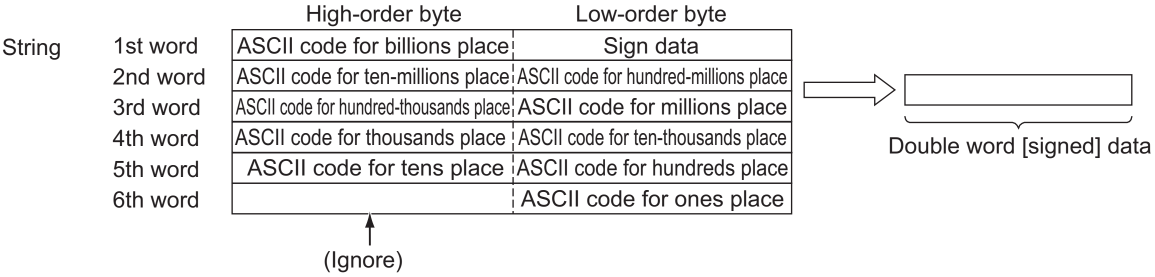
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

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 (s) into double word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



String	High-order byte	Low-order byte
1st word	ASCII code for billions place	Sign data
2nd word	ASCII code for ten-millions place	ASCII code for hundred-millions place
3rd word	ASCII code for hundred-thousands place	ASCII code for millions place
4th word	ASCII code for thousands place	ASCII code for ten-thousands place
5th word	ASCII code for tens place	ASCII code for hundreds place
6th word		ASCII code for ones place

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)

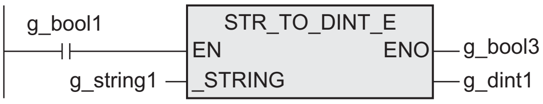
[Structured ladder/FBD]

[ST]

```
g_dint1 := STR_TO_DINT(g_string1);
```

2) Function with EN/ENO(STR_TO_DINT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := STR_TO_DINT_E(g_bool1, g_string1, g_dint1);
```

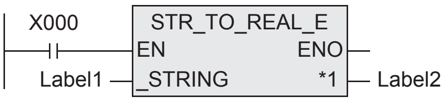
5.39 STR_TO_REAL(_E) / String data → float (single precision) data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
STR_TO_REAL		STR_TO_REAL(_STRING); Example: Label2:= STR_TO_REAL(Label1);
STR_TO_REAL_E		STR_TO_REAL_E(EN,_STRING, Output_label); Example: STR_TO_REAL_E(X000, Label1, Label2);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Float (single precision) data after conversion	FLOAT (Single Precision)

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 (s) into float (single precision) data, and outputs the data obtained by conversion to a device specified in (d).

String	High-order byte	Low-order byte
1st word	ASCII code for 1st character	Sign data
2nd word	ASCII code for 3rd character	ASCII code for 2nd character
3rd word	ASCII code for 5th character	ASCII code for 4th character
4th word	ASCII code for 7th character	ASCII code for 6th character
5th word	ASCII code for 9th character	ASCII code for 8th character
6th word	ASCII code for 11th character	ASCII code for 10th character
7th word	0000H (Indicates the end of the character string.)	

⇨ Float (single precision) data

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

a) In the case of decimal format

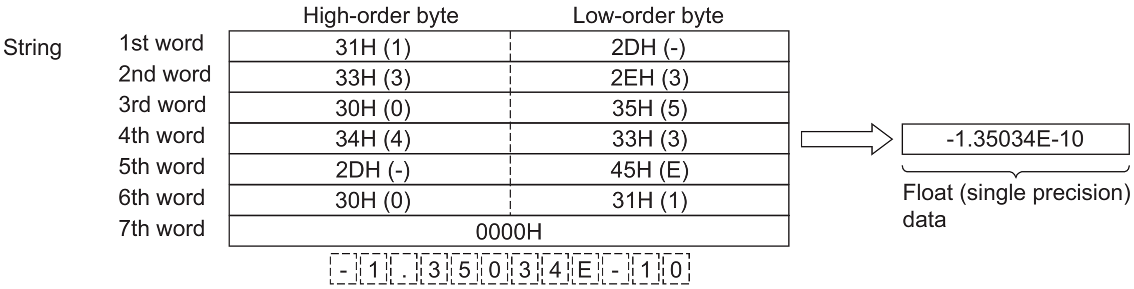
String	High-order byte	Low-order byte
1st word	31H (1)	2DH (-)
2nd word	33H (3)	2EH (.)
3rd word	30H (0)	35H (5)
4th word	34H (4)	33H (3)
5th word	0000H	

- 1 . 3 5 0 3 4

⇨ -1.35034

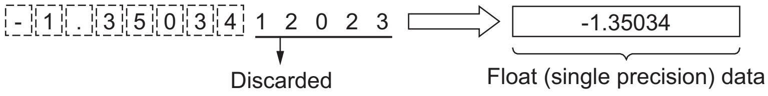
Float (single precision) data

b) In the case of exponent format

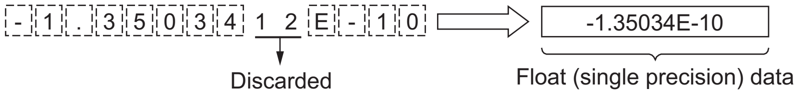


3) With regard to string data, six digits excluding the sign, decimal point and exponent part are valid, and the 7th and later digits are discarded during conversion.

a) In the case of decimal format



b) In the case of exponent format

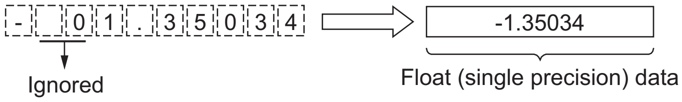


4) String data in the decimal format is handled as positive value during conversion when the sign is set to "2BH (+)" or when the sign is omitted. It is handled as negative value during conversion when the sign is set to "2DH (-)".

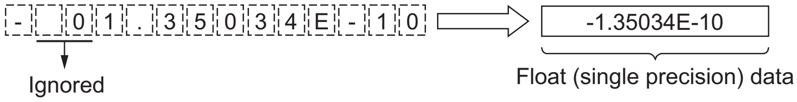
5) String data in the exponent format is handled as positive value during conversion when the sign of the exponent part is set to "2BH (+)" or when the sign is omitted. It is handled as negative value during conversion when the sign is set to "2DH (-)".

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.

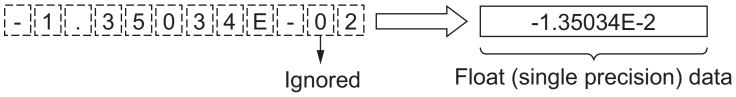
a) In the case of decimal format



b) In the case of exponent format



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



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 s 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)

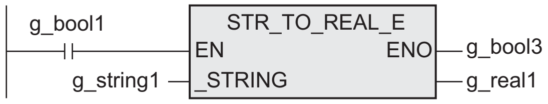
[Structured ladder/FBD]

[ST]

```
g_real1 := STR_TO_REAL(g_string1);
```

2) Function with EN/ENO(STR_TO_REAL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := STR_TO_REAL_E(g_bool1, g_string1, g_real1);
```

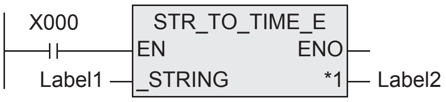
5.40 STR_TO_TIME(_E) / String data → time data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
STR_TO_TIME		STR_TO_TIME(_STRING); Example: Label2:= STR_TO_TIME(Label1);
STR_TO_TIME_E		STR_TO_TIME_E(EN,_STRING, Output_label); Example: STR_TO_TIME_E(X000, Label1, Label2);

*1. Output variable

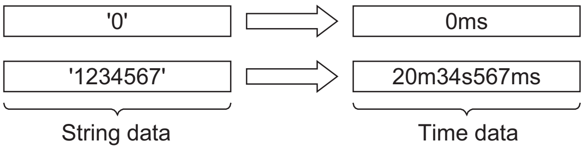
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (ⓢ)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Time data after conversion	Time

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 s 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 s is any other than "30H (0)" to "39H (9)", "20H (space)" or "00H (NULL)"
(Error code: K6706)

3) When the numeric value specified in s 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 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(STR_TO_TIME)

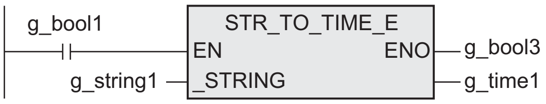
[Structured ladder/FBD]

[ST]

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

2) Function with EN/ENO(STR_TO_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := STR_TO_TIME_E(g_bool1, g_string1, g_time1);
```

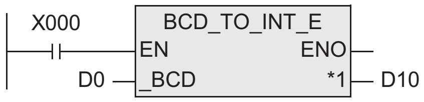
5.41 BCD_TO_INT(_E) / BCD data → word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BCD_TO_INT		BCD_TO_INT(_BCD); Example: D10:= BCD_TO_INT(D0);
BCD_TO_INT_E		BCD_TO_INT_E(EN,_BCD, Output_label); Example: BCD_TO_INT_E(X000,D0,D10);

*1. Output variable

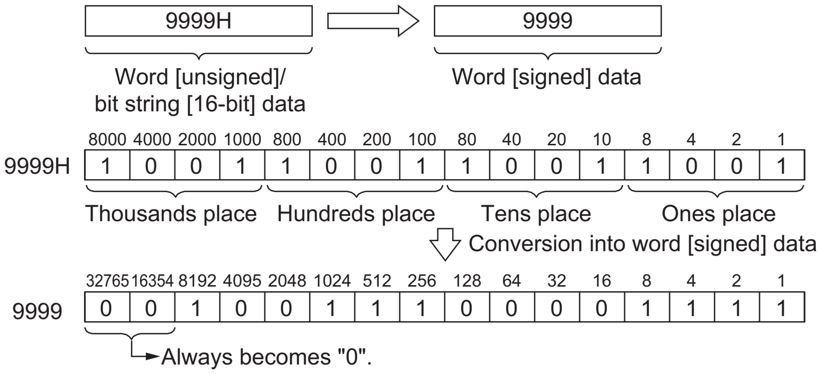
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BCD (s)	Conversion source BCD data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

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 (s) into word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



	8000	4000	2000	1000	800	400	200	100	80	40	20	10	8	4	2	1
9999H	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

	32765	16354	8192	4095	2048	1024	512	256	128	64	32	16	8	4	2	1
9999	0	0	1	0	0	1	1	1	0	0	0	0	1	1	1	1

Cautions

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

Error

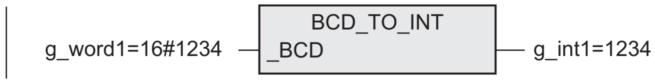
When the source data is not BCD (decimal number), M8067 (operation error) turns ON.

Program example

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

1) Function without EN/ENO(BCD_TO_INT)

[Structured ladder/FBD]

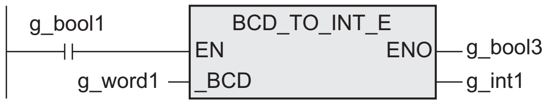


[ST]

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

2) Function with EN/ENO(BCD_TO_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BCD_TO_INT_E(g_bool1, g_word1, g_int1);
```

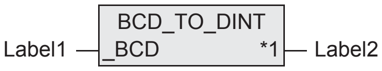
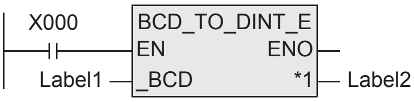
5.42 BCD_TO_DINT(_E) / BCD data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
BCD_TO_DINT		BCD_TO_DINT(_BCD); Example: Label2:= BCD_TO_DINT(Label1);
BCD_TO_DINT_E		BCD_TO_DINT_E(EN,_BCD, Output_label); Example: BCD_TO_DINT_E(X000, Label1, Label2);

*1. Output variable

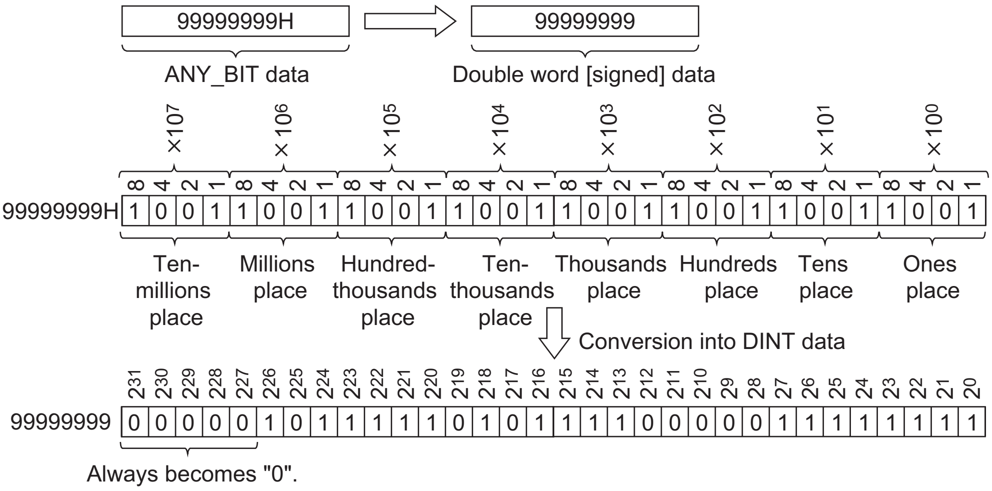
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BCD (s)	Conversion source BCD data	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

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 (s) into double word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

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

Error

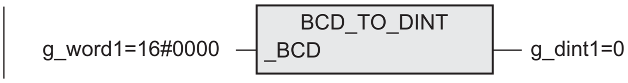
When the source data is not BCD (decimal number), M8067 (operation error) turns ON.

Program example

In this example, BCD data stored in a device specified in (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(BCD_TO_DINT)

[Structured ladder/FBD]

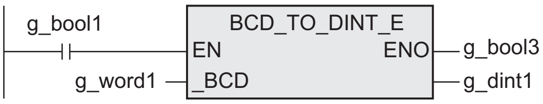


[ST]

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

2) Function with EN/ENO(BCD_TO_DINT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := BCD_TO_DINT_E(g_bool1, g_word1, g_dint1);
```

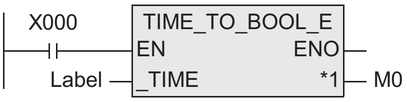
5.43 TIME_TO_BOOL(_E) / Time data → bit data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIME_TO_BOOL		TIME_TO_BOOL(_TIME); Example: M0:= TIME_TO_BOOL(Label);
TIME_TO_BOOL_E		TIME_TO_BOOL_E(EN,_TIME, Output_label); Example: TIME_TO_BOOL_E(X000, Label, M0);

*1. Output variable

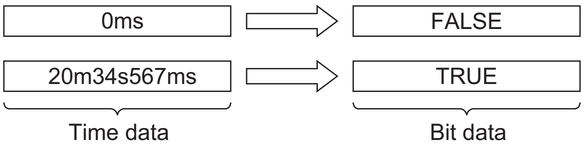
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

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

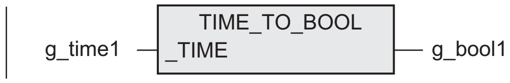
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, time data stored in a device specified in 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(TIME_TO_BOOL)

[Structured ladder/FBD]

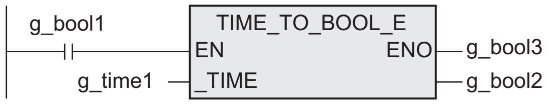


[ST]

```
g_bool1 := TIME_TO_BOOL(g_time1);
```

2) Function with EN/ENO(TIME_TO_BOOL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := TIME_TO_BOOL_E(g_bool1, g_time1, g_bool2);
```

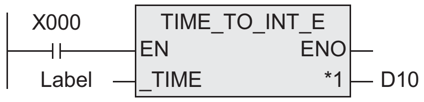
5.44 TIME_TO_INT(_E) / Time data → word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIME_TO_INT		TIME_TO_INT(_TIME); Example: D10:= TIME_TO_INT(Label);
TIME_TO_INT_E		TIME_TO_INT_E(EN,_TIME, Output_label); Example: TIME_TO_INT_E(X000, Label, D10);

*1. Output variable

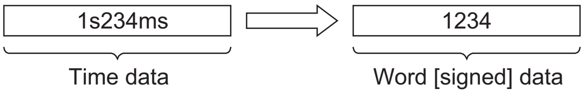
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

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 word [signed] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

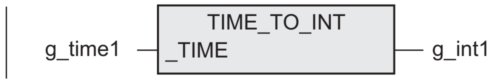
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, time data stored in a device specified in 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(TIME_TO_INT)

[Structured ladder/FBD]

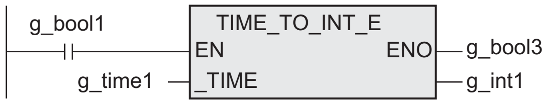


[ST]

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

2) Function with EN/ENO(TIME_TO_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := TIME_TO_INT_E(g_bool1, g_time1, g_int1);
```

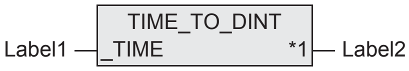
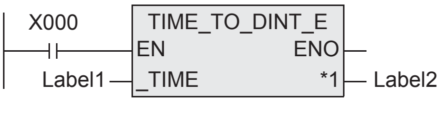
5.45 TIME_TO_DINT(_E) / Time data → double word [signed] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIME_TO_DINT		TIME_TO_DINT(_TIME); Example: Label2:= TIME_TO_DINT(Label1);
TIME_TO_DINT_E		TIME_TO_DINT_E(EN,_TIME, Output_label); Example: TIME_TO_DINT_E(X000,Label1, Label2);

*1. Output variable

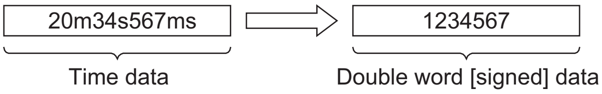
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (ⓢ)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [signed] data after conversion	Double Word [signed]

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

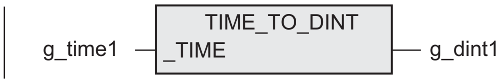
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, time data stored in a device specified in (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(TIME_TO_DINT)

[Structured ladder/FBD]

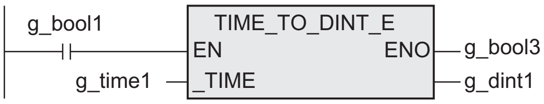


[ST]

```
g_dint1 := TIME_TO_DINT(g_time1);
```

2) Function with EN/ENO(TIME_TO_DINT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := TIME_TO_DINT_E(g_bool1, g_time1, g_dint1);
```

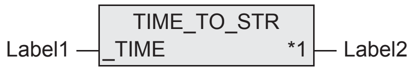
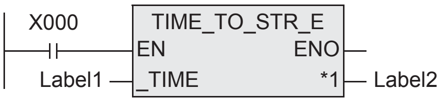
5.46 TIME_TO_STR(_E) / Time data → string data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIME_TO_STR		TIME_TO_STR(_TIME); Example: Label2:= TIME_TO_STR(Label1);
TIME_TO_STR_E		TIME_TO_STR_E(EN,_TIME, Output_label); Example: TIME_TO_STR_E(X000, Label1, Label2);

*1. Output variable

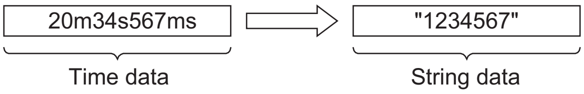
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

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 string 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.

Error

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

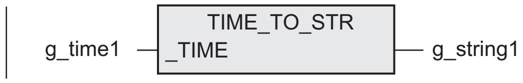
1) When the number of points occupied by the device specified in d exceeds the range of the corresponding device.

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]

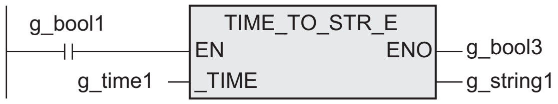


[ST]

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

2) Function with EN/ENO(TIME_TO_STR_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := TIME_TO_STR_E(g_bool1, g_time1, g_string1);
```

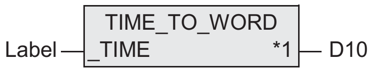
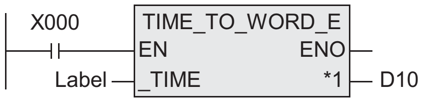
5.47 TIME_TO_WORD(_E) / Time data → word [unsigned]/bit string [16-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIME_TO_WORD		TIME_TO_WORD(_TIME); Example: D10:= TIME_TO_WORD(Label);
TIME_TO_WORD_E		TIME_TO_WORD_E(EN,_TIME, Output_label); Example: TIME_TO_WORD_E(X000, Label, D10);

*1. Output variable

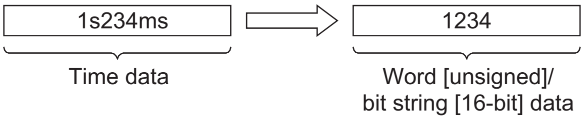
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

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 word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

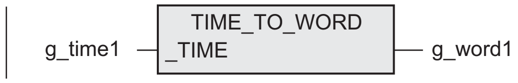
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, time data stored in a device specified in 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(TIME_TO_WORD)

[Structured ladder/FBD]

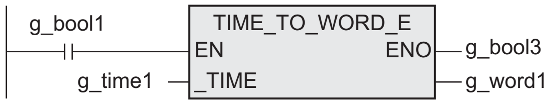


[ST]

```
g_word1 := TIME_TO_WORD(g_time1);
```

2) Function with EN/ENO(TIME_TO_WORD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := TIME_TO_WORD_E(g_bool1, g_time1, g_word1);
```

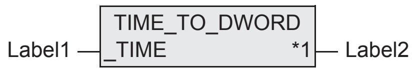
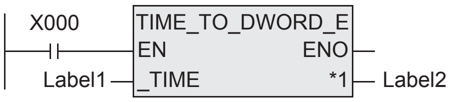
5.48 TIME_TO_DWORD(_E) / Time data → double word [unsigned]/ bit string [32-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

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

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIME_TO_DWORD		TIME_TO_DWORD(_TIME); Example: Label2:= TIME_TO_DWORD(Label1);
TIME_TO_DWORD_E		TIME_TO_DWORD_E(EN,_TIME, Output_label); Example: TIME_TO_DWORD_E(X000, Label1, Label2);

*1. Output variable

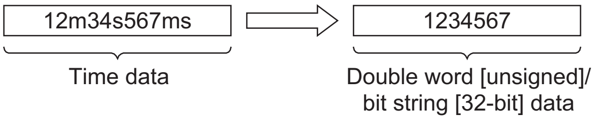
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

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

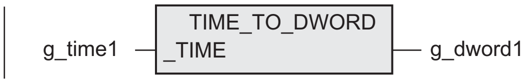
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

Program example

In this program, time data stored in a device specified in 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(TIME_TO_DWORD)

[Structured ladder/FBD]

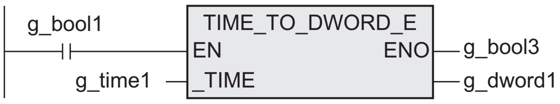


[ST]

```
g_dword1 := TIME_TO_DWORD(g_time1);
```

2) Function with EN/ENO(TIME_TO_DWORD_E)

[Structured ladder/FBD]



[ST]

```
g_dword1 := TIME_TO_DWORD(g_time1);
```

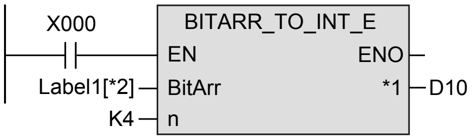
5.49 BITARR_TO_INT(_E) / Bit array → Word [signed] type, word [unsigned]/ bit String [16-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
BITARR_TO_INT		BITARR_TO_INT(BitArr, n); Example: D10:= BITARR_TO_INT Label1[*2], K4);
BITARR_TO_INT_E		BITARR_TO_INT_E(EN, BitArr, n, Output_label); Example: BITARR_TO_INT_E(X000, Label1[*2], K4, D10);

*1. Output variable

*2. Specify an array element.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	BitArr (s)	Start bit of conversion source bit array elements	Bit
	n	Number of specified bits	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data or word [unsigned]/bit [16-bit] data after conversion	ANY16

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

Explanation of function and operation

This function converts number of bits specified in n starting from a bit array element stored in a device specified in s into word [signed] data or word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in d.
Only a constant 4, 8, 12 or 16 can be specified in n.
"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]

[ST]

```
g_int1 := BITARR_TO_INT(g_bool4[0], K8);
```

2) Function with EN/ENO(BITARR_TO_INT_E)

[Structured ladder/FBD]

[ST]

```
g_bool2 := BITARR_TO_INT_E(g_bool1, g_bool4[0], K8, g_int1);
```

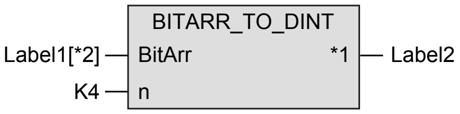
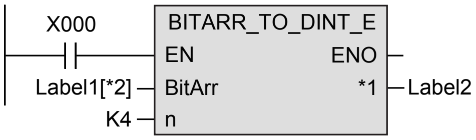
5.50 BITARR_TO_DINT(_E) / Bit array → Double word [signed] type, double word [unsigned]/bit string [32-bit] data conversion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
BITARR_TO_DINT		BITARR_TO_DINT(BitArr, n); Example: Label2:=BITARR_TO_DINT Label1[*2], K4);
BITARR_TO_DINT_E		BITARR_TO_DINT_E(EN, BitArr, n, Output_label); Example: BITARR_TO_DINT_E(X000, Label1[*2], K4, Label2);

*1. Output variable

*2. Specify an array element.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	BitArr (s)	Start bit of conversion source bit array elements	Bit
	n	Number of specified bits	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data or double word [unsigned]/bit string [32-bit] data after conversion	ANY32

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

Explanation of function and operation

This function converts number of bits specified in n starting from a bit array element stored in a device specified in s 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 d.
Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified in n.
"0" is set to output bits beyond the specified number of bits.

Cautions

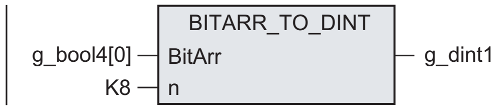
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data.
You can specify 32-bit counters directly,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



[ST]

```
g_dint1 := BITARR_TO_DINT(g_bool4[0], K8);
```

2) Function with EN/ENO(BITARR_TO_DINT_E)

[Structured ladder/FBD]

[ST]

```
g_bool2 := 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

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
INT_TO_BITARR		INT_TO_BITARR(s, n); Example: Label2[*1]:= INT_TO_BITARR(D10, K4);
INT_TO_BITARR_E		INT_TO_BITARR_E (EN, s, n, BitArr); Example: INT_TO_BITARR_E(X000, D10, K4, Label2[*1]);

*1. Specify an array element.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	s	Conversion source word [signed] data or word [unsigned]/bit string [16-bit] data	ANY16
	n	Number of specified bits	Word [signed]
Output variable	ENO	Execution status	Bit
	BitArr (d)	Start bit of bit array elements after conversion	Bit

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 s to a device specified in d.

Only a constant 4, 8, 12 or 16 can be specified in n.

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 s are output to a device specified in d.

1) Function without EN/ENO(INT_TO_BITARR)

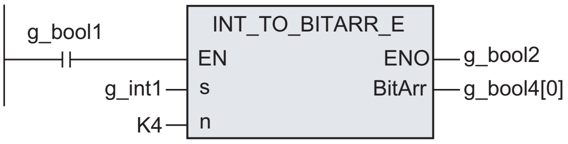
[Structured ladder/FBD]

[ST]

```
g_bool4[0] := INT_TO_BITARR(g_int1, K4);
```

2) Function with EN/ENO(INT_TO_BITARR_E)

[Structured ladder/FBD]



[ST]

```
g_bool2 := INT_TO_BITARR_E (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

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
DINT_TO_BITARR		DINT_TO_BITARR(s, n); Example: Label2[*1]:= DINT_TO_BITARR (Label1, K4)
DINT_TO_BITARR_E		DINT_TO_BITARR_E(EN, s, n, BitArr); Example: DINT_TO_BITARR_E(X000, Label1, K4, Label2[*1]);

*1. Specify an array element.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	s	Conversion source double word [signed] data or double word [unsigned]/bit string [32-bit] data	ANY32
	n	Number of specified bits	Word [signed]
Output variable	ENO	Execution status	Bit
	BitArr (d)	Start bit of bit array elements after conversion	Bit

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 s to a device specified in d.
Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified in n.
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 s are output to a device specified in d.

1) Function without EN/ENO(DINT_TO_BITARR)

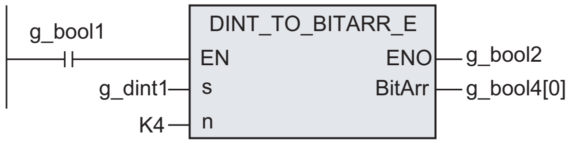
[Structured ladder/FBD]

[ST]

```
g_bool4[0] := DINT_TO_BITARR(g_dint1, K4);
```

2) Function with EN/ENO(INT_TO_BITARR_E)

[Structured ladder/FBD]



[ST]

```
g_bool2 := DINT_TO_BITARR_E (g_bool1, g_dint1, K4, g_bool4[0]);
```

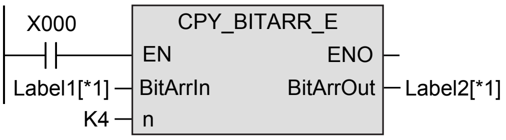
5.53 CPY_BITARR(_E) / Bit array copy

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function copies specified number of bits of a bit array.

1. Format

<table>
<tr><th rowspan="2">Function name</th><th colspan="2">Expression in each language</th></tr>
<tr><th>Structured ladder/FBD</th><th>ST</th></tr>
<tr><td>CPY_BITARR</td><td> </td><td>CPY_BITARR(BitArrIn, n);
Example:
Label2[*1]:= CPY_BITARR (Label1[*1], K4);</td></tr>
<tr><td>CPY_BITARR_E</td><td> </td><td>CPY_BITARR_E(EN, BitArrIn, n, BitArrOut);
Example:
CPY_BITARR_E(X000, Label1[*1], K4, Label2[*1]);</td></tr>
</table>

*1. Specify an array element.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	BitArrIn (s)	Start bit of copy source bit array elements	Bit
	n	Number of specified bits	Word [signed]
Output variable	ENO	Execution status	Bit
	BitArrOut (d)	Start bit of copy destination bit array elements	Bit

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 s to a device specified in d.
Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified in n.

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 s are output to the "num2"th element and later of a bit array stored in a device specified in d.

1) Function without EN/ENO(CPY_BITARR)

[Structured ladder/FBD]

[ST]

```
g_bool5[num2] := CPY_BITARR(g_bool4[num1], K12);
```

2) Function with EN/ENO(CPY_BITARR_E)

[Structured ladder/FBD]

[ST]

```
g_bool2 := CPY_BITARR_E(g_bool1, g_bool4[num1], K12, g_bool5[num2]);
```

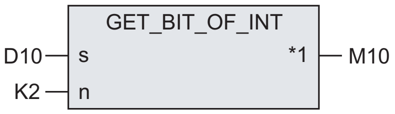
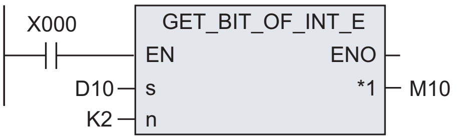
5.54 GET_BIT_OF_INT(_E) / Specified bit read of word [signed] data

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function reads a value of a specified bit of word [signed] data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
GET_BIT_OF_INT		GET_BIT_OF_INT(s, n); Example: M10:= GET_BIT_OF_INT (D10, K2);
GET_BIT_OF_INT_E		GET_BIT_OF_INT_E(EN, s, n, *1); Example: GET_BIT_OF_INT_E(X000, D10, K2, M10);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	s	Read source word [signed] data	Word [signed]
	n	Specified bit position	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Read destination bit	Bit

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

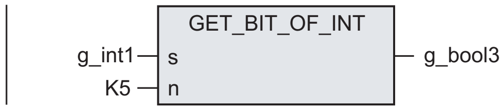
1) Use the function having "_E" in its name to connect a bus.

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]

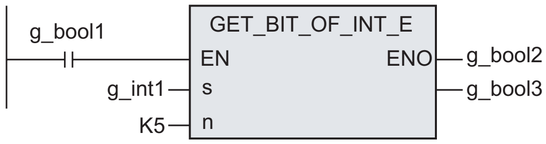


[ST]

```
g_bool3 := GET_BIT_OF_INT(g_int1, K5);
```

2) Function with EN/ENO(GET_BIT_OF_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool2 := GET_BIT_OF_INT_E (g_bool1, g_int1, K5, g_bool3);
```

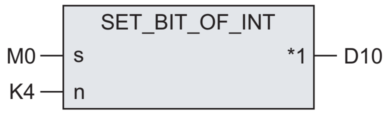
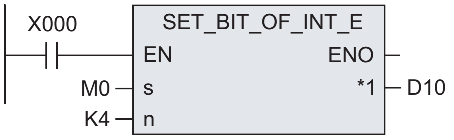
5.55 SET_BIT_OF_INT(_E) / Specified bit write of word [signed] data

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function writes a value to a specified bit of word [signed] data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
SET_BIT_OF_INT		SET_BIT_OF_INT(s, n); Example: D10:= SET_BIT_OF_INT (M0, K4);
SET_BIT_OF_INT_E		SET_BIT_OF_INT_E(EN, s, n, *1); Example: SET_BIT_OF_INT_E(X000, M0, K4, D10);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	s	Write source bit data	Bit
	n	Specified bit position	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Write destination word [signed] data	Word [signed]

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 s, and writes the read value to the "n"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 a device specified in s is written to the 3rd bit of a device specified in d.

1) Function without EN/ENO(SET_BIT_OF_INT)

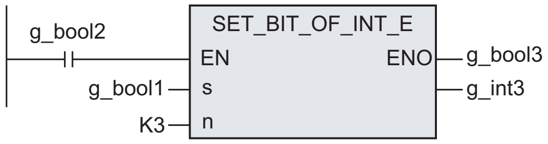
[Structured ladder/FBD]

[ST]

```
g_int3 := SET_BIT_OF_INT(g_bool1, K3);
```

2) Function with EN/ENO(SET_BIT_OF_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := SET_BIT_OF_INT_E(g_bool2, g_bool1, K3, g_int3);
```

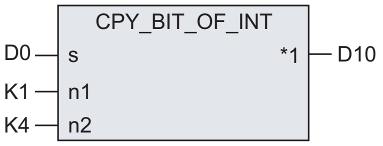
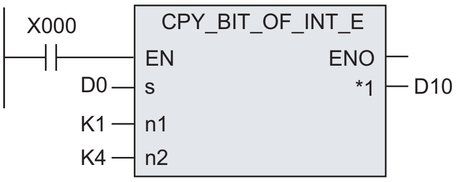
5.56 CPY_BIT_OF_INT(_E) / Specified bit copy of word [signed] data

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function copies a specified bit of word [signed] data to a specified bit of another word [signed] data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
CPY_BIT_OF_INT		CPY_BIT_OF_INT(s, n1, n2); Example: D10:= CPY_BIT_OF_INT (D0, K1, K4);
CPY_BIT_OF_INT_E		CPY_BIT_OF_INT_E(EN, s, n1, n2, *1); Example: CPY_BIT_OF_INT_E(X000, D0, K1, K4, D10);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	s	Copy source word [signed] data	Word [signed]
	n1	Specified bit position (copy source)	Word [signed]
	n2	Specified bit position (copy destination)	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Copy destination word [signed] data	Word [signed]

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

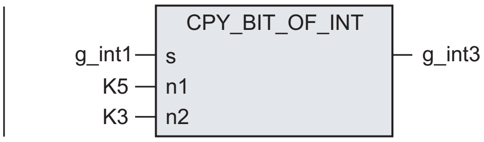
1) Use the function having "_E" in its name to connect a bus.

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]

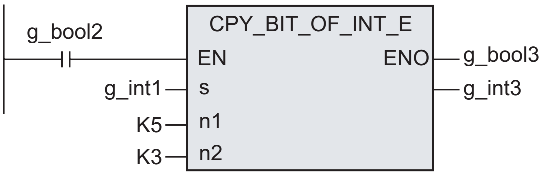


[ST]

```
g_int3 := CPY_BIT_OF_INT(g_int1, K5, K3);
```

2) Function with EN/ENO(CPY_BIT_OF_INT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := CPY_BIT_OF_INT_E(g_bool2, g_int1, K5, K3, g_int3);
```

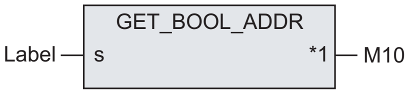
5.57 GET_BOOL_ADDR / Acquisition of start data

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function outputs the start data as bit data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
GET_BOOL_ADDR		GET_BOOL_ADDR(s); Example: M10:= GET_BOOL_ADDR (Label);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	s	Input data	ANY
Output variable	*1 (d)	Output data	Bit

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.

Input data type	Output data type
Bit, Bit string	Bit

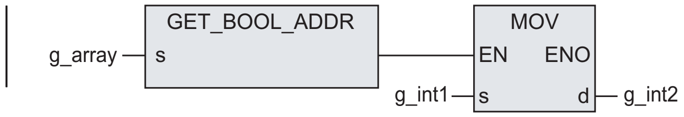
Cautions

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]



[ST]

```
MOV(GET_BOOL_ADDR(g_array), g_int1, g_int2);
```

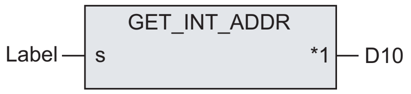
5.58 GET_INT_ADDR / Acquisition of start data

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function outputs the start data as word [signed] data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
GET_INT_ADDR		GET_INT_ADDR(s); Example: D10:= GET_INT_ADDR (Label);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	s	Input data	ANY
Output variable	*1 (d)	Output data	Word [signed]

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.

Input data type	Output data type
Word [signed], Double Word [signed], Word [unsigned]/Bit String [16-bit], FLOAT (Single Precision), String, Time, Array of Word [signed], Array of double word [signed], Array of word [unsigned]/bit string [16-bit], Array of double word [unsigned]/bit string [32-bit], Array of real number, Array of time type	Word [signed]

Cautions

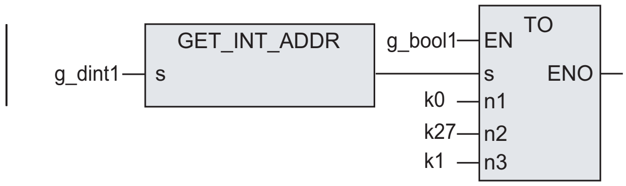
1) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when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



[ST]

```
TO(g_bool1, GET_INT_ADDR(g_dint1), K0, K27, K1);
```

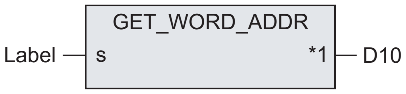
5.59 GET_WORD_ADDR / Acquisition of start data

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function outputs the start data as word [unsigned]/bit string [16-bit] data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
GET_WORD_ADDR		GET_WORD_ADDR(s); Example: D10:= GET_WORD_ADDR (Label);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	s	Input data	ANY
Output variable	*1 (d)	Output data	Word [unsigned]/ Bit String [16-bit]

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 [unsigned]/bit string [16-bit] data to a device specified in d.

Input data type	Output data type
Word [signed], Double Word [signed], Word [unsigned]/Bit String [16-bit], FLOAT (Single Precision), String, Time, Array of Word [signed], Array of double word [signed], Array of word [unsigned]/bit string [16-bit], Array of double word [unsigned]/bit string [32-bit], Array of real number, Array of time type	Word [unsigned]/ Bit String [16-bit]

Cautions

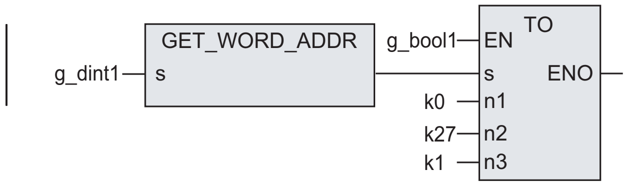
1) When handling 32-bit data and array data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data and array data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when 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 [unsigned]/bit string [16-bit] data to buffer memories.

[Structured ladder/FBD]



[ST]

```
TO(g_bool1, GET_WORD_ADDR(g_dint1), K0, K27, K1);
```

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

Function name	Function	Reference
ABS(_E)	Absolute value	Section 6.1

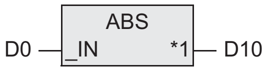
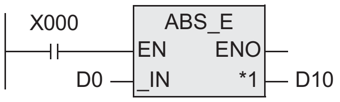
6.1 ABS(_E) / Absolute value

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function obtains the absolute value, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
ABS		ABS(_IN); Example: D10:= ABS(D0);
ABS_E		ABS_E(EN,_IN,Output_label); Example: ABS_E(X000,D0,D10);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Data whose absolute value is to be obtained, or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

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

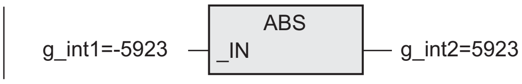
1) Use the function having "_E" in its name to connect a bus.
2) When the FLOAT (Single Precision) data to s is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this program, the absolute value is obtained for word [signed] data stored in a device specified in s, and the operation result is output to a device specified in d using the data type same as the data stored in a device specified in s.

1) Function without EN/ENO(ABS)

[Structured ladder/FBD]

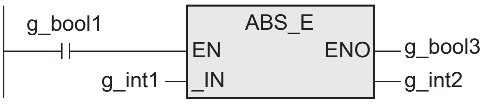


[ST]

```
g_int2 := ABS(g_int1);
```

2) Function with EN/ENO(ABS_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := ABS_E(g_bool1, g_int1, g_int2);
```

7. Applied Functions (Standard Arithmetic Functions)

Function name	Function	Reference
ADD_E	Addition	Section 7.1
SUB_E	Subtraction	Section 7.2
MUL_E	Multiplication	Section 7.3
DIV_E	Division	Section 7.4
MOD(_E)	Modulus operation	Section 7.5
EXPT(_E)	Exponentiation	Section 7.6
MOVE(_E)	Move operation	Section 7.7

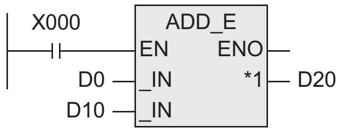
7.1 ADD_E / Addition

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs addition using two values (A + B = C), and outputs the operation result.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
ADD_E		ADD_E(EN,_IN,_IN, Output_label); Example: ADD_E(X000,D0,D10,D20);

*1. Output variable

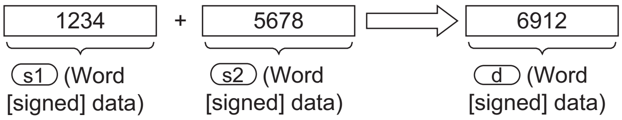
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Data for addition or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

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 d using the data type of data stored in devices specified in s1 to s28.
Example: When the data type is word [signed]



2) The number of pins for s can be changed in the range of 2 to 28.

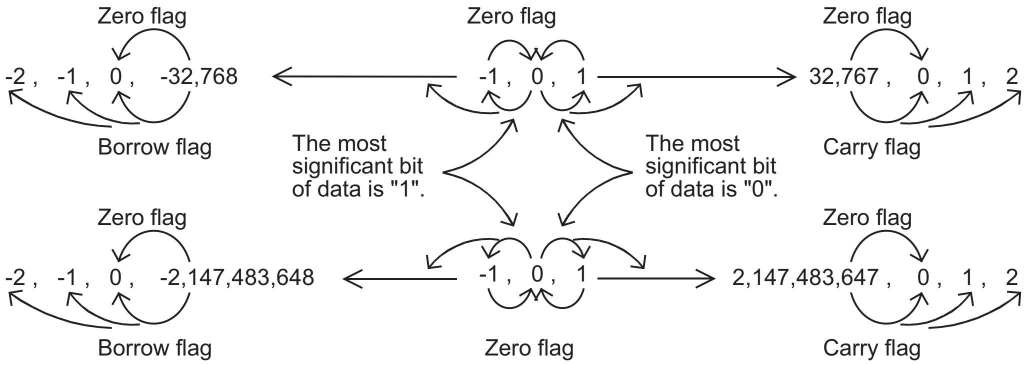
→ **Refer to Section 3. Function Construction**

Cautions

1) 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.

Device	Name	Description
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

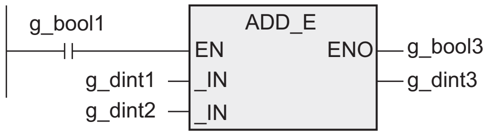


3) When the FLOAT (Single Precision) data to s is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this program, addition is performed using double word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool3:=ADD_E(g_bool1,g_dint1,g_dint2,g_dint3);
```

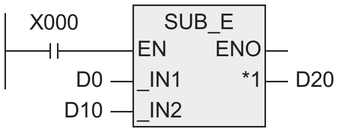
7.2 SUB_E / Subtraction

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs subtraction using two values (A - B = C), and outputs the operation result.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
SUB_E		SUB_E(EN,_IN1,_IN2, Output_label); Example: SUB_E(X000,D0,D10,D20);

*1. Output variable

2. Set data

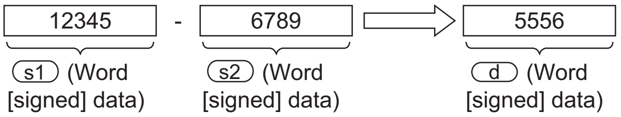
Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Data to be subtracted or word device which stores such data	ANY_NUM
	_IN2 (s2)	Data for subtraction or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

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

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]

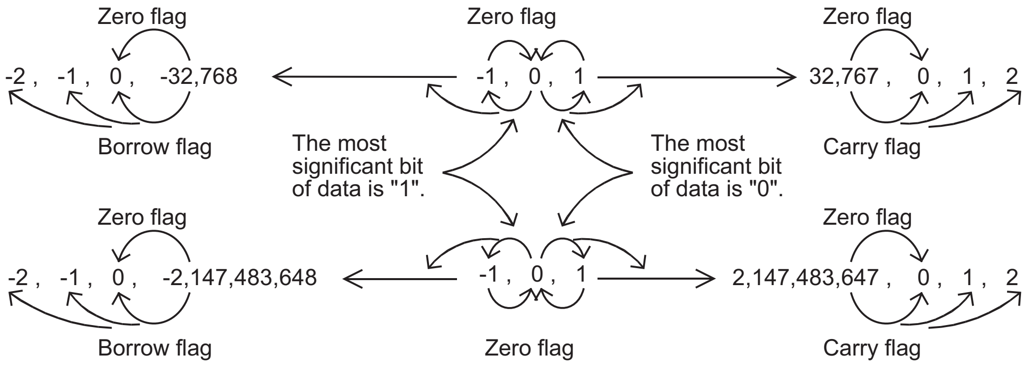


Cautions

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.

Device	Name	Description
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

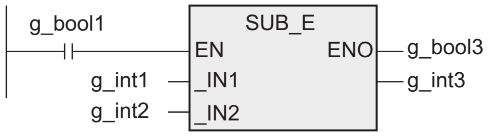


3) When the FLOAT (Single Precision) data to (s) is set from the programming tool, a rounding error may be generated.
   Refer to the MELSEC-Q/L/F Structured Programming Manual (Fundamentals) for cautions on setting the input value from the programming tool.

Program example

In this program, subtraction is performed using word [signed] data stored in devices specified in (s1) and (s2), the operation result is output to a device specified in (d).

[Structured ladder/FBD]



[ST]

```
g_bool3:=SUB_E(g_bool1,g_int1,g_int2,g_int3);
```

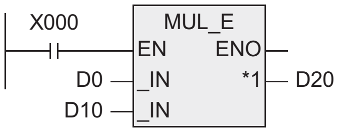
7.3 MUL_E / Multiplication

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs multiplication using two or more values (A × B = C), and outputs the operation result.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
MUL_E		MUL_E(EN,_IN,_IN, Output_label); Example: MUL_E(X000,D0,D10,D20);

*1. Output variable

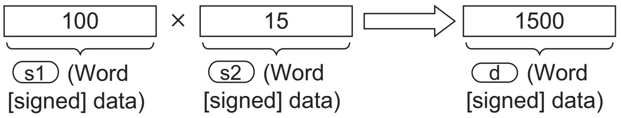
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Data for multiplication or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

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

Explanation of function and operation

1) This function performs multiplication (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 d using the data type of data stored in devices specified in s1 to s28.
Example: When the data type is word [signed]



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) 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.

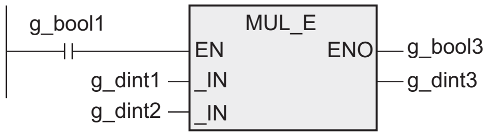
2) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. "TRUE" is output from ENO.
However, note that the obtained operation result is not accurate in this case.

3) When the FLOAT (Single Precision) data to 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, 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]



[ST]

```
g_bool3:=MUL_E(g_bool1,g_dint1,g_dint2,g_dint3);
```

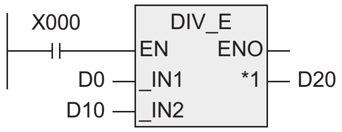
7.4 DIV_E / Division

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs division using two values (A / B = C … remainder), and outputs the quotient.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DIV_E		DIV_E(EN,_IN1,_IN2, Output_label); Example: DIV_E(X000,D0,D10,D20);

*1. Output variable

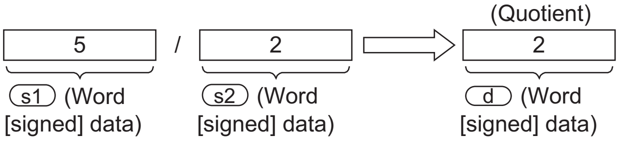
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Data to be divided, or word device which stores such data	ANY_NUM
	_IN2 (s2)	Data for division (divisor), or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

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]



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.

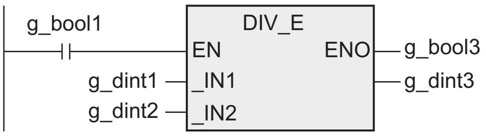
Error

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 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]



[ST]

```
g_bool3:=DIV_E(g_bool1,g_dint1,g_dint2,g_dint3);
```

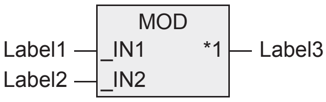
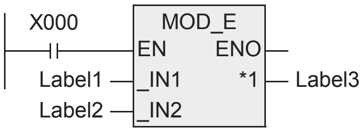
7.5 MOD(_E) / Modulus operation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs division using two values (A / B = C … remainder), and outputs the remainder.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	**ST**
MOD		_IN1 MOD _IN2; *2 Example: Label3:= Label1 MOD Label2;
MOD_E		MOD_E(EN,_IN1,_IN2, Output_label); *2 Example: MOD_E(X000,Label1, Label2,Label3);

*1. Output variable

*2. Refer to "Cautions".

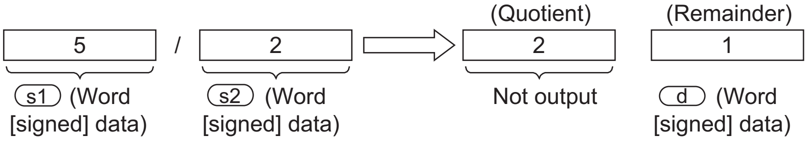
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Data to be divided, or word device which stores such data	ANY_INT
	_IN2 (s2)	Data for division (divisor), or word device which stores such data	ANY_INT
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_INT

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] data stored in devices specified in s1 and s2, and outputs the remainder 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]



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

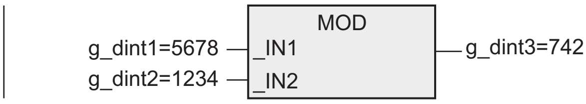
1) An operation error occurs when the divisor stored in a device specified in s2 is "0", and the function is not executed.
2) An operation error occurs when the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation).

Program example

In this program, division is performed using double word [signed] data stored in devices specified in s1 and s2, and the remainder 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(MOD)

[Structured ladder/FBD]

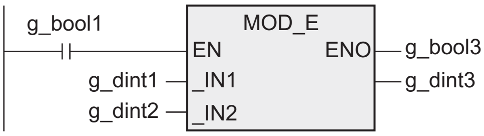


[ST]

```
g_dint3:=g_dint1 MOD g_dint2;
```

2) Function with EN/ENO(MOD_E)

[Structured ladder/FBD]



[ST]

```
g_bool3 := MOD_E(g_bool1, g_dint1, g_dint2, g_dint3);
```

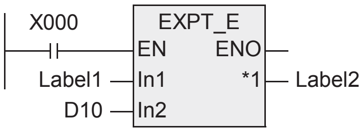
7.6 EXPT(_E) / Exponentiation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function obtains raised result, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
EXPT		EXPT(In1,In2); Example: Label2:= EXPT(Label1,D10);
EXPT_E		EXPT_E(EN,In1,In2, Output_label); Example: EXPT_E(X000,Label1,D10, Label2);

*1. Output variable

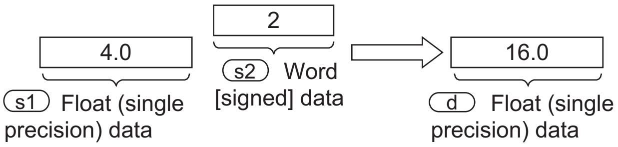
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	In1 (s1)	Data to be raised, or word device which stores such data	FLOAT (Single Precision)
	In2 (s2)	Power data, or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	FLOAT (Single Precision)

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 (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

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 s1 is negative
(Error code: K6706)
2) When the value stored in a device specified in 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 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.

1) Function without EN/ENO(EXPT)

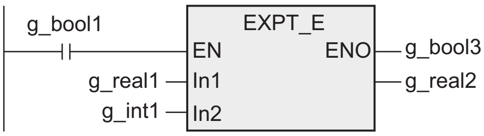
[Structured ladder/FBD]

[ST]

```
g_real2:=EXPT(g_real1,g_int1);
```

2) Function with EN/ENO(EXPT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=EXPT_E(g_bool1,g_real1,g_int1,g_real2);
```

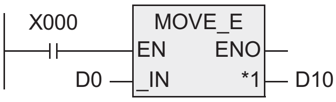
7.7 MOVE(_E) / Move operation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function transfers data stored in a device to another device.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
MOVE		MOVE(_IN); Example: D10:= MOVE(D0);
MOVE_E		MOVE_E(EN,_IN,Output_label); Example: MOVE_E(X000,D0,D10);

*1. Output variable

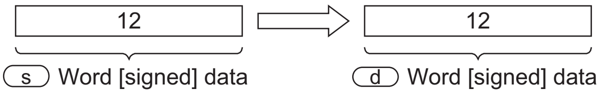
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Transfer source data, or word device which stores such data	ANY
Output variable	ENO	Execution status	Bit
	*1 (d)	Transfer destination word device	ANY

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 (s) 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 (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, word [signed] data stored in a device specified in s is transferred to a device specified in d.

1) Function without EN/ENO(MOVE)

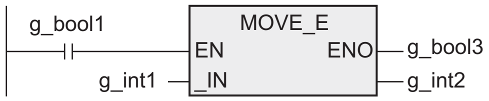
[Structured ladder/FBD]

[ST]

`g_int2:=MOVE(g_int1);`

2) Function with EN/ENO(MOVE_E)

[Structured ladder/FBD]



[ST]

`g_bool3:=MOVE_E(g_bool1,g_int1,g_int2);`

8. Applied Functions (Standard Bit Shift Functions)

Function name	Function	Reference
SHL(_E)	Left shift	Section 8.1
SHR(_E)	Right shift	Section 8.2

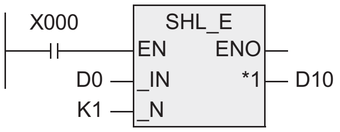
8.1 SHL(_E) / Left shift

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function shifts data of specified bit length leftward by the specified number of bits.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
SHL		SHL(_IN,_N); Example: D10:= SHL(D0,K1);
SHL_E		SHL_E(EN,_IN,_N,Output_label); Example: SHL_E(X000,D0,K1,D10);

*1. Output variable

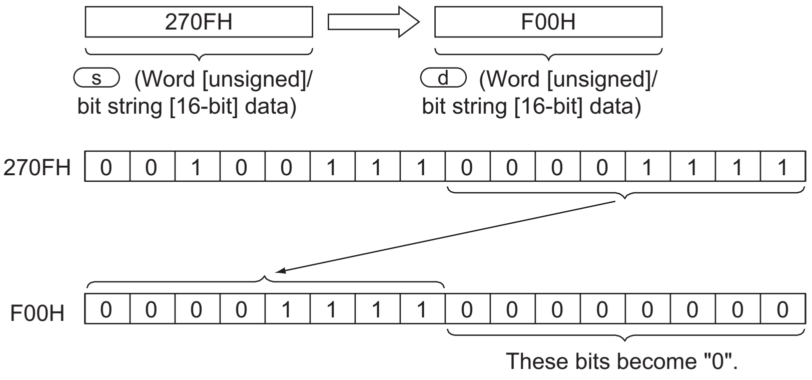
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Word device which stores data to be shifted leftward	ANY_BIT
	_N (n)	Number of shifted bits	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store data obtained by shift	ANY_BIT

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 leftward 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 leftward 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



270FH	0	0	1	0	0	1	1	1	0	0	0	0	1	1	1	1
F00H	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

2) "n" bits from the least significant bit become "0".

Cautions

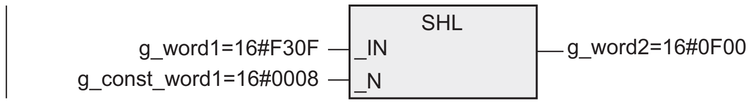
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit 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 s is shifted leftward by "n" bits, and the obtained data is output to a device specified in d using the data type of data stored in a device specified in s.

1) Function without EN/ENO(SHL)

[Structured ladder/FBD]

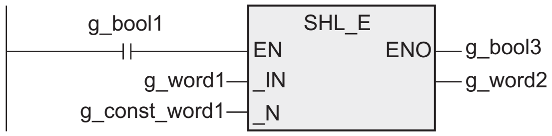


[ST]

```
g_word2:=SHL(g_word1,g_const_word1);
```

2) Function with EN/ENO(SHL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=SHL_E(g_bool1,g_word1,g_const_word1,g_word2);
```

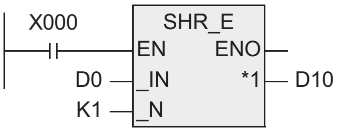
8.2 SHR(_E) / Right shift

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function shifts data of specified bit length rightward by the specified number of bits.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	**ST**
SHR		SHR(_IN,_K); Example: D10:= SHR(D0,K1);
SHR_E		SHR_E(EN,_IN,_N,Output_label); Example: SHR_E(X000,D0,K1,D10);

*1. Output variable

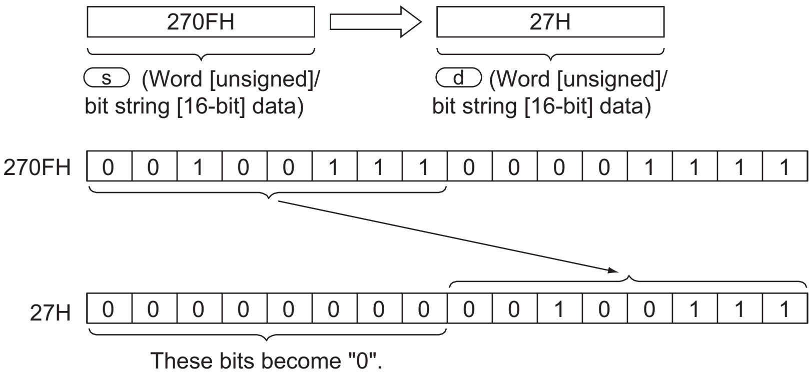
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN ((s))	Word device which stores data to be shifted rightward	ANY_BIT
	_K,_N ((n))	Number of shifted bits	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 ((d))	Word device which will store data obtained by shift	ANY_BIT

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

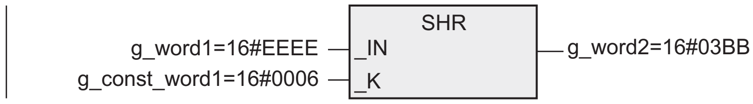
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit 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 s is shifted rightward by "n" bits, and the obtained data is output to a device specified in d using the data type of data stored in a device specified in s.

1) Function without EN/ENO(SHR)

[Structured ladder/FBD]

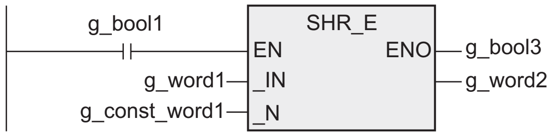


[ST]

```
g_word2:=SHR(g_word1,g_const_word1);
```

2) Function with EN/ENO(SHR_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=SHR_E(g_bool1,g_word1,g_const_word1,g_word2);
```

9. Applied Functions (Standard Bitwise Boolean Functions)

Function name	Function	Reference
AND_E	Logical product	Section 9.1
OR_E	Logical sum	Section 9.2
XOR_E	Exclusive logical sum	Section 9.3
NOT(_E)	Logical negation	Section 9.4

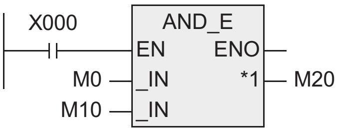
9.1 AND_E / Logical product

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function obtains the logical product of two or more bits, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
AND_E		AND_E(EN,_IN,_IN, Output_label); Example: AND_E(X000,M0,M10,M20);

*1. Output variable

2. Variable

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Device used to obtain the logical product	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the operation result	ANY_BIT

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	1	0	0	0	0	1	1	1	1

Logical product

s2	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0

⇩

d	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0

2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

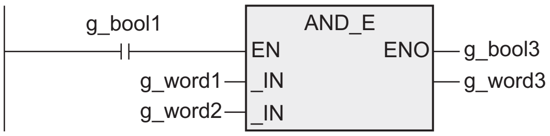
Cautions

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.

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]



[ST]

```
g_bool3:=AND_E(g_bool1,g_word1,g_word2,g_word3);
```

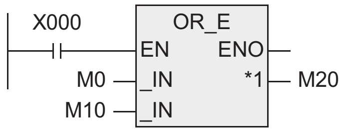
9.2 OR_E / Logical sum

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function obtains the logical sum of two or more bits, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	**ST**
OR_E		OR_E(EN,_IN,_IN,Output_label); Example: OR_E(X000,M0,M10,M20);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN ((s1) to (s28))	Device used to obtain the logical sum	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 ((d))	Device which will store the operation result	ANY_BIT

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 (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	0	0	1	1	1	1	0	0	0	0	1	1	1	1

Logical sum

(s2)	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0	0

⇩

(d)	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1

2) The number of pins for (s) can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

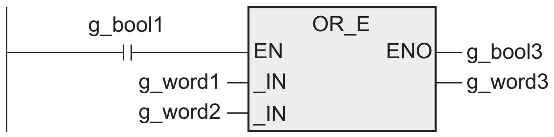
Cautions

1) 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 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]



[ST]

```
g_bool3:=OR_E(g_bool1,g_word1,g_word2,g_word3);
```

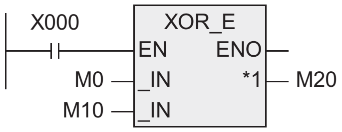
9.3 XOR_E / exclusive logical sum

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function obtains the exclusive logical sum of two or more bits, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
XOR_E		XOR_E(EN,_IN,_IN, Output_label); Example: XOR_E(X000,M0,M10,M20);

*1. Output variable

2. .Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Device used to obtain the exclusive logical sum	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the operation result	ANY_BIT

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 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	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Exclusive logical sum

s2	0	0	0	1	1	0	1	1	1	1	1	1	0	0	0	0

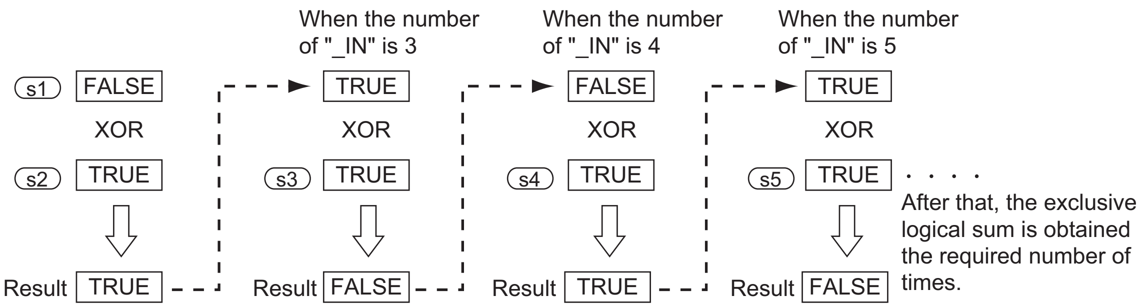
⇩

d	1	0	1	1	0	0	0	1	0	1	0	1	1	0	1	0

2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

3) If there are 3 or more s, the exclusive logical sum is obtained using the "exclusive logical sum of s1 and s2" and s3.
If there is s4, the exclusive logical sum is obtained using the "exclusive logical sum of "exclusive logical sum of s1 and s2" and "s3"" and s4. In this way, the exclusive logical sum is obtained the required number of times for all input labels s5 s6 …
Example: When the data type is bit



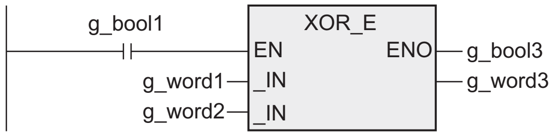
Cautions

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]/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]



[ST]

```
g_bool3:=XOR_E(g_bool1,g_word1,g_word2,g_word3);
```

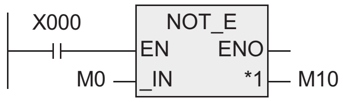
9.4 NOT(_E) / logical negation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function obtains the logical negation of bits, and outputs it.

1. Format

<table>
<tr><th rowspan="2">Function name</th><th colspan="2">Expression in each language</th></tr>
<tr><th>Structured ladder/FBD</th><th>ST</th></tr>
<tr><td>NOT</td><td> </td><td>NOT(_IN);
Example:
M10:=
NOT(M0);</td></tr>
<tr><td>NOT_E</td><td> </td><td>NOT_E(EN,_IN,Output_label);
Example:
NOT_E(X000,M0,M10);</td></tr>
</table>

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Device used to obtain the logical negation	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the operation result	ANY_BIT

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 (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]

s	0	1	1	0	1	0	1	1	0	0	0	0	1	1	1	1

Logical negation

d	1	0	0	1	0	1	0	0	1	1	1	1	0	0	0	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, 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.

1) Function without EN/ENO(NOT)

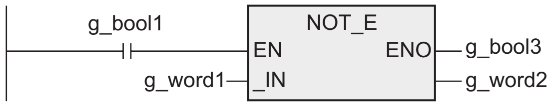
[Structured ladder/FBD]

[ST]

```
g_word2:= NOT(g_word1);
```

2) Function with EN/ENO(NOT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=NOT_E(g_bool1,g_word1,g_word2);
```

10. Applied Functions (Standard Selection Functions)

Function name	Function	Reference
SEL(_E)	Selection	Section 10.1
MAXIMUM(_E)	Maximum selection	Section 10.2
MINIMUM(_E)	Minimum selection	Section 10.3
LIMITATION(_E)	Upper/Lower limit control	Section 10.4
MUX(_E)	Multiplexer	Section 10.5

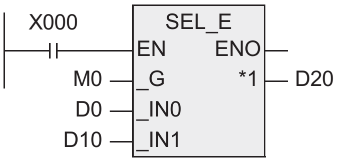
10.1 SEL(_E) / Selection

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function selects either one between two data in accordance with the input condition, and outputs the selection result.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
SEL		SEL(_G,_IN0,_IN1); Example: D20:= SEL(M0,D0,D10);
SEL_E		SEL_E(EN,_G,_IN0,_IN1, Output_label); Example: SEL_E(X000,M0,D0,D10,D20);

*1. Output variable

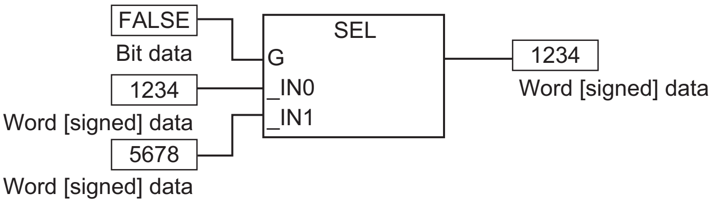
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_G (s1)	Bit data used as the selection condition	Bit
	_IN0 (s2)	Selectable data, or word device which stores such data	ANY
	_IN1 (s3)	Selectable data, or word device which stores such data	ANY
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the selection result	ANY

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 s2 and s3 in accordance with the value stored in a device specified in s1 to a device specified in d using the data type of data stored in a device specified in s2 and s3.
When the value stored in a device specified in s1 is "FALSE", this function outputs the value stored in a device specified in s2 to a device specified in d.
When the value stored in a device specified in s1 is "TRUE", this function outputs the value stored in a device specified in s3 to a device specified in d.
Example: When the data type of input variables s2 and s3 is word [signed]



Cautions

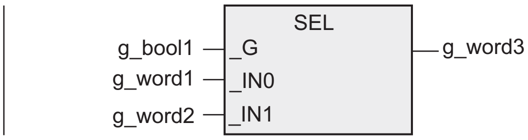
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) When the FLOAT (Single Precision) data to (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, either one between the values stored in devices specified in (s2) and (s3) is output in accordance with the value stored in a device specified in (s1) to a device specified in (d) using the data type of data stored in devices specified in (s2) and (s3).

1) Function without EN/ENO(SEL)

[Structured ladder/FBD]

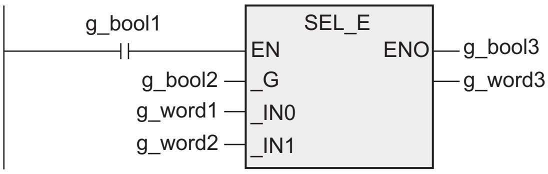


[ST]

```
g_word3:=SEL(g_bool1,g_word1,g_word2);
```

2) Function with EN/ENO(SEL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=SEL_E(g_bool1,g_bool2,g_word1,g_word2,g_word3);
```

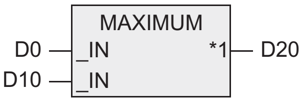
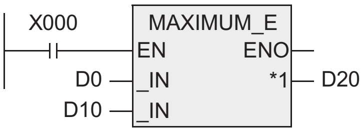
10.2 MAXIMUM(_E) / Maximum selection

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function searches the maximum value among data, and outputs the maximum value.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
MAXIMUM		MAXIMUM(_IN,_IN); Example: D20:= MAXIMUM(D0,D10);
MAXIMUM_E		MAXIMUM_E(EN,_IN,_IN, Output_label); Example: MAXIMUM_E(X000,D0,D10,D20);

*1. Output variable

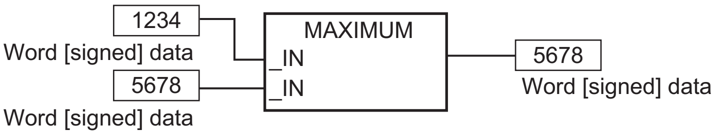
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the maximum value	ANY_SIMPLE

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 d using the data type of data stored in devices specified in s1 to s28.
   Example: When the data type is word [signed]



2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

Cautions

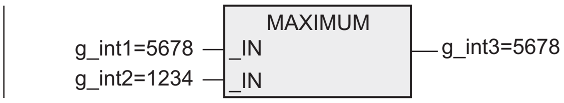
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit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

[Structured ladder/FBD]

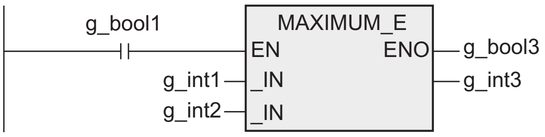


[ST]

```
g_int3:=MAXIMUM(g_int1,g_int2);
```

2) Function with EN/ENO(MAXIMUM_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=MAXIMUM_E(g_bool1,g_int1,g_int2,g_int3);
```

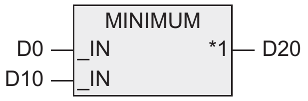
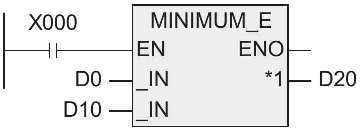
10.3 MINIMUM(_E) / Minimum selection

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function searches the minimum value among data, and outputs the minimum value.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
MINIMUM	 	MINIMUM(_IN,_IN); Example: D20:= MINIMUM(D0,D10);
MINIMUM_E	 	MINIMUM_E(EN,_IN,_IN, Output_label); Example: MINIMUM_E(X000,D0,D10,D20);

*1. Output variable

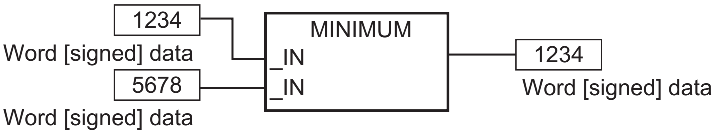
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the minimum value	ANY_SIMPLE

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 s1 to s28 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 [signed]



2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

Cautions

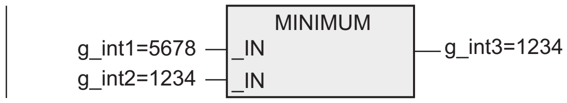
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit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

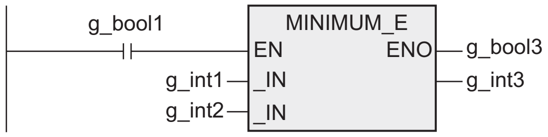


[ST]

```
g_int3:=MINIMUM(g_int1,g_int2);
```

2) Function with EN/ENO(MINIMUM_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=MINIMUM_E(g_bool1,g_int1,g_int2,g_int3);
```

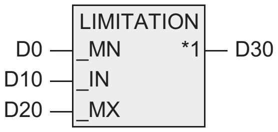
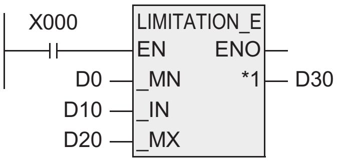
10.4 LIMITATION(_E) / Upper/Lower limit control

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function judges whether data is located within the range between the upper limit value and the lower limit value.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
LIMITATION		LIMITATION(_MN,_IN,_MX); Example: D30:= LIMITATION(D0,D10,D20);
LIMITATION_E		LIMITATION_E(EN,_MN,_IN,_MX, Output_label); Example: LIMITATION_E(X000,D0,D10,D20, D30);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_MN (s1)	Lower limit data, or word device which stores such data	ANY_SIMPLE
	_IN (s2)	Input data, or word device which stores such data	ANY_SIMPLE
	_MX (s3)	Upper limit data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the output data	ANY_SIMPLE

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

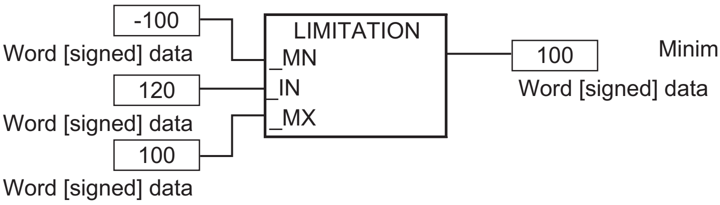
Explanation of function and operation

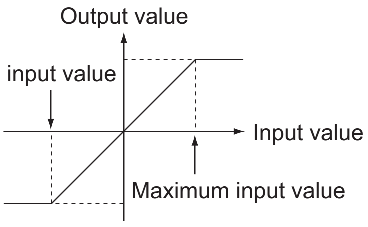
This function outputs data whose type is same as the data stored in devices specified in s1 , s2 and s3 to a device specified in d in accordance with ANY_SIMPLE type data stored in devices specified in s1 , s2 and s3 .

1) In the case of "Contents of a device specified in s2 > Contents of a device specified in s3 ", this function outputs the contents of a device specified in s3 to a device specified in d .

2) In the case of "Contents of a device specified in s2 < Contents of a device specified in s1 ", this function outputs the contents of a device specified in s1 to a device specified in d .

3) In the case of "Contents of a device specified in s1 ≤ Contents of a device specified in s2 ≤ Contents of a device specified in s3 ", this function outputs the contents of a device specified in s2 to a device specified in d .
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 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

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.

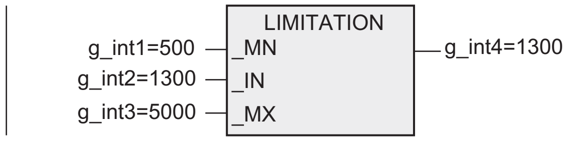
Contents of a device specified in s1 > Contents of a device specified in s3
(Lower limit data) (Upper limit data)

Program example

In this program, data whose type is same as the data stored in devices specified in s1, s2 and s3 is output to a device specified in d in accordance with ANY_SIMPLE type data stored in devices specified in s1, s2 and s3.

1) Function without EN/ENO(LIMITATION)

[Structured ladder/FBD]

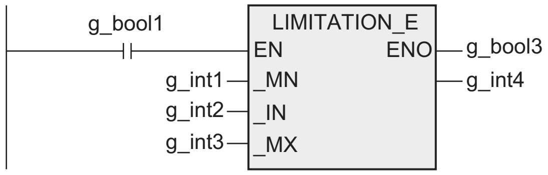


[ST]

```
g_int4:=LIMITATION(g_int1,g_int2,g_int3);
```

2) Function with EN/ENO(LIMITATION_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=LIMITATION_E(g_bool1,g_int1,g_int2,g_int3,g_int4);
```

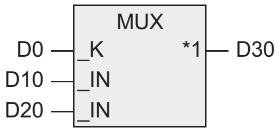
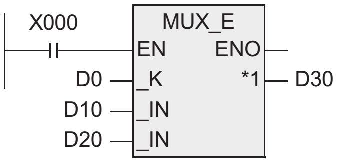
10.5 MUX(_E) / Multiplexer

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function selects data, and outputs the selected data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
MUX		MUX(_K,_IN,_IN); Example: D30:= MUX(D0,D10,D20);
MUX_E		MUX_E(EN,_K,_IN,_IN, Output_label); Example: MUX_E(X000,D0,D10,D20,D30);

*1. Output variable

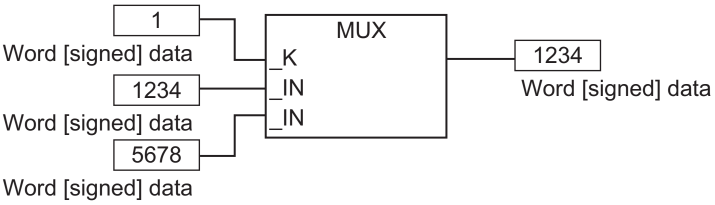
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_K (n)	Selection data, or word device which stores such data	Word [signed]
	_IN (s1 to s28)	Selectable data, or word device which stores such data	ANY
Output variable	ENO	Execution status	Bit
	*1	Word device which will store the selected data	ANY

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 s1 to s28 to a device specified in d in accordance with the value specified in n using the data type of data stored in devices specified in s1 to s28
   a) When the value specified in n is "1", this function outputs the value stored in a device specified in s1 to a device specified in d.
   b) When the value specified in n is "n", this function outputs the value stored in a device specified in sn to a device specified in d.
   Example: When the data type is word [signed]



2) When a value input to n is outside the pin number range for s1 to s28, this function outputs an indefinite value to a device specified in d.
   (An operation error does not occur. "MUX_E" outputs "FALSE" from ENO.)
3) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

Cautions

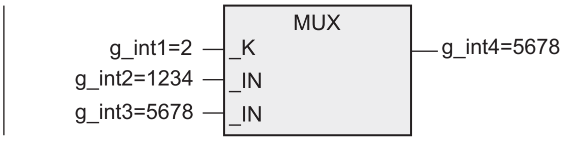
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit 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 example, either one among values stored in devices specified in s1 and s2 is output to a device specified in d in accordance with the value specified in n using the data type of data stored in devices specified in s1 and s2.

1) Function without EN/ENO(MUL)

[Structured ladder/FBD]

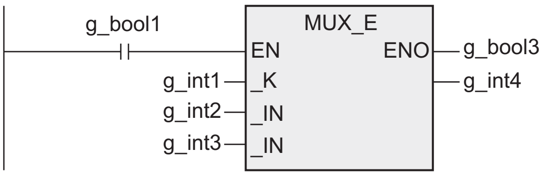


[ST]

```
g_int4:=MUX(g_int1,g_int2,g_int3);
```

2) Function with EN/ENO(MUL_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=MUX_E(g_bool1,g_int1,g_int2,g_int3,g_int4);
```

11. Applied Functions (Standard Comparison Functions)

Function name	Function	Reference
GT_E	Comparison	Section 11.1
GE_E	Comparison	Section 11.2
EQ_E	Comparison	Section 11.3
LE_E	Comparison	Section 11.4
LT_E	Comparison	Section 11.5
NE_E	Comparison	Section 11.6

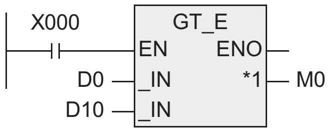
11.1 GT_E / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "> (larger)".

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	**ST**
GT_E		GT_E(EN,_IN,_IN,Output_label); Example: GT_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

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 d.
This function executes comparison [s1 > s2] & [s2 > s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

Cautions

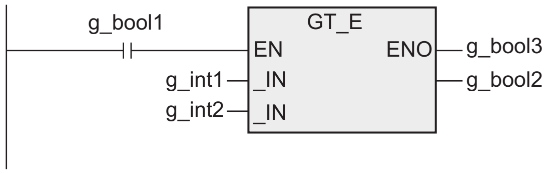
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects.
Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to 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 s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool3:=GT_E(g_bool1,g_int1,g_int2,g_bool2);
```

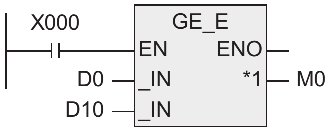
11.2 GE_E / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≥ (larger or equal)".

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
GE_E		GE_E(EN,_IN,_IN,Output_label); Example: GE_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

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 d.
   This function executes comparison [s1 ≥ s2] & [s2 ≥ s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ Refer to Section 3. Function Construction

Cautions

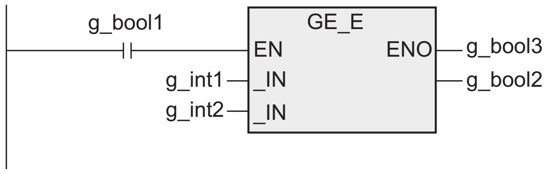
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to 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 (s1) and (s2) are compared, and the operation result is output to a device specified in (d).

[Structured ladder/FBD]



[ST]

```
g_bool3:=GE_E(g_bool1,g_int1,g_int2,g_bool2);
```

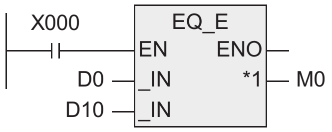
11.3 EQ_E / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "= (equal)".

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
EQ_E		EQ_E(EN,_IN,_IN,Output_label); Example: EQ_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

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 d.
This function executes comparison [s1 = s2] & [s2 = s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

Cautions

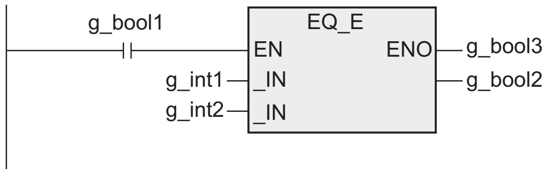
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to 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 s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool3:=EQ_E(g_bool1,g_int1,g_int2,g_bool2);
```

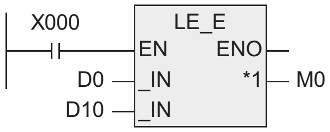
11.4 LE_E / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≤ (smaller or equal)".

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
LE_E		LE_E(EN,_IN,_IN,Output_label); Example: LE_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN ((s1) to (s28))	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 ((d))	Device which will store the comparison result	Bit

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 (d).
   This function executes comparison [(s1) ≤ (s2)] & [(s2) ≤ (s3)] & … & [(s n-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 (s) can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

Cautions

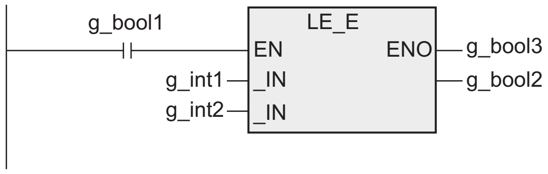
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to (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 s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool3:=LE_E(g_bool1,g_int1,g_int2,g_bool2);
```

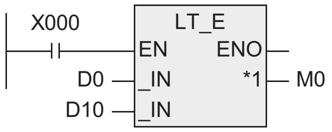
11.5 LT_E / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "< (smaller)".

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
LT_E		LT_E(EN,_IN,_IN,Output_label); Example: LT_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

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 d.
This function executes comparison [s1 < s2] & [s2 < s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

Cautions

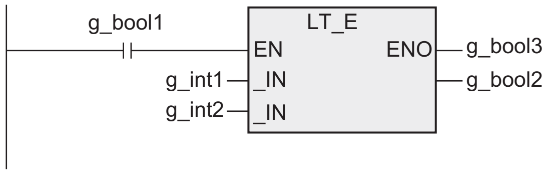
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

2) When the FLOAT (Single Precision) data to 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 s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool3:=LT_E(g_bool1,g_int1,g_int2,g_bool2);
```

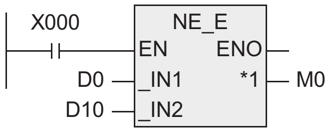
11.6 NE_E / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≠ (unequal)".

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
NE_E		NE_E(EN,_IN1,_IN2,Output_label); Example: NE_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Compared data, or word device which stores such data	ANY_SIMPLE
	_IN2 (s2)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

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 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

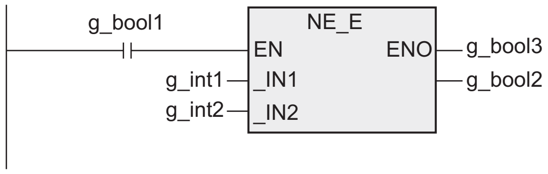
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

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 s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool3:=NE_E(g_bool1,g_int1,g_int2,g_bool2);
```

12. Applied Functions (Standard Character String Functions)

Function name	Function	Reference
MID(_E)	Extract mid string	Section 12.1
CONCAT(_E)	String concatenation	Section 12.2
INSERT(_E)	String insertion	Section 12.3
DELETE(_E)	String deletion	Section 12.4
REPLACE(_E)	String replacement	Section 12.5
FIND(_E)	Searches a character string	Section 12.6

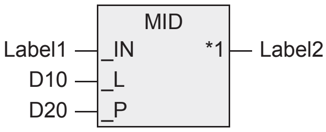
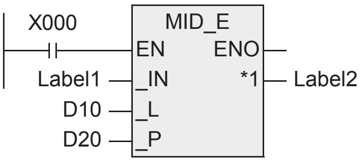
12.1 MID(_E) / Extract mid string

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function obtains a character string from a specified position.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	**ST**
MID		MID(_IN,_L ,_P); Example: Label2:= MID(Label1,D10,D20);
MID_E		MID_E(EN,_IN,_L ,_P, Output_label); Example: MID_E(X000,Label1,D10,D20, Label2);

*1. Output variable

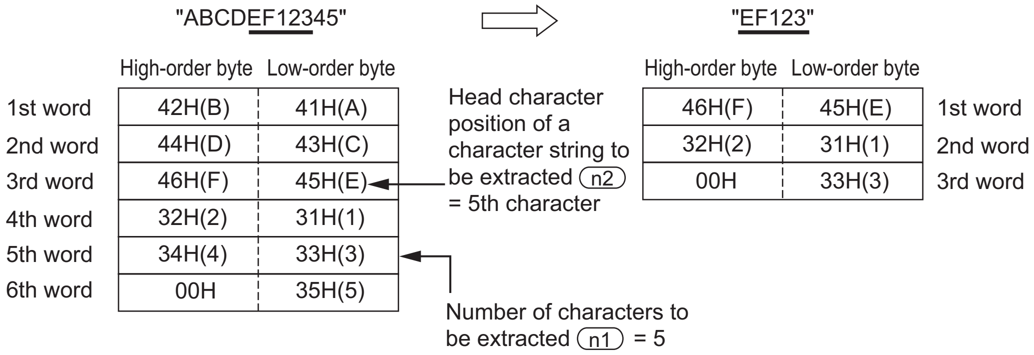
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Head word device which stores a character string	String
	_L (n1)	Word device which stores the number of characters to be obtained	Word [signed]
	_P (n2)	Word device which stores the head character position of a character string to be obtained	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the obtained character string	String

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

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 s, and outputs the obtained data to devices specified in d. The value specified in n1 specifies the number of characters to be extracted.
   The value specified in n2 specifies the head character position of characters to be extracted.
   Example: When "5" is specified in n1 and n2



"ABCDEF12345" ⇨ "EF123"

	High-order byte	Low-order byte
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)

Head character position of a character string to be extracted n2 = 5th character

Number of characters to be extracted n1 = 5

High-order byte	Low-order byte	
46H(F)	45H(E)	1st word
32H(2)	31H(1)	2nd word
00H	33H(3)	3rd word

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 n1 is "0", this function does not execute processing.

4) When the number of characters to be extracted specified in n1 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 n2 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 n1 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 n2 is negative
   (Error code: K6706)

6) When the value specified in n1 is "-2" or less
   (Error code: K6706)

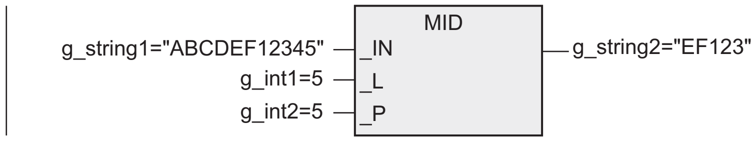
7) When the value specified in n1 exceeds the number of characters of a character string stored in devices specified in s
   (Error code: K6706)

Program example

In this program, specified number of characters are extracted from an arbitrary position of a character string stored in devices specified in s, and the obtained data is output to devices specified in d.

1) Function without EN/ENO(MID)

[Structured ladder/FBD]

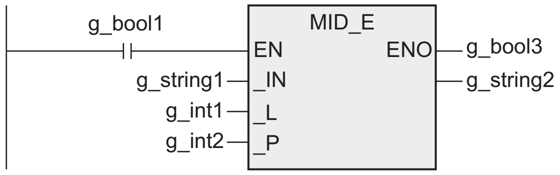


[ST]

```
g_string2:=MID(g_string1,g_int1,g_int2);
```

2) Function with EN/ENO(MID_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=MID_E(g_bool1,g_string1,g_int1,g_int2,g_string2);
```

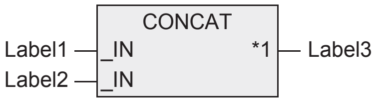
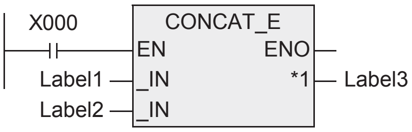
12.2 CONCAT(_E) / String concatenation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function connects character strings.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
CONCAT		CONCAT(_IN,_IN); Example: Label3:= CONCAT(Label1,Label2);
CONCAT_E		CONCAT_E(EN,_IN,_IN, Output_label); Example: CONCAT_E(X000,Label1,Label2, Label3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 to s28)	Head word device which stores the data (character string) to be connected, or directly specified character string	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the connected data (character string)	String

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 (s n+1) after a character string stored in devices specified in (sn), and outputs the character string obtained by connection to devices specified in (d).
When connecting a character string stored in devices specified in (s n+1), this function ignores "00H" which indicates the end of a character string stored in devices specified in (sn).
After two character strings are connected, "00H" is automatically added at the end.

"ABCDE" + "123456" ⇒ "ABCDE123456"

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	00H	45H(E)

	High-order byte	Low-order byte
1st word	32H(2)	31H(1)
2nd word	34H(4)	33H(3)
3rd word	36H(6)	35H(5)
4th word	0000H	

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	31H(1)	45H(E)
4th word	33H(3)	32H(2)
5th word	35H(5)	34H(4)
6th word	00H	36H(6)

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) For direct specification, up to 32 characters can be specified (input).
When word devices are specified in (s), this restriction (up to 32 characters) is not applicable.

4) When both a character string stored in devices specified in s begin with "00H" (when character = 0), this function stores "0000H" in devices specified in d.
5) 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 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.

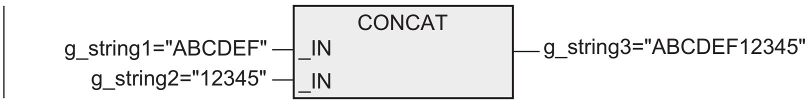
1) When the number of devices after the device number specified in d is smaller than the number of devices required for storing the character string obtained by connection
   (In this case, "00H" cannot be stored after all character strings and final character.)
   (Error code: K6706)
2) When devices which store character strings specified in s 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
   (Error code: K6706)

Program example

In this program, a character string stored in devices specified in s2 is connected after a character string stored in devices specified in s1, and the character string obtained by connection is output to devices specified in d.

1) Function without EN/ENO(CONCAT)

[Structured ladder/FBD]

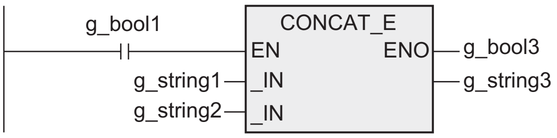


[ST]

```
g_string3:=CONCAT(g_string1,g_string2);
```

2) Function with EN/ENO(CONCAT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=CONCAT_E(g_bool1,g_string1,g_string2,g_string3);
```

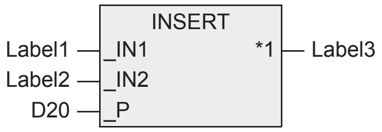
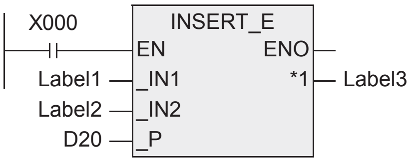
12.3 INSERT(_E) / String insertion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function inserts a character string.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
INSERT		INSERT(_IN1,_IN2,_P); Example: Label3:= INSERT(Label1,Label2,D20);
INSERT_E		INSERT_E(EN,_IN1,_IN2,_P, Output_label); Example: INSERT_E(X000,Label1,Label2, D20,Label3);

*1. Output variable

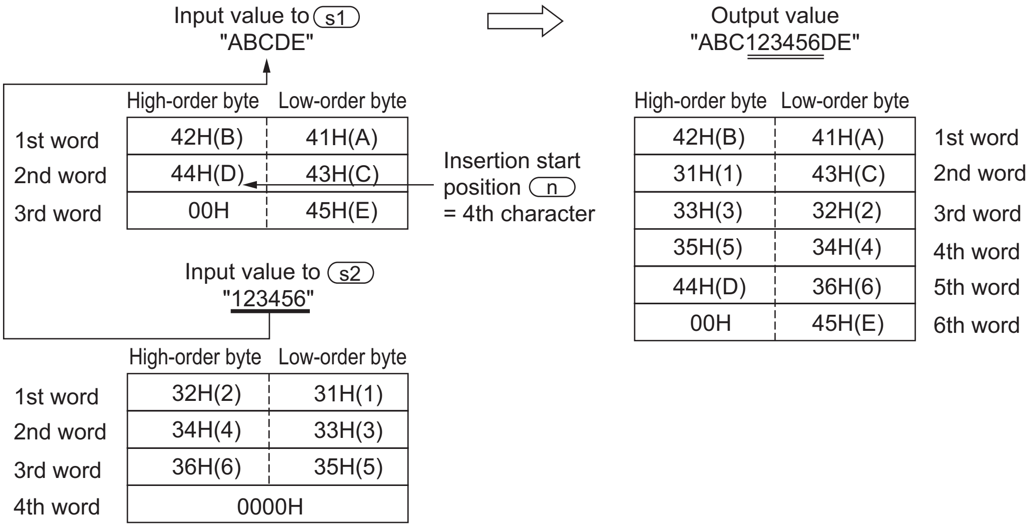
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores a character string to get insertion	String
	_IN2 (s2)	Head word device which stores a character string to be inserted	String
	_P (n)	Word device which stores a character position to get insertion	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store a character string obtained by insertion	String

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 s2 into an arbitrary position (counted from the head) of a character string stored in devices specified in s1, and outputs the character string obtained by insertion to devices specified in d.
   The value specified in n specifies the position from which the character string stored in devices specified in s2 is inserted.
   After inserting a character string stored in devices specified in s2 into a character string stored in devices specified in s1, this function ignores "00H" which indicates the end of a character string stored in devices specified in s2.
   Example: When "4" is specified in n



	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	00H	45H(E)

	High-order byte	Low-order byte
1st word	32H(2)	31H(1)
2nd word	34H(4)	33H(3)
3rd word	36H(6)	35H(5)
4th word	0000H	

High-order byte	Low-order byte	
42H(B)	41H(A)	1st word
31H(1)	43H(C)	2nd word
33H(3)	32H(2)	3rd word
35H(5)	34H(4)	4th word
44H(D)	36H(6)	5th word
00H	45H(E)	6th word

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.

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.

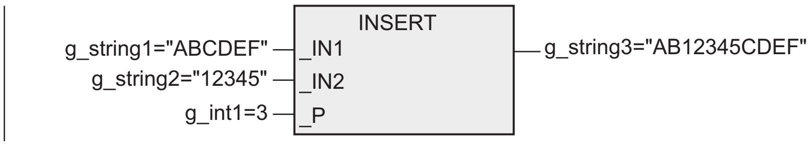
1) When the number of devices after the device number specified in d is smaller than the number of devices required for storing the output data obtained by insertion
   (Error code: K6706)
2) When devices which store character strings specified in s1 and s2 overlap device numbers specified in d which 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 s1 and s2
   (Error code: K6706)
4) When the number of characters of a character string stored in devices specified in s2 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 (s2) is inserted into an arbitrary position (counted from the head) of a character string stored in devices specified in (s1), and the character string obtained by insertion is output to devices specified in (d).

1) Function without EN/ENO(INSERT)

[Structured ladder/FBD]

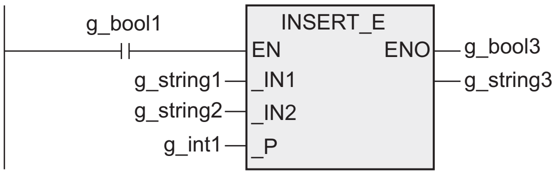


[ST]

```
g_string3:=INSERT(g_string1,g_string2,g_int1);
```

2) Function with EN/ENO(INSERT_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=INSERT_E(g_bool1,g_string1,g_string2,g_int1,g_string3);
```

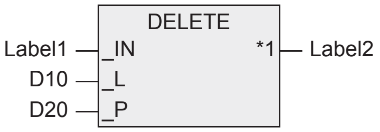
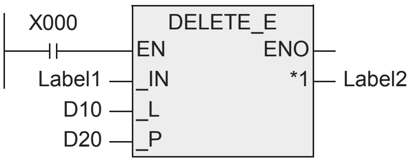
12.4 DELETE(_E) / String deletion

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function deletes a character string.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DELETE		DELETE(_IN,_L ,_P); Example: Label2:= DELETE(Label1,D10,D20);
DELETE_E		DELETE_E(EN,_IN,_L ,_P, Output_label); Example: DELETE_E(X000, Label1, D10, D20, Label2);

*1. Output variable

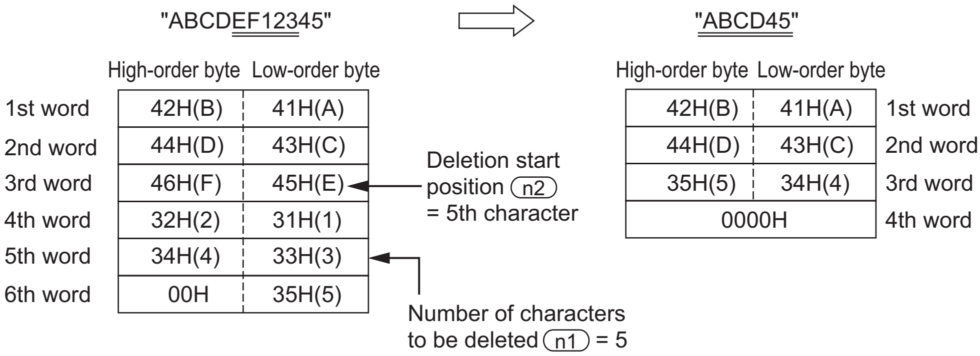
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Head word device which stores a character string to get deletion	String
	_L (n1)	Number of characters to be deleted	Word [signed]
	_P (n2)	Head position to get deletion	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store a character string remaining after deletion	String

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 s, and outputs the character string remaining after deletion to devices specified in d.
The value specified in n1 specifies the number of characters to be deleted.
The value specified in n2 specifies the position from which specified number of characters are deleted.
Example: When "5" is specified in n1 and n2



"ABCDEF12345" ⇒ "ABCD45"

	High-order byte	Low-order byte
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 n2 = 5th character

Number of characters to be deleted n1 = 5

High-order byte	Low-order byte	
42H(B)	41H(A)	1st word
44H(D)	43H(C)	2nd word
35H(5)	34H(4)	3rd word
0000H		4th word

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.

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.

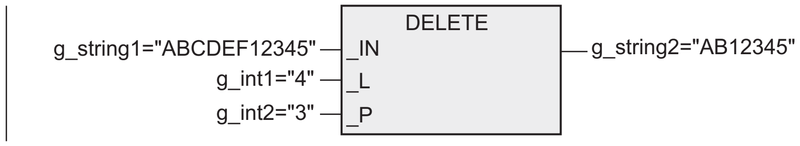
1) When "00H" does not exist in the corresponding device range after the device specified in s
   (Error code: K6706)
2) When the number of characters of a character string stored in devices specified in s is 32768 or more
   (Error code: K6706)
3) When the number of devices after the device number specified in d 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 n2 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 s, and the character string remaining after deletion is output to devices specified in d.

1) Function without EN/ENO(DELETE)

[Structured ladder/FBD]

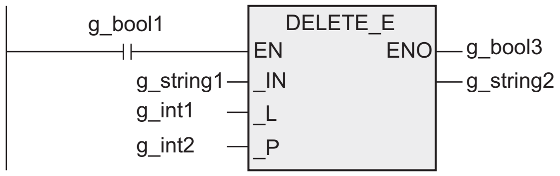


[ST]

```
g_string2:=DELETE(g_string1,g_int1,g_int2);
```

2) Function with EN/ENO(DELETE_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=DELETE_E(g_bool1,g_string1,g_int1,g_int2,g_string2);
```

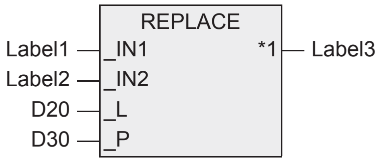
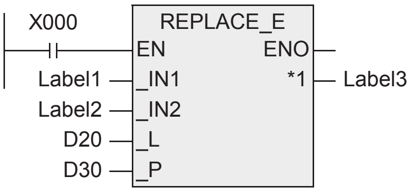
12.5 REPLACE(_E) / String replacement

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function replaces a character string.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
REPLACE		REPLACE(_IN1,_IN2,_L ,_P); Example: Label3:= REPLACE(Label1,Label2, D20,D30);
REPLACE_E		REPLACE_E(EN,_IN1,_IN2, _L ,_P,Output_label); Example: REPLACE_E(X000,Label1, Label2,D20,D30,Label3);

*1. Output variable

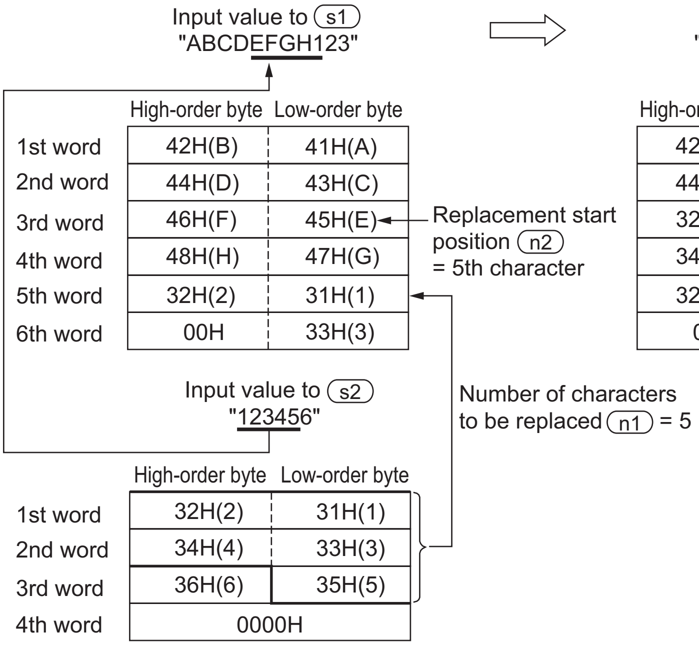
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores a character string to be replaced	String
	_IN2 (s2)	Head word device which stores a replacement character string	String
	_L (n1)	Word device which stores the number of characters to be replaced	Word [signed]
	_P (n2)	Word device which stores the head character position to be replaced in a character string to be replaced	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store a character string obtained by replacement	String

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



Input value to s1:

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	46H(F)	45H(E)
4th word	48H(H)	47H(G)
5th word	32H(2)	31H(1)
6th word	00H	33H(3)

Output value:

High-order byte	Low-order byte	
42H(B)	41H(A)	1st word
44H(D)	43H(C)	2nd word
32H(2)	31H(1)	3rd word
34H(4)	33H(3)	4th word
32H(2)	35H(5)	5th word
00H	33H(3)	6th word

Input value to s2:

	High-order byte	Low-order byte
1st word	32H(2)	31H(1)
2nd word	34H(4)	33H(3)
3rd word	36H(6)	35H(5)
4th word	0000H	

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 "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 d.

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.

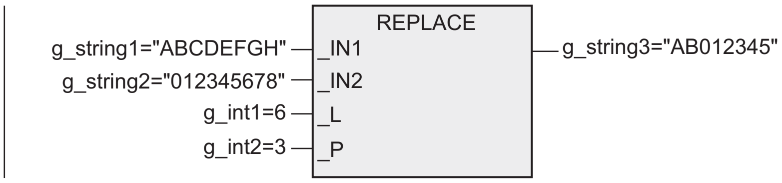
1) When "00H" does not exist in the corresponding device range after the devices specified in s1 and s2
   (Error code: K6706)
2) When the value specified in n1 exceeds the number of characters of a character string stored in devices specified in s2
   (Error code: K6706)
3) When the value specified 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]

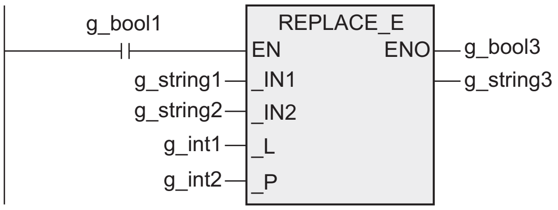


[ST]

```
g_string3:=REPLACE(g_string1,g_string2,g_int1,g_int2);
```

2) Function with EN/ENO(REPLACE_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=REPLACE_E(g_bool1,g_string1,g_string2,g_int1,g_int2,g_string3);
```

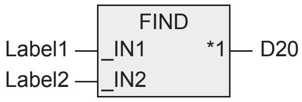
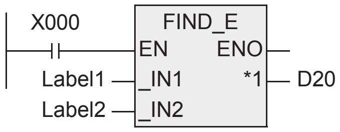
12.6 FIND(_E) / Searches a character string

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This function searches a character string.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
FIND		FIND(_IN1,_IN2); Example: D20:= FIND(Label1,Label2);
FIND_E		FIND_E(EN,_IN1,_IN2, Output_label); Example: FIND_E(X000,Label1,Label2, D20);

*1. Output variable

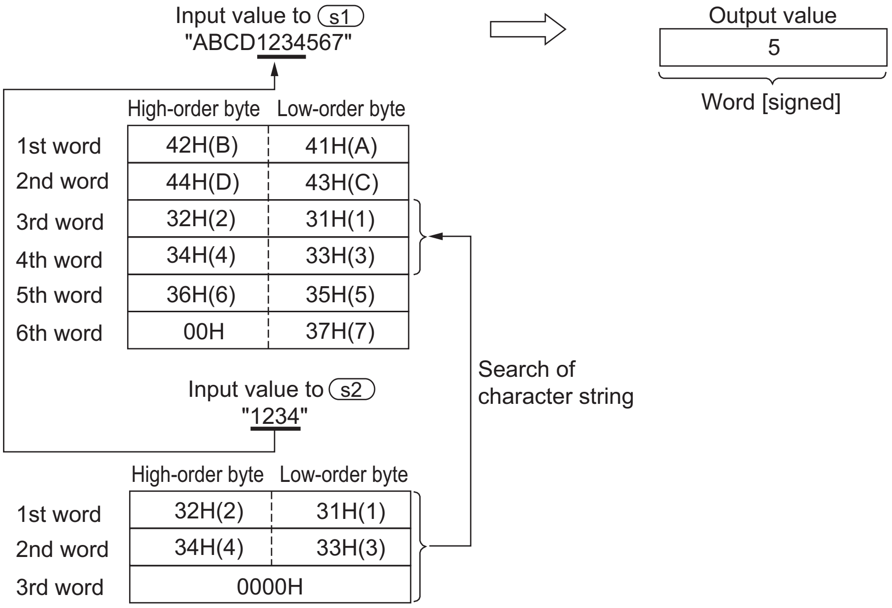
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores a character string to get search	String
	_IN2 (s2)	Head word device which stores a character string to be searched	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the search result	Word [signed]

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 s2 from the beginning of a character string stored in devices specified in s1, 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 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 s2 cannot be detected in a character string stored in devices specified in s1, 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.

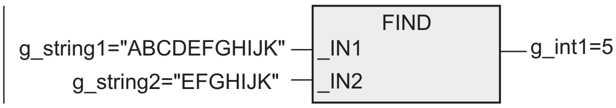
1) When "00H (NULL)" does not exist in the corresponding device range specified in s1
   (Error code: K6706)
2) When "00H (NULL)" does not exist in the corresponding device range specified in s2
   (Error code: K6706)

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]

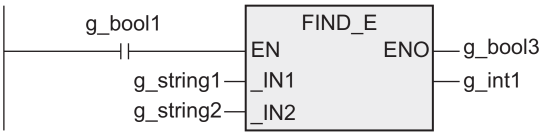


[ST]

```
g_int1:=FIND(g_string1,g_string2);
```

2) Function with EN/ENO(FIND_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=FIND_E(g_bool1,g_string1,g_string2,g_int1);
```

13. Applied Functions (Functions Of Time Data Types)

Function name	Function	Reference
ADD_TIME(_E)	Addition	Section 13.1
SUB_TIME(_E)	Subtraction	Section 13.2
MUL_TIME(_E)	Multiplication	Section 13.3
DIV_TIME(_E)	Division	Section 13.4

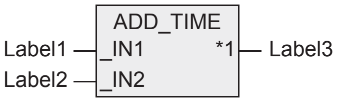
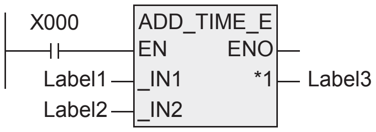
13.1 ADD_TIME(_E) / Addition

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function adds time data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
ADD_TIME		ADD_TIME(_IN1,_IN2); Example: Label3:= ADD_TIME(Label1,Label2);
ADD_TIME_E		ADD_TIME_E(EN,_IN1,_IN2, Output_label); Example: ADD_TIME_E(X000,Label1, Label2,Label3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get addition	Time
	_IN2 (s2)	Head word device which stores addition time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

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

Explanation of function and operation

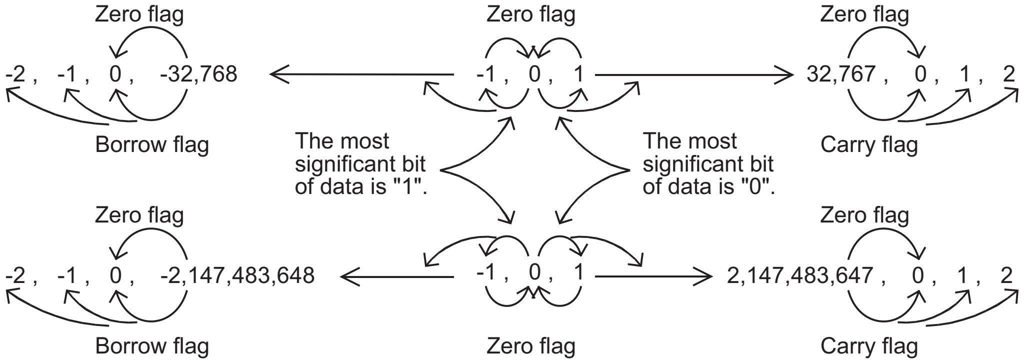
This function performs addition (s1+s2) of time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in d.

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.

Device	Name	Description
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

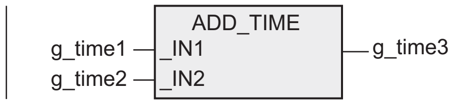


Program example

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]

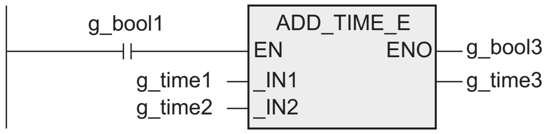


[ST]

```
g_time3:=ADD_TIME(g_time1,g_time2);
```

2) Function with EN/ENO(ADD_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=ADD_TIME_E(g_bool1,g_time1,g_time2,g_time3);
```

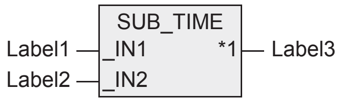
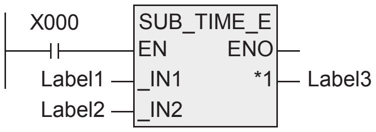
13.2 SUB_TIME(_E) / Subtraction

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs subtraction of time data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
SUB_TIME		SUB_TIME(_IN1,_IN2); Example: Label3:= SUB_TIME(Label1,Label2);
SUB_TIME_E		SUB_TIME_E(EN,_IN1,_IN2, Output_label); Example: SUB_TIME_E(X000,Label1, Label2,Label3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get subtraction	Time
	_IN2 (s2)	Head word device which stores subtraction data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

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

Explanation of function and operation

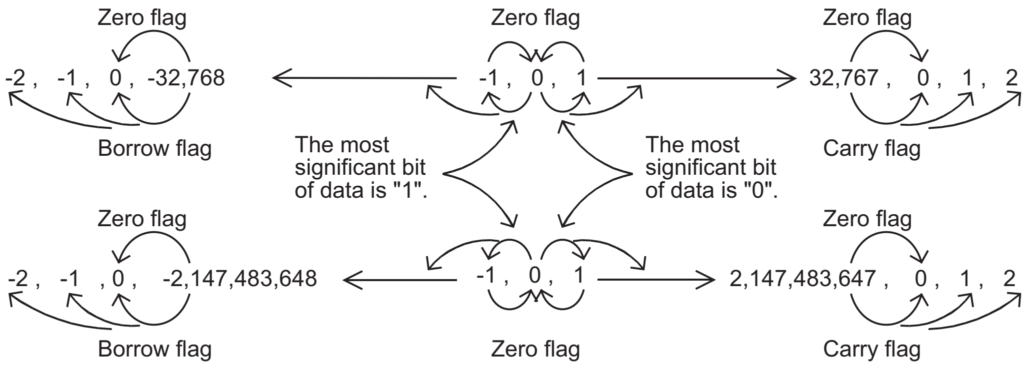
This function performs subtraction (s1-s2) of time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in d.

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.

Device	Name	Description
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

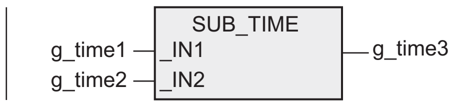


Program example

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]

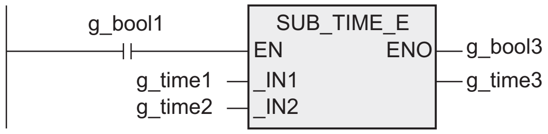


[ST]

```
g_time3:=SUB_TIME(g_time1,g_time2);
```

2) Function with EN/ENO(SUB_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=SUB_TIME_E(g_bool1,g_time1,g_time2,g_time3);
```

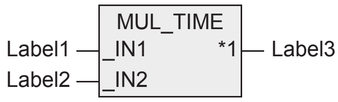
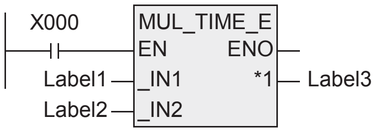
13.3 MUL_TIME(_E) / Multiplication

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs multiplication of time data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
MUL_TIME		MUL_TIME(_IN1,_IN2); Example: Label3:= MUL_TIME(Label1,Label2);
MUL_TIME_E		MUL_TIME_E(EN,_IN1,_IN2, Output_label); Example: MUL_TIME_E(X000,Label1, Label2,Label3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get multiplication	Time
	_IN2 (s2)	Multiplication data, or head word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

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 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.
("MUL_TIME_E" outputs "TRUE" from ENO.)
4) 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, multiplication (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(MUL_TIME)

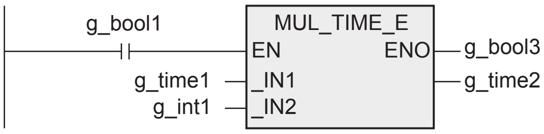
[Structured ladder/FBD]

[ST]

```
g_time2:=MUL_TIME(g_time1,g_int1);
```

2) Function with EN/ENO(MUL_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=MUL_TIME_E(g_bool1,g_time1,g_int1,g_time2);
```

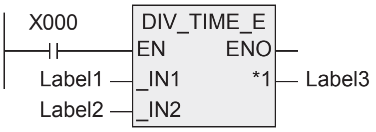
13.4 DIV_TIME(_E) / Division

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function performs division using time data.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
DIV_TIME		DIV_TIME(_IN1,_IN2); Example: Label3:= DIV_TIME(Label1,Label2);
DIV_TIME_E		DIV_TIME_E(EN,_IN1,_IN2, Output_label); Example: DIV_TIME_E(X000,Label1, Label2,Label3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get division	Time
	_IN2 (s2)	Division data, or head word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

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

Explanation of function and operation

1) This function performs division (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 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 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 (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(DIV_TIME)

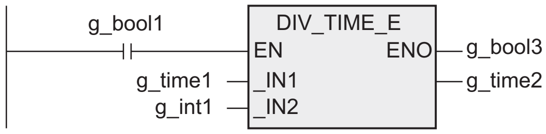
[Structured ladder/FBD]

[ST]

```
g_time2:=DIV_TIME(g_time1,g_int1);
```

2) Function with EN/ENO(DIV_TIME_E)

[Structured ladder/FBD]



[ST]

```
g_bool3:=DIV_TIME_E(g_bool1,g_time1,g_int1,g_time2);
```

14. Standard Function Blocks

Function name	Function	Reference
R_TRIG(_E)	Rising edge detector	Section 14.1
F_TRIG(_E)	Falling edge detector	Section 14.2
CTU(_E)	Up counter	Section 14.3
CTD(_E)	Down counter	Section 14.4
CTUD(_E)	Up/Down counter	Section 14.5
TP(_E) TP_10(_E)	Pulse timer	Section 14.6
TON(_E) TON_10(_E)	On delay timer	Section 14.7
TOF(_E) TOF_10(_E)	Off delay timer	Section 14.8
COUNTER_FB_M	Counter function blocks	Section 14.9
TIMER_10_FB_M	Timer function blocks	Section 14.10
TIMER_CONT_FB_M	Timer function blocks	Section 14.11
TIMER_100_FB_M	Timer function blocks	Section 14.12

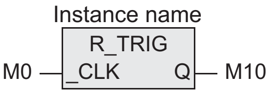
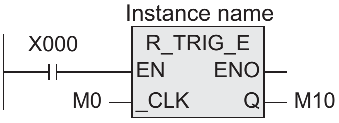
14.1 R_TRIG(_E) / Rising edge detector

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function block detects the rising edge of a signal, and outputs pulse signal.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
R_TRIG		R_TRIG(_CLK,Q); Example: Instance name(_CLK:=M0, Q:=M10);
R_TRIG_E		R_TRIG_E(EN,_CLK,Q,ENO); Example: Instance name(EN:=X000, _CLK:=M0,Q:=M10);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_CLK (s)	Input signal whose rising edge is to be detected	Bit
Output variable	ENO	Execution status	Bit
	Q (d)	Output signal	Bit

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 d when a device specified in s turns ON, and keeps ON the device specified in d 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

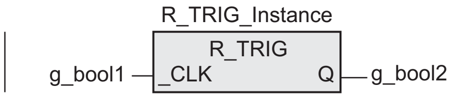
1) When an output number is specified in d and the specified output number does not exist due to indexing, M8316 (I/O inexistence error) turns ON.
   (Applicable to the FX3U and FX3UC PLCs only)
2) When a device (M, T or C) other than I/O number is specified in d and the specified device number does not exist due to indexing, an operation error (Error code: 6706) occurs.

Program example

In this program, a device specified in d turns ON when the bit data stored in a device specified in s turns ON from OFF, and the device specified in d remains ON only for 1 operation cycle.

1) Function without EN/ENO(R_TRIG)

[Structured ladder/FBD]

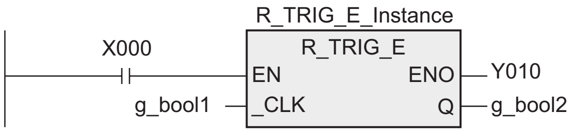


[ST]

```
R_TRIG_Instance(_CLK:=g_bool1,Q:=g_bool2);
```

2) Function with EN/ENO(R_TRIG_E)

[Structured ladder/FBD]



[ST]

```
R_TRIG_E_Instance(EN:=X000,_CLK:=g_bool1,Q:=g_bool2,ENO:=Y010);
```

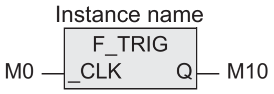
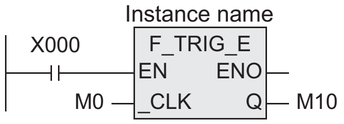
14.2 F_TRIG(_E) / Falling edge detector

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This function block detects the falling edge of a signal, and outputs pulse signal.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
F_TRIG		F_TRIG(_CLK,Q); Example: Instance name(_CLK:=M0, Q:=M10);
F_TRIG_E		F_TRIG_E(EN,_CLK,Q,ENO); Example: Instance name(EN:=X000, _CLK:=M0,Q:=M10);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_CLK (s)	Input signal whose falling edge is to be detected	Bit
Output variable	ENO	Execution status	Bit
	Q (d)	Output signal	Bit

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 d when a device specified in s turns OFF, and keeps ON the device specified in d 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

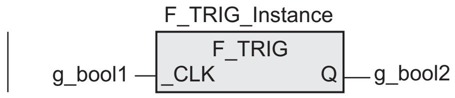
1) When an output number is specified in d and the specified output number does not exist due to indexing, M8316 (I/O inexistence error) turns ON.
(Applicable to the FX3U and FX3UC PLCs only)
2) When a device (M, T or C) other than I/O number is specified in d and the specified device number does not exist due to indexing, an operation error (Error code: 6706) occurs.

Program example

In this program, a device specified in (d) turns ON when the bit data stored in a device specified in (s) turns OFF from ON, and the device specified in (d) remains ON only for 1 operation cycle.

1) Function without EN/ENO(F_TRIG)

[Structured ladder/FBD]

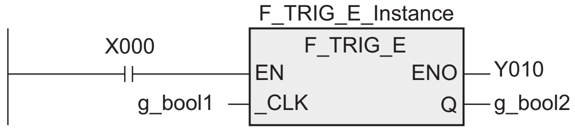


[ST]

```
F_TRIG_Instance(_CLK:=g_bool1,Q:=g_bool2);
```

2) Function with EN/ENO(F_TRIG_E)

[Structured ladder/FBD]



[ST]

```
F_TRIG_E_Instance(EN:=X000,_CLK:=g_bool1,Q:=g_bool2,ENO:=Y010);
```

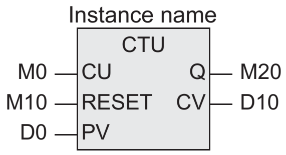
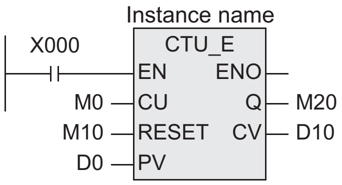
14.3 CTU(_E) / Up counter

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	×	×

Outline

This function block counts up the number of times of rising of a signal.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	**ST**
CTU		CTU(CU,RESET,PV,Q,CV); Example: Instance name(CU:=M0, RESET:=M10,PV:=D0,Q:=M20, CV:=D10);
CTU_E		CTU_E(EN,CU,RESET,PV,Q,CV, ENO); Example: Instance name(EN:=X000, CU:=M0,RESET:=M10,PV:=D0, Q:=M20,CV:=D10);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	CU (s1)	Count source signal	Bit
	RESET (s2)	Reset input signal	Bit
	PV (n)	Counter set value	Word [signed]
Output variable	ENO	Execution status	Bit
	Q (d1)	Count-up output signal	Bit
	CV (d2)	Number of times of rising	Word [signed]

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 d2 when a device specified in s1 turns ON.

When the count value reaches a value specified in n, a device specified in d1 turns ON.

When a device specified in s2 turns ON, this function block turns OFF a device specified in d1, and resets the count value of a device specified in d2.

Cautions

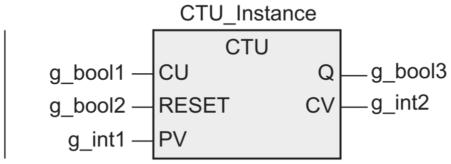
1) Use the function having "_E" in its name to connect a bus.
2) Expression of function blocks in each language
   - Set the instance when using a function block.
   - Describe the instance name when programming a function block.

Program example

In this program, the number of times the bit data stored in a device specified in (s1) turns ON from OFF is counted, and the count value is output to a device specified in (d2).

1) Function without EN/ENO(CTU)

[Structured ladder/FBD]

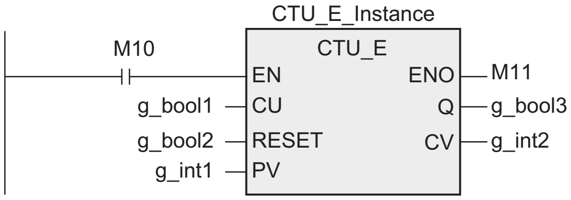


[ST]

```
CTU_Instance(CU:=g_bool1,RESET:=g_bool2,PV:=g_int1,Q:=g_bool3,CV:=g_int2);
```

2) Function with EN/ENO(CTU_E)

[Structured ladder/FBD]



[ST]

```
CTU_E_Instance(EN:=M10,CU:=g_bool1,RESET:=g_bool2,PV:=g_int1,Q:=g_bool3,CV:=g_int2,ENO:=M11);
```

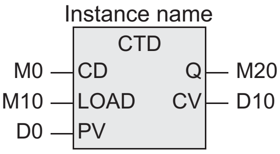
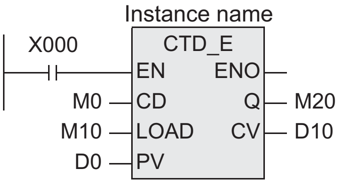
14.4 CTD(_E) / Down counter

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	×	×

Outline

This function block counts down the number of times of rising of a signal.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
CTD		CTD(CD,LOAD,PV,Q,CV); Example: Instance name(CD:=M0, LOAD:=M10,PV:=D0,Q:=M20, CV:=D10);
CTD_E		CTD_E(EN,CD,LOAD,PV,Q,CV, ENO); Example: Instance name(EN:=X000, CD:=M0,LOAD:=M10,PV:=D0, Q:=M20,CV:=D10);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	CD (s1)	Count source signal	Bit
	LOAD (s2)	Reset input signal	Bit
	PV (n)	Counter set value	Word [signed]
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal (which turns ON when the current counter value becomes "0" or less)	Bit
	CV (d2)	Number of times of rising	Word [signed]

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 d2 when a device specified in s1 turns ON.
The value n specifies the initial value for subtraction.
This function block turns ON a device specified in d1 when the count value becomes "0".
When a device specified in s2 turns ON, this function block turns OFF a device specified in d1, and sets the initial value for subtraction specified in n to the count value of a device specified in d2.

Cautions

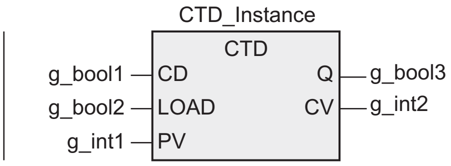
1) Use the function having "_E" in its name to connect a bus.
2) Expression of function blocks in each language
   - Set the instance when using a function block.
   - Describe the instance name when programming a function block.

Program example

In this program, the number of times the bit data stored in a device specified in s1 turns ON from OFF is counted, and a device specified in d1 turns ON when the value stored in a device specified in d2 becomes "0".

1) Function without EN/ENO(CTD)

[Structured ladder/FBD]

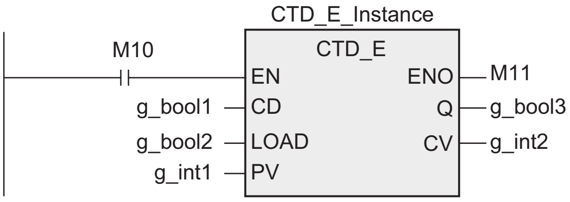


[ST]

```
CTD_Instance(CD:=g_bool1,LOAD:=g_bool2,PV:=g_int1,Q:=g_bool3,CV:=g_int2);
```

2) Function with EN/ENO(CTD_E)

[Structured ladder/FBD]



[ST]

```
CTD_E_Instance(EN:=M10,CD:=g_bool1,LOAD:=g_bool2,PV:=g_int1,Q:=g_bool3,CV:=g_int2,ENO:=M11);
```

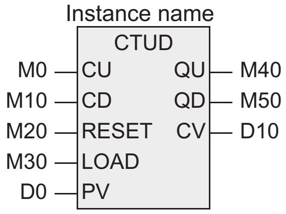
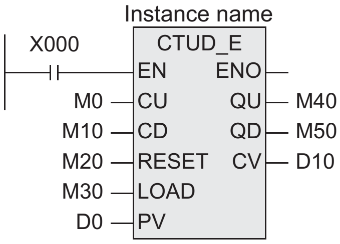
14.5 CTUD(_E) / Up/Down counter

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	×	×

Outline

This function block counts up/down the number of times of rising of a signal.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
CTUD		CTUD(CU,CD,RESET,LOAD,PV, QU,QD,CV); Example: Instance name(CU:=M0, CD:=M10,RESET:=M20, LOAD:=M30,PV:=D0,QU:=M40, QD:=M50,CV:=D10);
CTUD_E		CTUD_E(EN,CU,CD,RESET, LOAD,PV,QU,QD,CV,ENO); Example: Instance name(EN:=X000, CU:=M0,CD:=M10,RESET:=M20, LOAD:=M30,PV:=D0,QU:=M40, QD:=M50,CV:=D10);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	CU (s1)	Count up signal	Bit
	CD (s2)	Count down signal	Bit
	RESET (s3)	Reset input signal	Bit
	LOAD (s4)	Resetting signal	Bit
	PV (n)	Counter set value	Word [signed]
Output variable	ENO	Execution status	Bit
	QU (d1)	Count-up output signal	Bit
	QD (d2)	Output signal (which turns ON when the current counter value becomes "0" or less)	Bit
	CV (d3)	Count value data	Word [signed]

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 d3 when a device specified in s1 turns ON.

This function block counts down (subtracts "1" from) the value stored in a device specified in d3 when a device specified in s2 turns ON.

n specifies the maximum value of the counter.

When the value stored in a device specified in d3 reaches the maximum value n of the counter, a device specified in d1 turns ON.

When the value stored in a device specified in d3 becomes "0", a device specified in d2 turns ON.

This function block resets the count value of a device specified in d3 when a device specified in s3 turns ON.

This function block sets the value stored in n to a device specified in d3 when a device specified in s4 turns ON.

Cautions

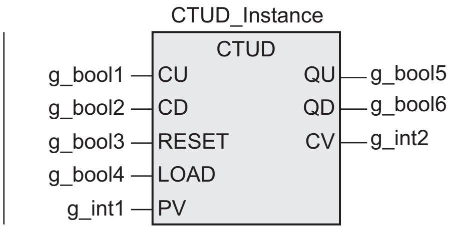
1) Use the function having "_E" in its name to connect a bus.
2) Expression of function blocks in each language
   - Set the instance when using a function block.
   - Describe the instance name when programming a function block.

Program example

In this program, the number of times the bit data stored in a device specified in (s1) turns ON from OFF is counted up (added by "1"). When the value stored in a device specified in (d3) reaches the value specified in (n), a device specified in (d1) turns ON.
At the same time, the number of times the bit data stored in a device specified in (s2) turns ON from OFF is counted down (subtracted by "1"). When the value stored in a device specified in (d3) becomes "0", a device specified in (d2) turns ON.

1) Function without EN/ENO(CTUD)

[Structured ladder/FBD]

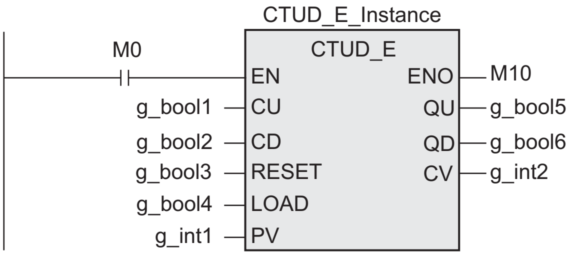


[ST]

```
CTUD_Instance(CU:=g_bool1,CD:=g_bool2,RESET:=g_bool3,LOAD:=g_bool4,PV:=g_int1,QU:=g_bool5,
QD:=g_bool6,CV:=g_int2);
```

2) Function with EN/ENO(CTUD_E)

[Structured ladder/FBD]



[ST]

```
CTUD_E_Instance(EN:=M0,CU:=g_bool1,CD:=g_bool2,RESET:=g_bool3,LOAD:=g_bool4,PV:=g_int1,
QU:=g_bool5,QD:=g_bool6,CV:=g_int2,ENO:=M10);
```

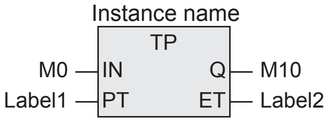
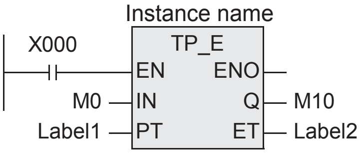
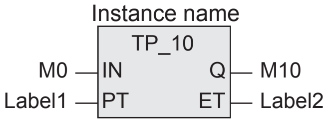
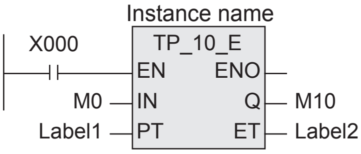
14.6 TP(_E), TP_10(_E) / Pulse timer

	FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
TP(_E)	○	○	○	○	○	○	○	×	×
TP_10(_E)	○	○	×	○	○	×	○	×	×

Outline

This function block keeps ON a signal for specified duration.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TP	 	TP(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1,Q:=M10, ET:=Label2);
TP_E	 	TP_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0,PT:=Label1,Q:=M10, ET:=Label2);
TP_10	 	TP_10(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1,Q:=M10, ET:=Label2);
TP_10_E	 	TP_10_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0,PT:=Label1,Q:=M10, ET:=Label2);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	IN (s)	ON start input signal	Bit
	PT (n)	ON duration data	Time
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal	Bit
	ET (d2)	ON duration current value	Time

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 (n).
The elapsed time while a device specified in (d1) remains ON is set to a device specified in (d2).
A device specified in (d1) turns OFF when the elapsed time reaches the set value.
Even if a device specified in (d1) turns OFF, this function block does not reset the elapsed time. When a device specified in (s) turns 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 (n).
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

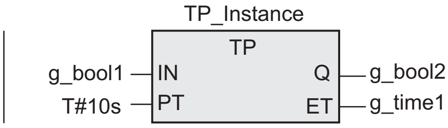
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit 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 d1 turns ON and remains ON for 10 seconds.

1) Function without EN/ENO(TP,TP_10)

[Structured ladder/FBD]

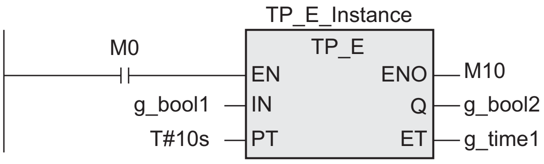


[ST]

```
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]



[ST]

```
TP_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1,ENO:=M10);
```

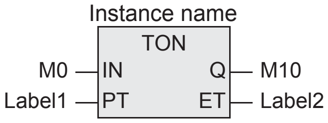
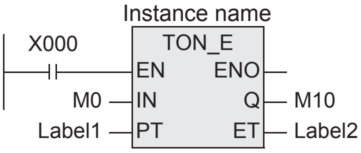
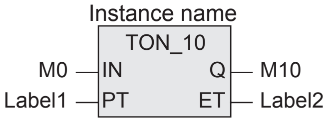
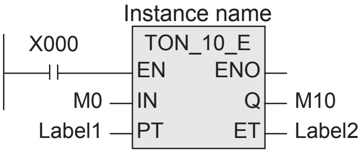
14.7 TON(_E), TON_10(_E) / On delay timer

	FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
TON(_E)	○	○	○	○	○	○	○	×	×
TON_10(_E)	○	○	×	○	○	×	○	×	×

Outline

This function block turns ON after specified time.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TON		TON(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1,Q:=M10, ET:=Label2);
TON_E		TON_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0,PT:=Label1,Q:=M10, ET:=Label2);
TON_10		TON_10(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1,Q:=M10, ET:=Label2);
TON_10_E		TON_10_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0,PT:=Label1,Q:=M10, ET:=Label2);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	IN (s)	Input signal	Bit
	PT (n)	ON start time data	Time
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal	Bit
	ET (d2)	ON start time current value	Time

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 after the time specified in n.
The delay elapsed time until a device specified in d1 turns ON is set to a device specified in d2.
When a device specified in s turns OFF, this function block turns OFF a device specified in d1 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

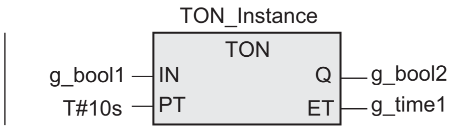
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit 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 (d1) turns ON 10 seconds later.

1) Function without EN/ENO(TON,TON_10)

[Structured ladder/FBD]

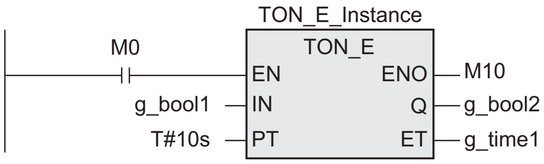


[ST]

```
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]



[ST]

```
TON_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1,ENO:=M10);
```

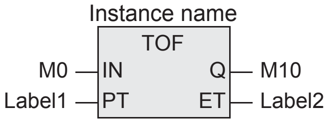
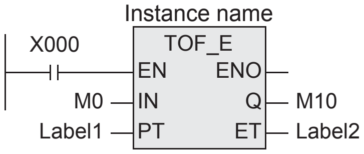
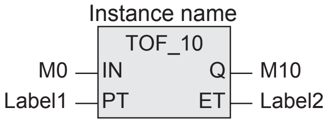
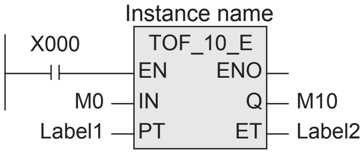
14.8 TOF(_E), TOF_10(_E) / Off delay timer

	FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
TOF(_E)	○	○	○	○	○	○	○	×	×
TOF_10(_E)	○	○	×	○	○	×	○	×	×

Outline

When the input signal turns OFF, this function block turns OFF the output signal after the specified time.

1. Format

Function name	Expression in each language	
	Structured ladder/FBD	ST
TOF		TOF(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1,Q:=M10, ET:=Label2);
TOF_E		TOF_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0,PT:=Label1,Q:=M10, ET:=Label2);
TOF_10		TOF_10(IN,PT,Q,ET); Example: Instance name(IN:=M0, PT:=Label1,Q:=M10, ET:=Label2);
TOF_10_E		TOF_10_E(EN,IN,PT,Q,ET,ENO); Example: Instance name(EN:=X000, IN:=M0,PT:=Label1,Q:=M10, ET:=Label2);

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	IN (s)	Input signal	Bit
	PT (n)	OFF start time data	Time
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal	Bit
	ET (d2)	OFF duration current value	Time

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.
   When a device specified in s turns OFF from ON, this function block turns OFF a device specified in d1 after the time specified in n.
   The elapsed time until a device specified in d1 turns OFF is set to a device specified in d2.
   When a device specified in s turns ON again, this function block turns ON a device specified in d1 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

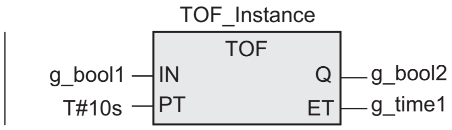
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit 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 d1 turns ON. When bit data stored in a device specified in s turns OFF, bit data stored in a device specified in d1 turns OFF 10 seconds later.

1) Function without EN/ENO(TOF,TOF_10)

[Structured ladder/FBD]

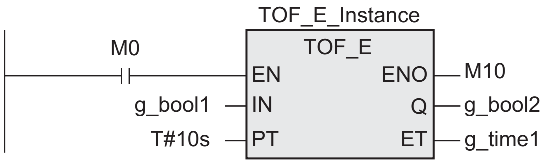


[ST]

```
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]



[ST]

```
TOF_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s,Q:=g_bool2,ET:=g_time1,ENO:=M10);
```

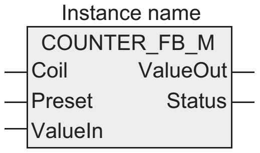
14.9 COUNTER_FB_M / Counter function blocks

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
COUNTER_FB_M		COUNTER_FB_M(Coil,Preset,ValueIn,ValueOut,Status);

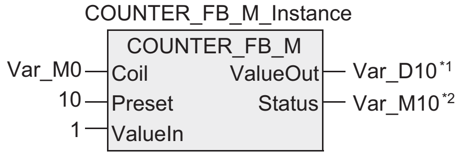
2. Set data

Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Counter set value	Word [signed]
	ValueIn	Counter initial value	Word [signed]
Output variable	ValueOut	Counter current value	ANY16
	Status	Counter output contact	Bit

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.

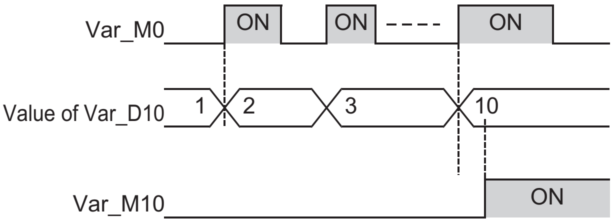
[Structured ladder/FBD]



[ST]

COUNTER_FB_M_Instance(Coil:= Var_M0,Preset:=10,
ValueIn:=1,ValueOut:=Var_D10,Status:=Var_M10);

timing chart

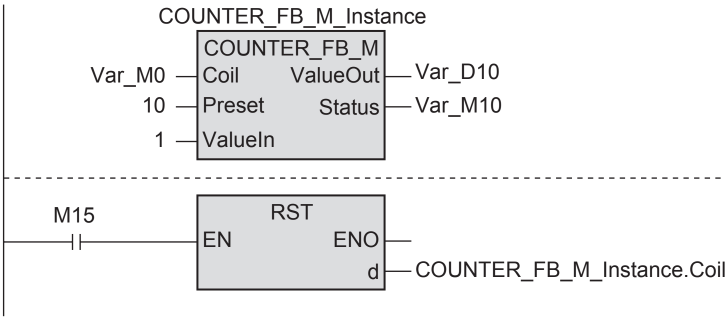


*1. Var_D10 is a global label and is defined as D10.

*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]



[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.

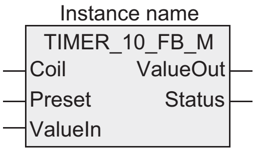
14.10 TIMER_10_FB_M / Timer function blocks

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	×	○	○	×	○	×	×

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIMER_10_FB_M		TIMER_10_FB_M(Coil,Preset, ValueIn,ValueOut,Status);

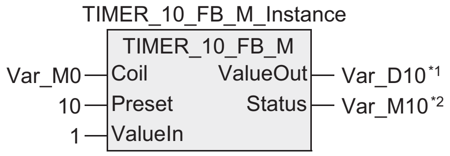
2. Set data

Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Timer set value	Word [signed]
	ValueIn	Initial timer value	Word [signed]
Output variable	ValueOut	Current timer value	ANY16
	Status	Timer output contact	Bit

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.

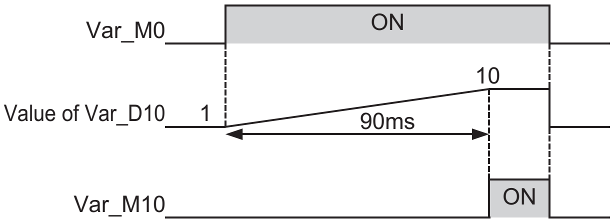
[Structured ladder/FBD]



[ST]

TIMER_10_FB_M_Instance(Coil:= Var_M0,Preset:= 10,
ValueIn:= 1,ValueOut:= Var_D10,Status:=Var_M10);

timing chart



*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.

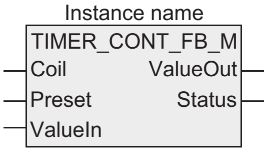
14.11 TIMER_CONT_FB_M / Timer function blocks

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	×	○	×	×

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIMER_CONT_FB_M		TIMER_CONT_FB_M(Coil, Preset,ValueIn,ValueOut,Status);

2. Set data

Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Timer set value	Word [signed]
	ValueIn	Initial timer value	Word [signed]
Output variable	ValueOut	Current timer value	ANY16
	Status	Timer output contact	Bit

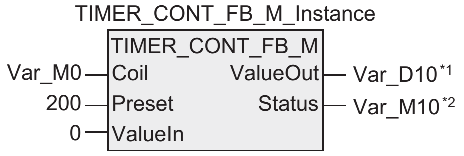
Function and operation explanation

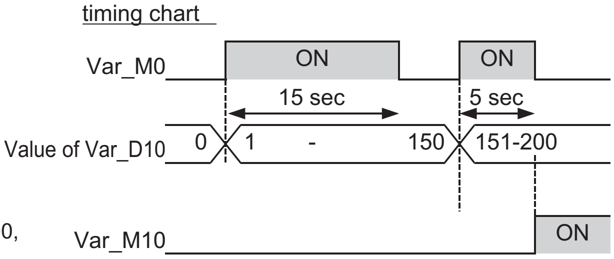
1) This is a retentive timer that counts the time when the variable is ON. It starts counting the current value when the execution condition of the input argument Coil turns ON.
The timer starts counting from "ValueIn × 1 ms or 100 ms". When it counts up to "Preset × 1 ms or 100 ms", the output argument 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]





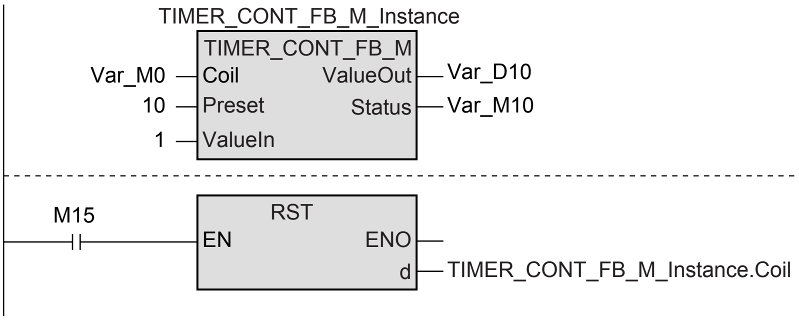
[ST]
TIMER_CONT_FB_M_Instance(Coil:= Var_M0,Preset:= 200, ValueIn:= 0,ValueOut:= Var_D10,Status:=Var_M10);

*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]



[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.

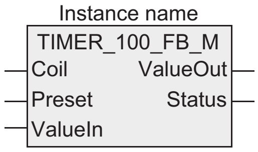
14.12 TIMER_100_FB_M / Timer function blocks

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

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

Function name	Expression in each language	
	Structured ladder/FBD	ST
TIMER_100_FB_M		TIMER_100_FB_M(Coil,Preset, ValueIn,ValueOut,Status);

2. Set data

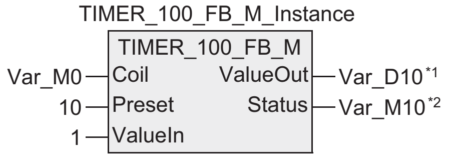
Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Timer set value	Word [signed]
	ValueIn	Initial timer value	Word [signed]
Output variable	ValueOut	Current timer value	ANY16
	Status	Timer output contact	Bit

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.

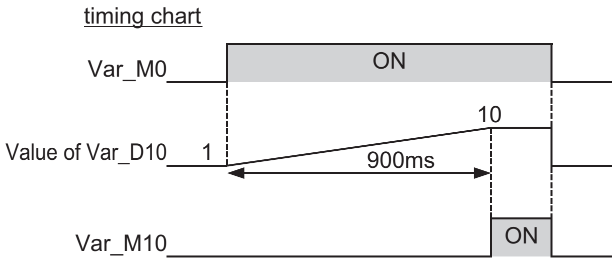
2) When the execution condition of the input argument Coil turns OFF, the current value takes on the value of ValueIn and the output argument Status also turns OFF.

[Structured ladder/FBD]



[ST]

TIMER_100_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.

15. Operator

Function name	Function	Reference
ADD	Addition	Section 15.1
SUB	Subtraction	Section 15.2
MUL	Multiplication	Section 15.3
DIV	Division	Section 15.4
MOD	Modulus operation	Section 15.5
**	Exponentiation	Section 15.6
AND	Logical product	Section 15.7
OR	Logical sum	Section 15.8
XOR	Exclusive logical sum	Section 15.9
NOT	Logical negation	Section 15.10
GT	Comparison	Section 15.11
GE	Comparison	Section 15.12
EQ	Comparison	Section 15.13
LE	Comparison	Section 15.14
LT	Comparison	Section 15.15
NE	Comparison	Section 15.16

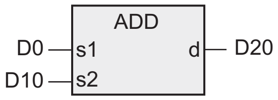
15.1 ADD / Addition

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator performs addition using two values (A + B = C), and outputs the operation result.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
ADD		s1+s2; Example: D20:=D0+D10;

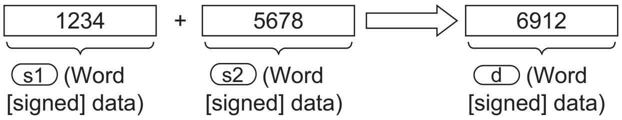
In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Data for addition or word device which stores such data	ANY_NUM
Output variable	d	Word device which will store the operation result	ANY_NUM

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 d using the data type of data stored in devices specified in s1 to s28.
Example: When the data type is word [signed]



2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

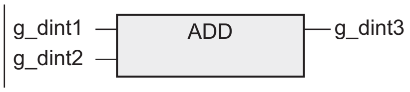
Cautions

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]



[ST]

```
g_dint3:=(g_dint1)+(g_dint2);
```

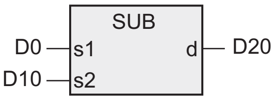
15.2 SUB / Subtraction

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator performs subtraction using two values (A - B = C), and outputs the operation result.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
SUB		s1-s2; Example: D20:=D0-D10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

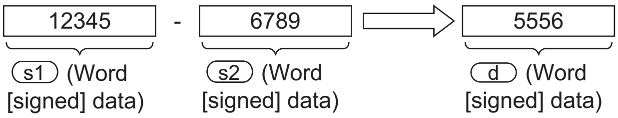
2. Set data

Variable		Description	Data type
Input variable	s1	Data to be subtracted or word device which stores such data	ANY_NUM
	s2	Data for subtraction or word device which stores such data	ANY_NUM
Output variable	d	Word device which will store the operation result	ANY_NUM

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]



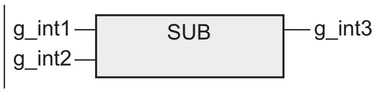
Cautions

Refer to Section 7.2.

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 d.

[Structured ladder/FBD]



[ST]

```
g_int3:=(g_int1)-(g_int2);
```

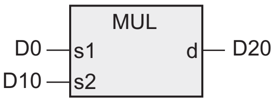
15.3 MUL / Multiplication

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator performs multiplication using two or more values (A × B = C), and outputs the operation result.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
MUL		s1*s2; Example: D20:=D0*D10;

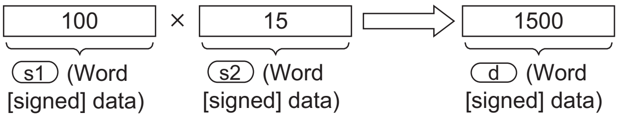
In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Data for multiplication or word device which stores such data	ANY_NUM
Output variable	d	Word device which will store the operation result	ANY_NUM

Explanation of function and operation

1) This function performs multiplication (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 d using the data type of data stored in devices specified in s1 to s28.
Example: When the data type is word [signed]



2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

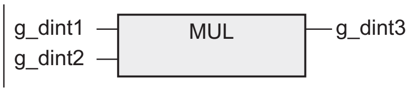
Cautions

Refer to Section 7.3.

Program example

In this program, multiplication is performed using double word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_dint3:=(g_dint1)*(g_dint2);
```

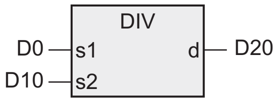
15.4 DIV / Division

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator performs division using two values (A / B = C … remainder), and outputs the quotient.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
DIV		s1/s2; Example: D20:=D0/D10;

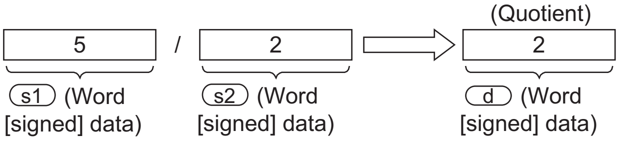
In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1	Data to be divided, or word device which stores such data	ANY_NUM
	s2	Data for division (divisor), or word device which stores such data	ANY_NUM
Output variable	d	Word device which will store the operation result	ANY_NUM

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]



Cautions

Refer to Section 7.4.

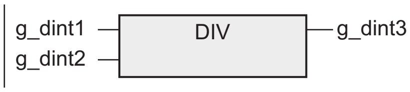
Error

Refer to Section 7.4.

Program example

In this program, division is performed using double word [signed] data stored in devices specified in s1 and s2, 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]



[ST]

```
g_dint3:=(g_dint1)/(g_dint2);
```

15.5 MOD / Modulus operation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator performs division using two values (A / B = C …remainder), and outputs the remainder.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
MOD	The MOD operator is not available in Structured ladder/ FBD language.	s1 MOD s2 ; Example: d:=s1 MOD s2 ;

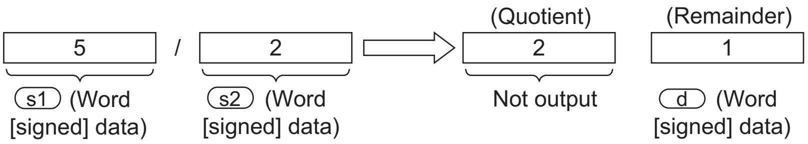
In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1	Data to be divided, or word device which stores such data	ANY_INT
	s2	Data for division (divisor), or word device which stores such data	ANY_INT
Output variable	d	Word device which will store the operation result	ANY_INT

Explanation of function and operation

This function performs division (s1/s2) using word [signed]/double word [signed] data stored in devices specified in s1 and s2, and outputs the remainder 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]



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 s1 and s2, and the remainder is output to a device specified in d using the data type of data stored in devices specified in s1 and s2.

[ST]

```
g_dint3:=g_dint1 MOD g_dint2;
```

15.6 ** / Exponentiation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×	×

Outline

This operator obtains raised result, and outputs it.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
**	The "**" operator is not available in Structured ladder/ FBD language.	s1 **s2; Example: d:=s1 **s2;

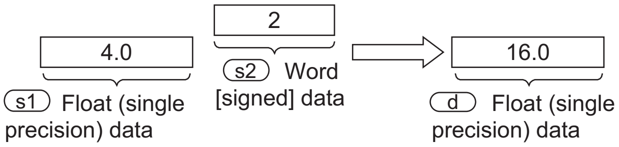
In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1	Data to be raised, or word device which stores such data	FLOAT (Single Precision)
	s2	Power data, or word device which stores such data	ANY_NUM
Output variable	d	Word device which will store the operation result	FLOAT (Single Precision)

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

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_real2:=EXPT(g_real1,g_int1);
```

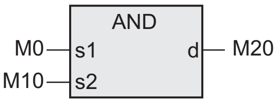
15.7 AND / Logical product

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator obtains the logical product of two or more bits, and outputs it.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
AND		s1 AND s2; Example: M20:=M0 AND M10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Device used to obtain the logical product	ANY_BIT
Output variable	d	Device which will store the operation result	ANY_BIT

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	1	0	0	0	0	1	1	1	1

Logical product

s2	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0

⇩

d	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0

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 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]

[ST]

```
g_word3:=(g_word1) AND (g_word2);
or
g_word3:=(g_word1) & (g_word2);
```

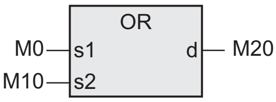
15.8 OR / Logical sum

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator obtains the logical sum of two or more bits, and outputs it.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
OR		s1 OR s2; Example: M20:=M0 OR M10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Device used to obtain the logical sum	ANY_BIT
Output variable	d	Device which will store the operation result	ANY_BIT

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 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	0	0	1	1	1	1	0	0	0	0	1	1	1	1

Logical sum

s2	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0	0

⇩

d	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1

2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

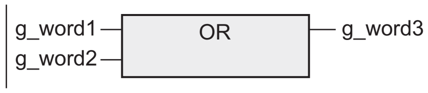
Cautions

Refer to Section 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 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]



[ST]

```
g_word3:=(g_word1) OR (g_word2);
```

15.9 XOR / Exclusive logical sum

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator obtains the logical sum of two or more bits, and outputs it.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
XOR		s1 XOR s2; Example: M20:=M0 XOR M10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Device used to obtain the exclusive logical sum	ANY_BIT
Output variable	d	Device which will store the operation result	ANY_BIT

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 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	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Exclusive logical sum

s2	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0

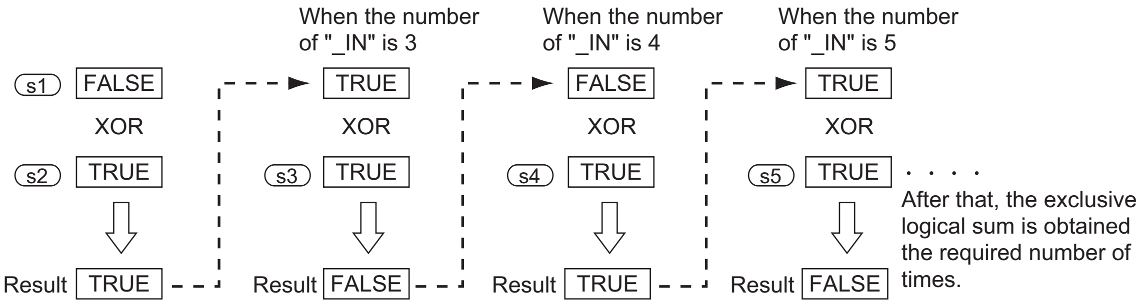
⇩

d	1	0	1	1	0	0	0	1	0	1	0	1	1	0	1	0

2) The number of pins for s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

3) If there are 3 or more s, the exclusive logical sum is obtained using the "exclusive logical sum of s1 and s2" and s3.
If there is s4, the exclusive logical sum is obtained using the "exclusive logical sum of "exclusive logical sum of s1 and s2" and "s3"" and s4. In this way, the exclusive logical sum is obtained the required number of times for all input labels s5 s6 …
Example: When the data type is bit



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 (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]

[ST]

```
g_word3:=(g_word1) XOR (g_word2);
```

15.10 NOT / Logical negation

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator obtains the logical negation of bits, and outputs it.

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
NOT	The NOT operator is not available in Structured ladder/ FBD language.	NOT s1; Example: d:=NOT(s);

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s	Device used to obtain the logical negation	ANY_BIT
Output variable	d	Device which will store the operation result	ANY_BIT

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]

s	0	1	1	0	1	0	1	1	0	0	0	0	1	1	1	1

Logical negation

d	1	0	0	1	0	1	0	0	1	1	1	1	0	0	0	0

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);
```

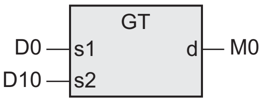
15.11 GT / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator compares data with regard to "> (larger)".

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
GT		s1>s2; Example: M0:=D0>D10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Compared data, or word device which stores such data	ANY_BIT
Output variable	d	Device which will store the comparison result	Bit

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 d.
   This function executes comparison [s1 > s2] & [s2 > s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

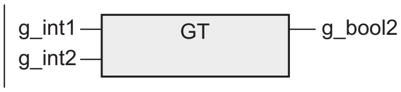
Cautions

Refer to Section 11.1.

Program examples

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool2:=(g_int1)>(g_int2);
```

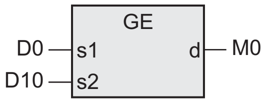
15.12 GE / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator compares data with regard to "≥ (larger or equal)".

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
GE		s1>=s2; Example: M0:=D0>=D10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	d	Device which will store the comparison result	Bit

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 d.
   This function executes comparison [s1 ≥ s2] & [s2 ≥ s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

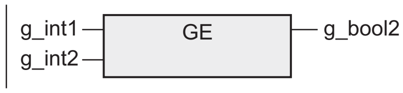
Cautions

Refer to Section 11.2.

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]



[ST]

```
g_bool2:=(g_int1)>=(g_int2);
```

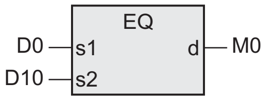
15.13 EQ / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator compares data with regard to "= (equal)".

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
EQ		s1=s2; Example: M0:=D0=D10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	d	Device which will store the comparison result	Bit

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 d.
This function executes comparison [s1 = s2] & [s2 = s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

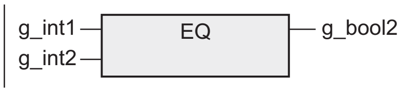
Cautions

Refer to Section 11.3.

Program examples

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool2:=(g_int1)=(g_int2);
```

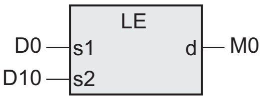
15.14 LE / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator compares data with regard to " ≤ (smaller or equal)".

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	**ST**
LE		s1<=s2; Example: M0:=D0<=D10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	d	Device which will store the comparison result	Bit

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 d.
This function executes comparison [s1 ≤ s2] & [s2 ≤ s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

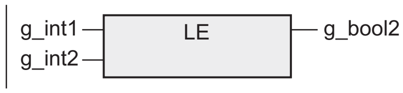
Cautions

Refer to Section 11.4.

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]



[ST]

```
g_bool2:=(g_int1)<=(g_int2);
```

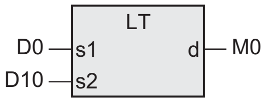
15.15 LT / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator compares data with regard to "< (smaller)".

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
LT		s1<s2; Example: M0:=D0<D10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s28	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	d	Device which will store the comparison result	Bit

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 d.
This function executes comparison [s1 < s2] & [s2 < s3] & … & [s n-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 s can be changed in the range of 2 to 28.

→ **Refer to Section 3. Function Construction**

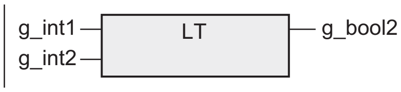
Cautions

Refer to Section 11.5.

Program examples

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder/FBD]



[ST]

```
g_bool2:=(g_int1)<(g_int2);
```

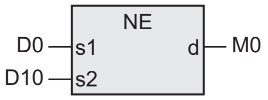
15.16 NE / Comparison

FX3U(C)	FX3G(C)	FX3S	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○	○

Outline

This operator compares data with regard to "≠ (unequal)".

1. Format

Operator name	Expression in each language	
	Structured ladder/FBD	ST
NE		s1<>s2; Example: M0:=D0<>D10;

In explanation of operators, the input variable is described as "s□" and the output variable is descried as "d".

2. Set data

Variable		Description	Data type
Input variable	s1 to s2	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	d	Device which will store the comparison result	Bit

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 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]

[ST]

```
g_bool2:=(g_int1)<>(g_int2);
```

Appendix A: Correspondence between Devices and Addresses

The table below shows the correspondence between devices and addresses.

Device			**Notation**		**Example of correspondence between device and address**	
			Device	**Address**	**Device**	**Address**
Input relay		X	Xn	%IXn	X367	%IX247
Output relay		Y	Yn	%QXn	Y367	%QX247
Auxiliary relay		M	Mn	%MX0.n	M499	%MX0.499
Timer	Contact	TS	Tn	%MX3.n	TS191	%MX3.191
	Coil	TC	Tn	%MX5.n	TC191	%MX5.191
	Current value	TN	Tn	%MW3.n %MD3.n	TN190 T190	%MW3.191 %MD3.190
Counter	Contact	CS	Cn	%MX4.n	CS99	%MX4.99
	Coil	CC	Cn	%MX6.n	CC99	%MX6.99
	Current value	CN	Cn	%MW4.n %MD4.n	CN98 C98	%MW4.99 %MD4.98
Data register		D	Dn	%MW0.n %MD0.n	D198* D198	%MW0.199 %MD0.198
Intelligent function unit device		G	Ux\Gn	%MW14.x.n %MD14.x.n	U0\G09 U0\G09	%MW14.0.10 %MD14.0.9
Extension register		R	Rn	%MW2.n %MD2.n	R32766 R32766	%MW2.32767 %MD2.32766
Extension file register		ER	ERn	No correspondence	-	-
Pointer		P	Pn	" "(NULL character)	P4095	No correspondence
Interrupt pointer		I	In	No correspondence	-	-
Nesting		N	Nn	No correspondence	-	-
Index register		Z	Zn	%MW7.n %MD7.n	Z6 Z6	%MW7.7 %MD7.6
		V	Vn	%MW6.n	V7	%MW6.7
State		S	Sn	%MX2.n	S4095	%MX2.4095

MEMO

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

Appendix B-1 [By type]

1. Type Conversion Functions

Function name	Function	Ref. Page
BOOL_TO_INT(_E)	Converts bit data into word [signed] data.	40
BOOL_TO_DINT(_E)	Converts bit data into double word [signed] data.	42
BOOL_TO_STR(_E)	Converts bit data into string data.	44
BOOL_TO_WORD(_E)	Converts bit data into word [unsigned]/bit string [16-bit] data.	46
BOOL_TO_DWORD(_E)	Converts bit data into double word [unsigned]/bit string [32-bit] data.	48
BOOL_TO_TIME(_E)	Converts bit data into time data.	50
INT_TO_DINT(_E)	Converts word [signed] data into double word [signed] data	52
DINT_TO_INT(_E)	Converts double word [signed] data into word [signed] data.	54
INT_TO_BOOL(_E)	Converts word [signed] data into bit data.	56
DINT_TO_BOOL(_E)	Converts double word [signed] data into bit data.	58
INT_TO_REAL(_E)	Converts word [signed] data into float (single precision) data.	60
DINT_TO_REAL(_E)	Converts double word [signed] data into float (single precision) data.	62
INT_TO_STR(_E)	Converts word [signed] data into string data.	64
DINT_TO_STR(_E)	Converts double word [signed] data into string data.	66
INT_TO_WORD(_E)	Converts word [signed] data into word [unsigned]/bit string [16-bit] data.	68
DINT_TO_WORD(_E)	Converts double word [signed] data into word [unsigned]/bit string [16-bit] data.	70
INT_TO_DWORD(_E)	Converts word [signed] data into double word [unsigned]/bit string [32-bit] data.	72
DINT_TO_DWORD(_E)	Converts double word [signed] data into double word [unsigned]/bit string [32-bit] data.	74
INT_TO_BCD(_E)	Converts word [signed] data into BCD data.	76
DINT_TO_BCD(_E)	Converts double word [signed] data into BCD data.	78
INT_TO_TIME(_E)	Converts word [signed] data into time data.	80
DINT_TO_TIME(_E)	Converts double word [signed] data into time data.	82
REAL_TO_INT(_E)	Converts float (single precision) data into word [signed] data.	84
REAL_TO_DINT(_E)	Converts float (single precision) data into double word [signed] data.	86
REAL_TO_STR(_E)	Converts float (single precision) data into string data.	88
WORD_TO_BOOL(_E)	Converts word [unsigned]/bit string [16-bit] data into bit data.	91
DWORD_TO_BOOL(_E)	Converts double word [unsigned]/bit string [32-bit] data into bit data.	93
WORD_TO_INT(_E)	Converts word [unsigned]/bit string [16-bit] data into word [signed] data.	95
WORD_TO_DINT(_E)	Converts word [unsigned]/bit string [16-bit] data into double word [signed] data.	97
DWORD_TO_INT(_E)	Converts double word [unsigned]/bit string [32-bit] data into word [signed] data.	99
DWORD_TO_DINT(_E)	Converts double word [unsigned]/bit string [32-bit] data into double word [signed] data.	101
WORD_TO_DWORD(_E)	Converts word [unsigned]/bit string [16-bit] data into double word [unsigned]/bit string [32-bit].	103
DWORD_TO_WORD(_E)	Converts double word [unsigned]/bit string [32-bit] data into word [unsigned]/bit string [16-bit] data.	105
WORD_TO_TIME(_E)	Converts word [unsigned]/bit string [16-bit] data into time data.	107
DWORD_TO_TIME(_E)	Converts double word [unsigned]/bit string [32-bit] data into time data.	109
STR_TO_BOOL(_E)	Converts string data into bit data.	111
STR_TO_INT(_E)	Converts string data into word [signed] data.	113
STR_TO_DINT(_E)	Converts string data into double word [signed] data.	115
STR_TO_REAL(_E)	Converts string data into float (single precision) data.	117
STR_TO_TIME(_E)	Converts string data into time data.	120
BCD_TO_INT(_E)	Converts BCD data into word [signed] data.	122
BCD_TO_DINT(_E)	Converts BCD data into double word [signed] data.	124
TIME_TO_BOOL(_E)	Converts time data into bit data.	126
TIME_TO_INT(_E)	Converts time data into word [signed] data.	128
TIME_TO_DINT(_E)	Converts time data into double word [signed] data.	130
TIME_TO_STR(_E)	Converts time data into string data.	132
TIME_TO_WORD(_E)	Converts time data into word [unsigned]/bit string [16-bit] data.	134

Function name	Function	Ref. Page
TIME_TO_DWORD(_E)	Converts time data into double word [unsigned]/bit string [32-bit] data.	136
BITARR_TO_INT(_E)	Converts specified number of bits of a bit array into word [signed] data or word [unsigned]/bit string [16-bit] data.	138
BITARR_TO_DINT(_E)	Converts specified number of bits of a bit array into double word [signed] data or double word [unsigned]/bit string [32-bit] data.	140
INT_TO_BITARR(_E)	Outputs low-order "n" bits of word [signed] data or word [unsigned]/bit string [16-bit] data to a bit array.	142
DINT_TO_BITARR(_E)	Outputs low-order "n" bits of double word [signed] data or double word [unsigned]/bit string [32-bit] data to a bit array.	144
CPY_BITARR(_E)	Copies specified number of bits of a bit array.	146
GET_BIT_OF_INT(_E)	Reads the value of a specified bit of word [signed] data.	148
SET_BIT_OF_INT(_E)	Writes a value to a specified bit of word [signed] data.	150
CPY_BIT_OF_INT(_E)	Copies a specified bit of word [signed] data to a specified bit of another word [signed] data.	152
GET_BOOL_ADDR	Outputs start data as bit data.	154
GET_INT_ADDR	Outputs start data as word [signed] data.	155
GET_WORD_ADDR	Outputs start data as word [unsigned]/bit string [16-bit] data.	156

2. Standard Functions Of One Numeric Variable

Function name	Function	Ref. Page
ABS(_E)	Obtains the absolute value.	158

3. Standard Arithmetic Functions

Function name	Function	Ref. Page
ADD_E	Adds data. (Number of pins variable)	161
SUB_E	Subtracts data.	163
MUL_E	Multiplies data. (Number of pins variable)	165
DIV_E	Divides data (, and outputs the quotient).	167
MOD(_E)	Divides data (, and outputs the remainder).	169
EXPT(_E)	Obtains the raised result.	171
MOVE(_E)	Transfers data.	173

4. Standard Bit Shift Functions

Function name	Function	Ref. Page
SHL(_E)	Shifts bits leftward.	176
SHR(_E)	Shifts bits rightward.	178

5. Standard Bitwise Boolean Functions

Function name	Function	Ref. Page
AND_E	Obtains the logical product. (Number of pins variable)	181
OR_E	Obtains the logical sum. (Number of pins variable)	183
XOR_E	Obtains the exclusive logical sum. (Number of pins variable)	185
NOT(_E)	Obtains the logical not.	187

6. Standard Selection Functions

Function name	Function	Ref. Page
SEL(_E)	Selects data in accordance with the input condition.	190
MAXIMUM(_E)	Searches the maximum value. (Number of pins variable)	192
MINIMUM(_E)	Searches the minimum value. (Number of pins variable)	194
LIMITATION(_E)	Judges whether data is located within the range between the upper limit value and the lower limit value.	196
MUX(_E)	Selects data, and outputs it. (Number of pins variable)	198

7. Standard Comparison Functions

Function name	Function	Ref. Page
GT_E	Compares data with regard to "> (larger)". (Number of pins variable)	201
GE_E	Compares data with regard to "≥ (larger or equal)". (Number of pins variable)	203
EQ_E	Compares data with regard to "= (equal)". (Number of pins variable)	205
LE_E	Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)	207
LT_E	Compares data with regard to "< (smaller)". (Number of pins variable)	209
NE_E	Compares data with regard to "≠ (unequal)".	211

8. Standard Character String Functions

Function name	Function	Ref. Page
MID(_E)	Obtains a character string from a specified position.	214
CONCAT(_E)	Connects character strings. (Number of pins variable)	217
INSERT(_E)	Inserts a character string.	219
DELETE(_E)	Deletes a character string.	222
REPLACE(_E)	Replaces a character string.	224
FIND(_E)	Searches a character string.	227

9. Functions Of Time Data Types

Function name	Function	Ref. Page
ADD_TIME (_E)	Adds time data.	231
SUB_TIME (_E)	Subtracts time data.	233
MUL_TIME (_E)	Multiplies time data.	235
DIV_TIME (_E)	Divides time data.	237

10.Standard Function Blocks

Function name	Function	Ref. Page
R_TRIG (_E)	Detects the rising edge of a signal, and outputs pulse signal.	240
F_TRIG (_E)	Detects the falling edge of a signal, and outputs pulse signal.	242
CTU(_E)	Counts up the number of times of rising of a signal.	244
CTD(_E)	Counts down the number of times of rising of a signal.	246
CTUD(_E)	Counts up/down the number of times of rising of a signal.	248
TP(_E) TP_10(_E)	Keeps ON a signal during specified time duration.	250
TON(_E) TON_10(_E)	Keeps OFF a signal during specified time duration.	252
TOF(_E) TOF_10(_E)	Turns OFF the output signal at specified time after the input signal turned OFF.	254
COUNTER_FB_M	Counter drive	256
TIMER_10_FB_M	10ms timer drive	258
TIMER_CONT_FB_M	Retentive timer drive	259
TIMER_100_FB_M	100ms timer drive	261

11.Operator (Arithmetic operations)

Operator name		Function	Ref. Page
Structured ladder /FBD	ST		
ADD	+	Adds data. (Number of pins variable)	263
SUB	-	Subtracts data.	264
MUL	*	Multiplies data. (Number of pins variable)	265
DIV	/	Divides data (, and outputs the quotient).	266
-	MOD	Divides data (, and outputs the remainder).	267
-	**	Obtains the raised result	268

12.Operator (Logical operations)

Operator name		Function	Ref. Page
Structured ladder /FBD	ST		
AND	& AND	Obtains the logical product. (Number of pins variable)	269
OR	OR	Obtains the logical sum. (Number of pins variable)	270
XOR	XOR	Obtains the exclusive logical sum. (Number of pins variable)	271
-	NOT	Obtains the logical not.	273

13.Operator (Comparison operations)

Operator name		Function	Ref. Page
Structured ladder /FBD	ST		
GT	>	Compares data with regard to "> (larger)". (Number of pins variable)	274
GE	>=	Compares data with regard to "≥ (larger or equal)". (Number of pins variable)	275
EQ	=	Compares data with regard to "= (equal)". (Number of pins variable)	276
LE	<=	Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)	277
LT	<	Compares data with regard to "< (smaller)". (Number of pins variable)	278
NE	<>	Compares data with regard to "≠ (unequal)".	279

Appendix B-2 [In alphabetic order]

Functions

Function name	Function	Ref. Page
A		
ABS(_E)	Obtains the absolute value.	158
ADD_TIME (_E)	Adds time data.	231
ADD_E	Adds data. (Number of pins variable)	161
AND_E	Obtains the logical product. (Number of pins variable)	181
B		
BCD_TO_DINT(_E)	Converts BCD data into double word [signed] data.	124
BCD_TO_INT(_E)	Converts BCD data into word [signed] data.	122
BITARR_TO_DINT(_E)	Converts specified number of bits of a bit array into double word [signed] data or double word [unsigned]/bit string [32-bit] data.	140
BITARR_TO_INT(_E)	Converts specified number of bits of a bit array into word [signed] data or word [unsigned]/bit string [16-bit] data.	138
BOOL_TO_DINT(_E)	Converts bit data into double word [signed] data.	42
BOOL_TO_DWORD(_E)	Converts bit data into double word [unsigned]/bit string [32-bit] data.	48
BOOL_TO_INT(_E)	Converts bit data into word [signed] data.	40
BOOL_TO_STR(_E)	Converts bit data into string data.	44
BOOL_TO_TIME(_E)	Converts bit data into time data.	50
BOOL_TO_WORD(_E)	Converts bit data into word [unsigned]/bit string [16-bit] data.	46
C		
CONCAT(_E)	Connects character strings. (Number of pins variable)	217
COUNTER_FB_M	Counter drive	256
CPY_BITARR(_E)	Copies specified number of bits of a bit array.	146
CPY_BIT_OF_INT(_E)	Copies a specified bit of word [signed] data to a specified bit of another word [signed] data.	152
CTD(_E)	Counts down the number of times of rising of a signal.	246
CTUD(_E)	Counts up/down the number of times of rising of a signal.	248
CTU(_E)	Counts up the number of times of rising of a signal.	244
D		
DELETE(_E)	Deletes a character string.	222
DINT_TO_BCD(_E)	Converts double word [signed] data into BCD data.	78
DINT_TO_BITARR(_E)	Outputs low-order "n" bits of double word [signed] data or double word [unsigned]/bit string [32-bit] data to a bit array.	144
DINT_TO_BOOL(_E)	Converts double word [signed] data into bit data.	58
DINT_TO_DWORD(_E)	Converts double word [signed] data into double word [unsigned]/bit string [32-bit] data.	74
DINT_TO_INT(_E)	Converts double word [signed] data into word [signed] data.	54
DINT_TO_REAL(_E)	Converts double word [signed] data into float (single precision) data.	62
DINT_TO_STR(_E)	Converts double word [signed] data into string data.	66
DINT_TO_TIME(_E)	Converts double word [signed] data into time data.	82
DINT_TO_WORD(_E)	Converts double word [signed] data into word [unsigned]/bit string [16-bit] data.	70
DIV_TIME (_E)	Divides time data.	237
DIV_E	Divides data (, and outputs the quotient).	167
DWORD_TO_BOOL (_E)	Converts double word [unsigned]/bit string [32-bit] data into bit data.	93
DWORD_TO_DINT(_E)	Converts double word [unsigned]/bit string [32-bit] data into double word [signed] data.	101
DWORD_TO_INT(_E)	Converts double word [unsigned]/bit string [32-bit] data into word [signed] data.	99
DWORD_TO_TIME(_E)	Converts double word [unsigned]/bit string [32-bit] data into time data.	109
DWORD_TO_WORD (_E)	Converts double word [unsigned]/bit string [32-bit] data into word [unsigned]/bit string [16-bit] data.	105
E		
EQ_E	Compares data with regard to "= (equal)". (Number of pins variable)	205
EXPT(_E)	Obtains the raised result.	171
F		
FIND(_E)	Searches a character string.	227
F_TRIG(_E)	Detects the falling edge of a signal, and outputs pulse signal.	242
G		
GE_E	Compares data with regard to "≥ (larger or equal)". (Number of pins variable)	203
GET_BIT_OF_INT(_E)	Reads the value of a specified bit of word [signed] data.	148
GET_BOOL_ADDR	Outputs start data as bit data.	154
GET_INT_ADDR	Outputs start data as word [signed] data.	155
GET_WORD_ADDR	Outputs start data as word [unsigned]/bit string [16-bit] data.	156
GT_E	Compares data with regard to "> (larger)". (Number of pins variable)	201
I		
INSERT(_E)	Inserts a character string.	219
INT_TO_BCD(_E)	Converts word [signed] data into BCD data.	78
INT_TO_BITARR(_E)	Outputs low-order "n" bits of word [signed] data or word [unsigned]/bit string [16-bit] data to a bit array.	142
INT_TO_BOOL(_E)	Converts word [signed] data into bit data.	56
INT_TO_DINT(_E)	Converts word [signed] data into double word [signed] data.	52

Function name	Function	Ref. Page
I		
INT_TO_DWORD(_E)	Converts word [signed] data into double word [unsigned]/bit string [32-bit] data.	72
INT_TO_REAL(_E)	Converts word [signed] data into float (single precision) data.	60
INT_TO_STR(_E)	Converts word [signed] data into string data.	64
INT_TO_TIME(_E)	Converts word [signed] data into time data.	80
INT_TO_WORD(_E)	Converts word [signed] data into word [unsigned]/bit string [16-bit] data.	68
L		
LE_E	Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)	207
LIMITATION(_E)	Judges whether data is located within the range between the upper limit value and the lower limit value.	196
LT_E	Compares data with regard to "< (smaller)". (Number of pins variable)	209
M		
MAXIMUM(_E)	Searches the maximum value. (Number of pins variable)	192
MID(_E)	Obtains a character string from a specified position.	214
MINIMUM(_E)	Searches the minimum value. (Number of pins variable)	194
MOD(_E)	Divides data (, and outputs the remainder).	169
MOVE(_E)	Transfers data.	173
MUL_TIME(_E)	Multiplies time data.	235
MUL_E	Multiplies data. (Number of pins variable)	165
MUX(_E)	Selects data, and outputs it. (Number of pins variable)	198
N		
NE_E	Compares data with regard to "≠ (unequal)".	211
NOT(_E)	Obtains the logical not.	187
O		
OR_E	Obtains the logical sum. (Number of pins variable)	183
R		
REAL_TO_DINT(_E)	Converts float (single precision) data into double word [signed] data.	86
REAL_TO_INT(_E)	Converts float (single precision) data into word [signed] data.	84
REAL_TO_STR(_E)	Converts float (single precision) data into string data.	88
REPLACE(_E)	Replaces a character string.	224
R_TRIG(_E)	Detects the rising edge of a signal, and outputs pulse signal.	240
S		
SEL(_E)	Selects data in accordance with the input condition.	190
SET_BIT_OF_INT(_E)	Writes a value to a specified bit of word [signed] data.	150
SHL(_E)	Shifts bits leftward.	176
SHR(_E)	Shifts bits rightward.	178
STR_TO_BOOL(_E)	Converts string data into bit data.	111

Function name	Function	Ref. Page
S		
STR_TO_DINT(_E)	Converts string data into double word [signed] data.	115
STR_TO_INT(_E)	Converts string data into word [signed] data.	113
STR_TO_REAL(_E)	Converts string data into float (single precision) data.	117
STR_TO_TIME(_E)	Converts string data into time data.	120
SUB_TIME(_E)	Subtracts time data.	233
SUB_E	Subtracts data.	163
T		
TIME_TO_BOOL(_E)	Converts time data into bit data.	126
TIME_TO_DINT(_E)	Converts time data into double word [signed] data.	130
TIME_TO_DWORD(_E)	Converts time data into double word [unsigned]/bit string [32-bit] data.	136
TIME_TO_INT(_E)	Converts time data into word [signed] data.	128
TIME_TO_STR(_E)	Converts time data into string data.	132
TIME_TO_WORD(_E)	Converts time data into word [unsigned]/bit string [16-bit] data.	134
TIMER_CONT_FB_M	Retentive timer drive	259
TIMER_10_FB_M	10ms timer drive	258
TIMER_100_FB_M	100ms timer drive	261
TOF(_E) TOF_10(_E)	Turns OFF the output signal at specified time after the input signal turned OFF.	254
TON(_E) TON_10(_E)	Keeps OFF a signal during specified time duration.	252
TP(_E) TP_10(_E)	Keeps ON a signal during specified time duration.	250
W		
WORD_TO_BOOL(_E)	Converts word [unsigned]/bit string [16-bit] data into bit data.	91
WORD_TO_DINT(_E)	Converts word [unsigned]/bit string [16-bit] data into double word [signed] data.	97
WORD_TO_DWORD(_E)	Converts word [unsigned]/bit string [16-bit] data into double word [unsigned]/bit string [32-bit].	103
WORD_TO_INT(_E)	Converts word [unsigned]/bit string [16-bit] data into word [signed] data.	95
WORD_TO_TIME(_E)	Converts word [unsigned]/bit string [16-bit] data into time data.	107
X		
XOR_E	Obtains the exclusive logical sum. (Number of pins variable)	185

Operator

Operator name	Function	Ref. Page
Symbol		
+	Adds data. (Number of pins variable)	263
-	Subtracts data.	264
*	Multiplies data. (Number of pins variable)	265
/	Divides data (, and outputs the quotient).	266
**	Obtains the raised result.	268
&	Obtains the logical product. (Number of pins variable)	269
>	Compares data with regard to "> (larger)". (Number of pins variable)	274
>=	Compares data with regard to "≥ (larger or equal)". (Number of pins variable)	275
=	Compares data with regard to "= (equal)". (Number of pins variable)	276
<=	Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)	277
<	Compares data with regard to "< (smaller)". (Number of pins variable)	278
<>	Compares data with regard to "≠ (unequal)".	279
A		
ADD	Adds data. (Number of pins variable)	263
AND	Obtains the logical product. (Number of pins variable)	269
D		
DIV	Divides data (, and outputs the quotient).	266
E		
EQ	Compares data with regard to "= (equal)". (Number of pins variable)	276
G		
GE	Compares data with regard to "≥ (larger or equal)". (Number of pins variable)	275
GT	Compares data with regard to "> (larger)". (Number of pins variable)	274
L		
LE	Compares data with regard to "≤ (smaller or equal)". (Number of pins variable)	277
LT	Compares data with regard to "< (smaller)". (Number of pins variable)	278
M		
MOD	Divides data (, and outputs the remainder).	267
MUL	Multiplies data. (Number of pins variable)	265
N		
NE	Compares data with regard to "≠ (unequal)".	279
NOT	Obtains the logical not.	273
O		
OR	Obtains the logical sum. (Number of pins variable)	270
S		
SUB	Subtracts data.	264
X		
XOR	Obtains the exclusive logical sum. (Number of pins variable)	271

MEMO

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, applications in which human life or property that could be greatly affected, such as in aircraft, medical applications, incineration and fuel devices, manned transportation, equipment for recreation and amusement, and safety devices, shall also be excluded from the programmable logic controller range of applications.
   However, 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

Date of preparation	Revision	Description
1/2009	A	First Edition.
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

MITSUBISHI ELECTRIC CORPORATION
HEAD OFFICE: TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN

MODEL	FX-KP-OK-E
MODEL CODE	09R927

JY997D34801K
(MEE)

Effective April 2015
Specifications are subject to change without notice.