Before using MELSEC-Q or -L series programmable controllers, please read the manuals included with each product and the relevant manuals introduced in those manuals carefully, and pay full attention to safety to handle the product correctly.

Make sure that the end users read the manuals included with each product, and keep the manuals in a safe place for future reference.
(1) Mitsubishi programmable controller ("the PRODUCT") shall be used in conditions;
   i) where any problem, fault or failure occurring in the PRODUCT, if any, shall not lead to any major or serious accident; and
   ii) where the backup and fail-safe function are systematically or automatically provided outside of the PRODUCT for the case of any problem, fault or failure occurring in the PRODUCT.

(2) The PRODUCT has been designed and manufactured for the purpose of being used in general industries.
MITSUBISHI SHALL HAVE NO RESPONSIBILITY OR LIABILITY (INCLUDING, BUT NOT LIMITED TO ANY AND ALL RESPONSIBILITY OR LIABILITY BASED ON CONTRACT, WARRANTY, TORT, PRODUCT LIABILITY) FOR ANY INJURY OR DEATH TO PERSONS OR LOSS OR DAMAGE TO PROPERTY CAUSED BY the PRODUCT THAT ARE OPERATED OR USED IN APPLICATION NOT INTENDED OR EXCLUDED BY INSTRUCTIONS, PRECAUTIONS, OR WARNING CONTAINED IN MITSUBISHI’S USER, INSTRUCTION AND/OR SAFETY MANUALS, TECHNICAL BULLETINS AND GUIDELINES FOR the PRODUCT.
("Prohibited Application")
Prohibited Applications include, but not limited to, the use of the PRODUCT in;
• Nuclear Power Plants and any other power plants operated by Power companies, and/or any other cases in which the public could be affected if any problem or fault occurs in the PRODUCT.
• Railway companies or Public service purposes, and/or any other cases in which establishment of a special quality assurance system is required by the Purchaser or End User.
• Aircraft or Aerospace, Medical applications, Train equipment, transport equipment such as Elevator and Escalator, Incineration and Fuel devices, Vehicles, Manned transportation, Equipment for Recreation and Amusement, and Safety devices, handling of Nuclear or Hazardous Materials or Chemicals, Mining and Drilling, and/or other applications where there is a significant risk of injury to the public or property.

Notwithstanding the above, restrictions Mitsubishi may in its sole discretion, authorize use of the PRODUCT in one or more of the Prohibited Applications, provided that the usage of the PRODUCT is limited only for the specific applications agreed to by Mitsubishi and provided further that no special quality assurance or fail-safe, redundant or other safety features which exceed the general specifications of the PRODUCTs are required. For details, please contact the Mitsubishi representative in your region.
<table>
<thead>
<tr>
<th>Print date</th>
<th>Manual number</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jul., 2008</td>
<td>SH(NA)-080784ENG-A</td>
<td>First edition</td>
</tr>
<tr>
<td>Jan., 2009</td>
<td>SH(NA)-080784ENG-B</td>
<td>Model Addition: Q00UJCPU, Q00UCPU, Q01UCPU, Q10UDHCPU, Q10UDEHCPU, Q20UDHCPU, Q20UDEHCPU, Section 2.1.5, Section 5.5.2, Section 5.5.3, Correction: GENERIC TERMS AND ABBREVIATIONS IN THIS MANUAL, Section 3.7.1</td>
</tr>
<tr>
<td>Jul., 2009</td>
<td>SH(NA)-080784ENG-C</td>
<td>Model Addition: Q00JCPU, Q00CPU, Q01CPU, Addition: Section 2.2, Section 3.1, Correction: Section 1.2, Section 2.1, Section 2.1.1 to Section 2.1.12 are changed to Section 2.2.1 to Section 2.2.12, Section 3.1 is changed to Section 3.2, Chapter 4, Chapter 5, Deletion: Section 3.2 to Section 3.7, Appendix 1, Appendix 2</td>
</tr>
<tr>
<td>Jan., 2010</td>
<td>SH(NA)-080784ENG-D</td>
<td>Model Addition: L02CPU, L26CPU-BT, Addition: CONDITIONS OF USE FOR THE PRODUCT, MANUALS, Section 1.1, Section 1.2, Section 3.1, Chapter 4, Correction: Section 1.1, Section 1.2</td>
</tr>
<tr>
<td>Apr., 2010</td>
<td>SH(NA)-080784ENG-E</td>
<td>Model Addition: Q50UDEHCPU, Q100UDEHCPU, Correction: Section 1.1, Section 1.2</td>
</tr>
<tr>
<td>Sep., 2010</td>
<td>SH(NA)-080784ENG-F</td>
<td>Addition: Section 5.1.39, Section 5.1.40, Section 5.1.41, Section 5.1.42, Section 5.1.43, Section 5.1.44, Section 5.1.45, Correction: Section 2.2.1, Section 5.1.30, Section 5.1.31, Section 5.1.33, Section 5.6.1</td>
</tr>
<tr>
<td>Jan., 2011</td>
<td>SH(NA)-080784ENG-G</td>
<td>Correction: Section 1.1, Section 2.2, Section 5.1.13, Section 5.1.19, Section 5.1.25, Section 5.1.29, Section 5.1.30, Section 5.1.31, Section 5.1.32, Section 5.1.33, Section 5.1.34, Section 5.1.36, Section 5.1.37, Section 5.2.1, Section 5.3.6, Section 5.5.1, Section 5.5.2, Section 5.5.3, Section 5.5.4, Section 5.6.1, Section 5.7.1, Section 5.8.1, Section 5.8.2, Section 5.8.4, Section 5.11.1, Section 5.11.2, Section 5.11.3, Section 5.11.4, Section 5.12.1, Section 5.12.2, Section 5.12.3, Section 5.12.4</td>
</tr>
<tr>
<td>Print date</td>
<td>Manual number</td>
<td>Revision</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>Mar., 2011</td>
<td>SH(NA)-080784ENG-H</td>
<td>Addition&lt;br&gt;Section 2.3, Chapter 6&lt;br&gt;Correction&lt;br&gt;Section 1.1, Section 2.1, Section 2.2.3, Section 2.2.4, Section 2.2.6, Section 5.1.10, Section 5.1.15, Section 5.1.32, Section 5.1.34, Section 5.1.38, Section 5.2.1, Section 5.3.1, Section 5.3.2, Section 5.3.3, Section 5.3.4, Section 5.3.6, Section 5.3.7, Section 5.4.1, Section 5.6.1, Section 5.8.1, Section 5.8.2, Section 5.8.3, Section 5.8.4, Section 5.9.1, Section 5.9.2, Section 5.10.1, Section 5.10.2, Section 5.11.1, Section 5.11.2, Section 5.11.3, Section 5.11.4, Section 5.12.1, Section 5.12.2, Section 5.12.3, Section 5.12.4</td>
</tr>
<tr>
<td>Jul., 2011</td>
<td>SH(NA)-080784ENG-I</td>
<td>Model Addition&lt;br&gt;L02CPU-P, L26CPU-PBT&lt;br&gt;Correction&lt;br&gt;Section 1.2, Section 2.3, Chapter 4, Section 5.1.14, Section 5.1.16, Section 5.1.17, Section 5.1.18, Section 5.1.19, Section 5.1.45, Section 5.2.1, Section 5.3.1, Section 5.3.2, Section 5.3.3, Section 5.3.4, Section 5.3.6, Section 5.3.7, Section 5.5.1, Section 5.5.2, Section 5.5.3, Section 5.5.4, Section 5.6.1, Section 5.8.3, Section 5.8.4, Section 5.11.14</td>
</tr>
<tr>
<td>Feb., 2013</td>
<td>SH(NA)-080784ENG-J</td>
<td>Descriptions concerning the model additions of a Process CPU, Redundant CPU, Universal model QCPU, and LPCPU&lt;br&gt;Model Addition&lt;br&gt;Q02PHCPU, Q06PHCPU, Q12PHCPU, Q25PHCPU, Q12PRHCPU, Q25PRHCPU, Q03UDVCPU, Q04UDVCPU, Q06UDVCPU, Q13UDVCPU, Q26UDVCPU, L02SCPU, L06CPU, L26CPU</td>
</tr>
<tr>
<td>Jun., 2013</td>
<td>SH(NA)-080784ENG-K</td>
<td>Model Addition&lt;br&gt;L02SCPLU-P, L06CPU-P, L26CPU-P&lt;br&gt;Correction&lt;br&gt;Section 1.2</td>
</tr>
</tbody>
</table>

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

© 2008 MITSUBISHI ELECTRIC CORPORATION
INTRODUCTION

Thank you for purchasing the Mitsubishi MELSEC-Q or -L series programmable controllers.

Before using this product, please read this manual and the relevant manuals carefully and develop familiarity with the programming specifications to handle the product correctly.

When applying the program examples introduced in this manual to an actual system, ensure the applicability and confirm that it will not cause system control problems.

CONTENTS

SAFETY PRECAUTIONS ........................................................................................................... A - 1
CONDITIONS OF USE FOR THE PRODUCT.............................................................................................. A - 2
REVISIONS...................................................................................................................... A - 3
INTRODUCTION................................................................................................................... A - 5
CONTENTS ...................................................................................................................................................... A - 5
MANUALS......................................................................................................................................................... A - 9

1. OVERVIEW 1 - 1 to 1 - 6
  1.1 Purpose of This Manual 1 - 2
  1.2 Terms 1 - 5

2. FUNCTION TABLES 2 - 1 to 2 - 8
  2.1 How to Read Function Tables 2 - 2
  2.2 Function Tables 2 - 3
    2.2.1 Type conversion functions ........................................................................... 2 - 3
    2.2.2 Standard functions of one numeric variable ...................................................... 2 - 5
    2.2.3 Standard arithmetic functions .......................................................................... 2 - 5
    2.2.4 Standard bitwise Boolean functions .................................................................. 2 - 5
    2.2.5 Standard selection functions .......................................................................... 2 - 5
    2.2.6 Standard comparison functions ........................................................................ 2 - 6
    2.2.7 Standard character string functions ............................................................... 2 - 6
    2.2.8 Functions of time data types ............................................................................ 2 - 6
    2.2.9 Standard bistable function blocks ................................................................... 2 - 6
    2.2.10 Standard edge detection function blocks ....................................................... 2 - 7
    2.2.11 Standard counter function blocks ................................................................. 2 - 7
    2.2.12 Standard timer function blocks .................................................................... 2 - 7
  2.3 Operator Tables 2 - 8
    2.3.1 Arithmetic operations ...................................................................................... 2 - 8
    2.3.2 Logical operations ......................................................................................... 2 - 8
    2.3.3 Comparison operations .................................................................................. 2 - 8

3. CONFIGURATION OF FUNCTIONS 3 - 1 to 3 - 4
  3.1 Configuration of Functions 3 - 2
  3.2 Input Pins Variable Function 3 - 3

4. HOW TO READ FUNCTIONS 4 - 1 to 4 - 4
5. APPLICATION FUNCTIONS

5.1 Type Conversion Functions

5.1.1 Bit type \(\rightarrow\) word (signed), double word (signed) type conversion ........................................ 5 - 2
5.1.2 Bit type \(\rightarrow\) string type conversion ........................................................................................................... 5 - 5
5.1.3 Bit type \(\rightarrow\) word (unsigned)/16-bit string, double word (unsigned)/32-bit string type
conversion ..................................................................................................................................................................... 5 - 7
5.1.4 Bit type \(\rightarrow\) time type conversion .......................................................................................................................... 5 - 10
5.1.5 Word (signed) type \(\rightarrow\) double word (signed) type conversion ............................................................... 5 - 12
5.1.6 Double word (signed) type \(\rightarrow\) word (signed) type conversion ............................................................... 5 - 14
5.1.7 Word (signed), double word (signed) type \(\rightarrow\) bit type conversion ............................................................... 5 - 16
5.1.8 Word (signed), double word (signed) type \(\rightarrow\) single-precision real type conversion ....................... 5 - 19
5.1.9 Word (signed), double word (signed) type \(\rightarrow\) double-precision real type conversion .................... 5 - 22
5.1.10 Word (signed), double word (signed) type \(\rightarrow\) string type conversion ....................................................... 5 - 25
5.1.11 Word (signed), double word (signed) type \(\rightarrow\) word (unsigned)/16-bit string type conversion ...... 5 - 29
5.1.12 Word (signed), double word (signed) type \(\rightarrow\) double word (unsigned)/32-bit string type
conversion ..................................................................................................................................................................... 5 - 32
5.1.13 Word (signed), double word (signed) type \(\rightarrow\) BCD type conversion ....................................................... 5 - 35
5.1.14 Word (signed), double word (signed) type \(\rightarrow\) time type conversion ....................................................... 5 - 38
5.1.15 Single-precision real type \(\rightarrow\) word (signed), double word (signed) type conversion ..................... 5 - 41
5.1.16 Double-precision real type \(\rightarrow\) word (signed), double word (signed) type conversion ..................... 5 - 44
5.1.17 Single-precision real type \(\rightarrow\) double-precision real type conversion ..................................................... 5 - 47
5.1.18 Double-precision real type \(\rightarrow\) single-precision real type conversion ................................................. 5 - 50
5.1.19 Single-precision real type \(\rightarrow\) string type conversion .................................................................................. 5 - 53
5.1.20 Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type \(\rightarrow\) bit type
conversion ................................................................................................................................................................. 5 - 57
5.1.21 Word (unsigned)/16-bit string type \(\rightarrow\) word (signed), double word (signed) type
conversion ................................................................................................................................................................. 5 - 60
5.1.22 Double word (unsigned)/32-bit string type \(\rightarrow\) word (signed), double word (signed) type
conversion ................................................................................................................................................................. 5 - 63
5.1.23 Word (unsigned)/16-bit string type \(\rightarrow\) double word (unsigned)/32-bit string type conversion ...... 5 - 66
5.1.24 Double word (unsigned)/32-bit string type \(\rightarrow\) word (unsigned)/16-bit string type conversion ...... 5 - 69
5.1.25 Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type \(\rightarrow\) string type
conversion ................................................................................................................................................................. 5 - 72
5.1.26 Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type \(\rightarrow\) time type
conversion ................................................................................................................................................................. 5 - 75
5.1.27 String type \(\rightarrow\) bit type conversion ................................................................................................................... 5 - 78
5.1.28 String type \(\rightarrow\) word (signed), double word (signed) type conversion ................................................... 5 - 81
5.1.29 String type \(\rightarrow\) single-precision real type conversion ................................................................................... 5 - 84
5.1.30 String type \(\rightarrow\) word (unsigned)/16-bit string, double word (unsigned)/32-bit string type
conversion ................................................................................................................................................................. 5 - 88
5.1.31 String type \(\rightarrow\) time type conversion ................................................................................................................... 5 - 92
5.1.32 String type \(\rightarrow\) BCD type conversion ................................................................................................................... 5 - 95
5.1.33 BCD type \(\rightarrow\) word (signed), double word (signed) type conversion ....................................................... 5 - 100
5.1.34 BCD type \(\rightarrow\) string type conversion ................................................................................................................... 5 - 104
5.1.35 Time type \(\rightarrow\) bit type conversion ................................................................................................................... 5 - 107
5.1.36 Time type \(\rightarrow\) word (signed), double word (signed) type conversion ....................................................... 5 - 109
5.1.37 Time type \(\rightarrow\) string type conversion ................................................................................................................... 5 - 112
5.1.38 Time type \(\rightarrow\) word (unsigned)/16-bit string, double word (unsigned)/32-bit string type
conversion ................................................................................................................................................................. 5 - 114
5.1.39 Bit array \(\rightarrow\) word (signed) type, word (unsigned)/16-bit string type, double word (signed) type,
double word (unsigned)/32-bit string type conversion ............................................................................................. 5 - 117
5.1.40 Word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, double word (unsigned)/32-bit string type → bit array conversion .......................................................... 5 - 119
5.1.41 Bit array copy........................................................................................................................................ 5 - 121
5.1.42 Specified bit read of word (signed) type data .................................................................................... 5 - 123
5.1.43 Specified bit write of word (signed) type data ................................................................................... 5 - 125
5.1.44 Specified bit copy of word (signed) type data ................................................................................... 5 - 127
5.1.45 Nonessential type conversion ........................................................................................................... 5 - 129
5.2 Standard Functions of One Numeric Variable ....................................................................................... 5 - 131
5.2.1 Absolute value ...................................................................................................................................... 5 - 131
5.3 Standard Arithmetic Functions ............................................................................................................... 5 - 135
5.3.1 Addition.................................................................................................................................................. 5 - 135
5.3.2 Multiplication ....................................................................................................................................... 5 - 138
5.3.3 Subtraction .......................................................................................................................................... 5 - 141
5.3.4 Division ................................................................................................................................................ 5 - 144
5.3.5 Modules operation ................................................................................................................................. 5 - 147
5.3.6 Exponentiation....................................................................................................................................... 5 - 150
5.3.7 Move operation .................................................................................................................................... 5 - 153
5.4 Standard Bitwise Boolean Functions ....................................................................................................... 5 - 157
5.4.1 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT ............................................ 5 - 157
5.5 Standard Selection Functions .................................................................................................................. 5 - 162
5.5.1 Selection.............................................................................................................................................. 5 - 162
5.5.2 Maximum/Minimum selection .............................................................................................................. 5 - 165
5.5.3 Upper/Lower limit control .................................................................................................................. 5 - 168
5.5.4 Multiplexer ......................................................................................................................................... 5 - 171
5.6 Standard Comparison Functions ........................................................................................................... 5 - 174
5.6.1 Comparison ........................................................................................................................................ 5 - 174
5.7 Standard Character String Functions ................................................................................................... 5 - 178
5.7.1 Extract mid string ................................................................................................................................. 5 - 178
5.7.2 String concatenation ............................................................................................................................ 5 - 181
5.7.3 String insertion ................................................................................................................................... 5 - 184
5.7.4 String deletion .................................................................................................................................... 5 - 187
5.7.5 String replacement ............................................................................................................................... 5 - 190
5.8 Functions of Time Data Type .................................................................................................................. 5 - 193
5.8.1 Addition ............................................................................................................................................. 5 - 193
5.8.2 Subtraction ........................................................................................................................................ 5 - 196
5.8.3 Multiplication ..................................................................................................................................... 5 - 199
5.8.4 Division .............................................................................................................................................. 5 - 202
5.9 Standard Bistable Function Blocks ......................................................................................................... 5 - 204
5.9.1 Standard bistable function blocks (Set-dominant) ............................................................................ 5 - 204
5.9.2 Standard bistable function blocks (Reset-dominant) ....................................................................... 5 - 207
5.10 Standard Edge Detection Function Blocks .......................................................................................... 5 - 210
5.10.1 Rising edge detector .......................................................................................................................... 5 - 210
5.10.2 Falling edge detector ........................................................................................................................ 5 - 213
5.11 Standard Counter Function Blocks ....................................................................................................... 5 - 215
5.11.1 Up counter ....................................................................................................................................... 5 - 215
5.11.2 Down counter .................................................................................................................................. 5 - 218
5.11.3 Up/Down counter .............................................................................................................................. 5 - 221
5.11.4 Counter function blocks .................................................................................................................... 5 - 225

A-7
5.12 Standard Timer Function Blocks

5.12.1 Pulse timer

5.12.2 On delay timer

5.12.3 Off delay timer

5.12.4 Timer function blocks

6. OPERATOR

6.1 Arithmetic Operations

6.1.1 Addition

6.1.2 Multiplication

6.1.3 Subtraction

6.1.4 Division

6.1.5 Modules operation

6.1.6 Exponentiation

6.2 Logical Operations

6.2.1 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT

6.3 Comparison Operations

6.3.1 Comparison

INDEX
MANUALS

The manuals related to this product are listed below.
Order each manual as needed, referring to the following lists.

(1) Structured programming

<table>
<thead>
<tr>
<th>Manual name</th>
<th>Manual number (model code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MELSEC-Q/L/F Structured Programming Manual (Fundamentals)</td>
<td>SH-080782ENG (13JW06)</td>
</tr>
<tr>
<td>Methods and languages for structured programming</td>
<td>(Sold separately)</td>
</tr>
<tr>
<td>MELSEC-Q/L Structured Programming Manual (Common Instructions)</td>
<td>SH-080783ENG (13JW07)</td>
</tr>
<tr>
<td>Specifications and functions of common instructions, such as sequence</td>
<td>(Sold separately)</td>
</tr>
<tr>
<td>instructions, basic instructions, and application instructions, that can</td>
<td></td>
</tr>
<tr>
<td>be used in structured programs</td>
<td></td>
</tr>
<tr>
<td>MELSEC-Q/L Structured Programming Manual (Special Instructions)</td>
<td>SH-080785ENG (13JW09)</td>
</tr>
<tr>
<td>Specifications and functions of special instructions, such as module</td>
<td>(Sold separately)</td>
</tr>
<tr>
<td>dedicated instructions, PID control instructions, and built-in I/O</td>
<td></td>
</tr>
<tr>
<td>function instructions, that can be used in structured programs</td>
<td></td>
</tr>
</tbody>
</table>

(2) Operation of GX Works2

<table>
<thead>
<tr>
<th>Manual name</th>
<th>Manual number (model code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX Works2 Version 1 Operating Manual (Common)</td>
<td>SH-080779ENG (13JU63)</td>
</tr>
<tr>
<td>System configuration, parameter settings, and online operations of GX</td>
<td>(Sold separately)</td>
</tr>
<tr>
<td>Works2, which are common to Simple projects and Structured projects</td>
<td></td>
</tr>
<tr>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
<td>SH-080781ENG (13JU65)</td>
</tr>
<tr>
<td>Operations, such as programming and monitoring in Structured projects, of</td>
<td>(Sold separately)</td>
</tr>
<tr>
<td>GX Works2</td>
<td></td>
</tr>
<tr>
<td>GX Works2 Beginner's Manual (Structured Project)</td>
<td>SH-080788ENG (13JZ23)</td>
</tr>
<tr>
<td>Basic operations, such as programming, editing, and monitoring in Structured</td>
<td>(Sold separately)</td>
</tr>
<tr>
<td>projects, of GX Works2. This manual is intended for first-time users of</td>
<td></td>
</tr>
<tr>
<td>GX Works2.</td>
<td></td>
</tr>
</tbody>
</table>

POINT

Operating manuals in PDF format are stored on the CD-ROM of the software package. Printed manuals are sold separately. To order manuals, please provide the manual number (model code) listed in the table above.
1 OVERVIEW

1.1 Purpose of This Manual ......................................................... 1-2
1.2 Terms ................................................................. 1-5
1.1 Purpose of This Manual

This manual explains the application functions used for creating structured programs. Manuals for reference are listed in the following table according to their purpose.

For information such as the contents and number of each manual, refer to the list of 'Related manuals'.

(1) Operation of GX Works2

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the operating environment and installation method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning a USB driver installation method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation of GX Works2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning all functions of GX Works2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the project types and available languages in GX Works2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the basic operations and operating procedures when creating a simple project for the first time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the basic operations and operating procedures when creating a structured project for the first time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the operations of available functions regardless of project type.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the functions and operation methods for programming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning data setting methods for intelligent function module</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(2) Operations in each programming language
For details of instructions used in each programming language, refer to the section 3 on the next page

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple Project</td>
<td>Structured Project</td>
</tr>
<tr>
<td>Simple Project</td>
<td>Ladder</td>
<td></td>
</tr>
<tr>
<td>SFC</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured Project/FBD</td>
<td>Ladder</td>
<td></td>
</tr>
<tr>
<td>SFC</td>
<td>*1</td>
<td></td>
</tr>
<tr>
<td>Structured ladder/FBD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1: MELSAP3 and FX series SFC only
### Purpose

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fundamentals</td>
<td>Common Instructions</td>
<td>Special Instructions</td>
<td>Application Functions</td>
<td>Common Instruction</td>
<td>PID Control Instructions</td>
</tr>
<tr>
<td>All languages</td>
<td>Learning details of programmable controller CPU error codes, special relays, and special registers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using ladder language</td>
<td>Learning the types and details of common instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for intelligent function modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for network modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for the PID control function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of the process control instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using SFC language</td>
<td>Learning details of specifications, functions, and instructions of SFC (MELSAP3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using structured ladder/FBD/ST language</td>
<td>Learning the fundamentals for creating a structured program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of common instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for intelligent function modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for network modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for the PID control function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of application functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of the process control instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1 Refer to the User’s Manual (Hardware Design, Maintenance and Inspection) for the CPU module used.
1.2 Terms

This manual uses the generic terms and abbreviations listed in the following table to discuss the software packages and programmable controller CPUs. Corresponding module models are also listed if needed.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX Works2</td>
<td>Product name of the software package for the MELSEC programmable controllers</td>
</tr>
<tr>
<td>Basic model QCPU</td>
<td>A generic term for Q00JCPU, Q00CPU, and Q01CPU</td>
</tr>
<tr>
<td>High Performance model QCPU</td>
<td>A generic term for Q02CPU, Q02HCPU, Q06HCPU, Q12HCPU, and Q25HCPU</td>
</tr>
<tr>
<td>Process CPU</td>
<td>A generic term for the Q02PHCPU, Q06PHCPU, Q12PHCPU, and Q25PHCPU</td>
</tr>
<tr>
<td>Redundant CPU</td>
<td>A generic term for the Q12PRHCPU and Q25PRHCPU</td>
</tr>
<tr>
<td>Universal model QCPU</td>
<td>A generic term for Q00UJCPU, Q00UCPU, Q01UCPU, Q02UCPU, Q03UDCPU, Q03UDVCPU, Q03UDECPU, Q04UDHCPU, Q04UDVCPU, Q04UDEHCPU, Q06UDHCPU, Q06UDVCPU, Q06UDEHCPU, Q10UDHCPU, Q10UDEHCPU, Q13UDHCPU, Q13UDEHCPU, Q20UDHCPU, Q20UDEHCPU, Q26UDHCPU, Q26UDEHCPU, Q50UDEHCPU, and Q100UDEHCPU</td>
</tr>
<tr>
<td>QCPU (Q mode)</td>
<td>A generic term for the Basic model QCPU, High Performance model QCPU, Process CPU, Redundant CPU, and Universal model QCPU</td>
</tr>
<tr>
<td>LCPU</td>
<td>A generic term for the L02SCPU, L02SCPU-P, L02CPU, L02CPU-P, L06CPU, L06CPU-P, L26CPU, L26CPU-P, L26CPU-BT, and L26CPU-PBT</td>
</tr>
<tr>
<td>CPU module</td>
<td>A generic term for QCPU (Q mode) and LCPU</td>
</tr>
<tr>
<td>Personal computer</td>
<td>A generic term for personal computer on which Windows® operates</td>
</tr>
<tr>
<td>Common instruction</td>
<td>A generic term for the sequence instructions, basic instructions, application instructions, data link instructions, multiple CPU dedicated instructions, multiple CPU high-speed transmission dedicated instructions, and redundant system instructions</td>
</tr>
<tr>
<td>Special instruction</td>
<td>A generic term for module dedicated instructions, PID control instructions, socket communication function instructions, built-in I/O function instructions, and data logging function instructions</td>
</tr>
<tr>
<td>Application function</td>
<td>A generic term for the functions, such as functions and function blocks, defined in IEC61131-3. (The functions are executed with a set of common instructions in a programmable controller.)</td>
</tr>
</tbody>
</table>
2 FUNCTION TABLES

2.1 How to Read Function Tables ...................................................... 2-2
2.2 Function Tables ................................................................. 2-3
2.1 How to Read Function Tables

Function tables in Section 2.2 are shown in the following format.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_E</td>
<td>σ1, σ2, ..., σ8 (Number of pins variable)</td>
<td>Outputs the sum (σ1 + σ2 + ... + σ8) of input values.</td>
<td>5-135</td>
</tr>
<tr>
<td>MUL_E</td>
<td>σ1, σ2, ..., σ8 (Number of pins variable)</td>
<td>Outputs the product (σ1 × σ2 × ... × σ8) of input values.</td>
<td>5-138</td>
</tr>
<tr>
<td>SUB_E</td>
<td>σ1, σ2, ..., σ8 (Number of pins variable)</td>
<td>Outputs the difference (σ1 - σ2) between input values.</td>
<td>5-141</td>
</tr>
<tr>
<td>DIV_E</td>
<td>σ1, σ2, ..., σ8 (Number of pins variable)</td>
<td>Outputs the quotient (σ1 ÷ σ2) of input values.</td>
<td>5-144</td>
</tr>
</tbody>
</table>

Description

① ........Indicates the functions used in a program. 'Function name(_E)' is used as a function with EN/ENO.

② ........Indicates the arguments of the function.

③ : Source ...................................... Stores data before operation.

④ : Destination ................................... Indicates the destination of data after operation.

(Number of pins variable) .......... Allows the number of ③ (source) to be changed in the range from 2 to 28.

Changing the number of pins

③ ........Indicates the processing details of each function.

④ ........Indicates the pages on which the functions are explained.
# 2.2 Function Tables

## 2.2.1 Type conversion functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_INT(_E)</td>
<td>1, 2</td>
<td>Converts bit type data into word (signed) or double word (signed) type data.</td>
<td>5-2</td>
</tr>
<tr>
<td>BOOL_TO_DINT(_E)</td>
<td>1, 2</td>
<td>Converts bit type data into string type data.</td>
<td>5-5</td>
</tr>
<tr>
<td>BOOL_TO_WORD(_E)</td>
<td>1, 2</td>
<td>Converts bit type data into word (unsigned) or double word (unsigned) type data.</td>
<td>5-7</td>
</tr>
<tr>
<td>BOOL_TO_DWORD(_E)</td>
<td>1, 2</td>
<td>Converts bit type data into time type data.</td>
<td>5-10</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) type data into double word (signed) type data.</td>
<td>5-12</td>
</tr>
<tr>
<td>DINT_TO_INT(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into bit type data.</td>
<td>5-14</td>
</tr>
<tr>
<td>INT_TO_BOOL(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into string type data.</td>
<td>5-16</td>
</tr>
<tr>
<td>DINT_TO_BOOL(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into single-precision real type data.</td>
<td>5-19</td>
</tr>
<tr>
<td>INT_TO_REAL(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into double-precision real type data.</td>
<td>5-22</td>
</tr>
<tr>
<td>DINT_TO_REAL(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into double-precision real type data.</td>
<td>5-25</td>
</tr>
<tr>
<td>INT_TO_LREAL(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into time type data.</td>
<td>5-28</td>
</tr>
<tr>
<td>DINT_TO_LREAL(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into time type data.</td>
<td>5-31</td>
</tr>
<tr>
<td>INT_TO_BCD(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into BCD type data.</td>
<td>5-34</td>
</tr>
<tr>
<td>DINT_TO_BCD(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into BCD type data.</td>
<td>5-37</td>
</tr>
<tr>
<td>INT_TO_TIME(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into time type data.</td>
<td>5-39</td>
</tr>
<tr>
<td>DINT_TO_TIME(_E)</td>
<td>1, 2</td>
<td>Converts word (signed) or double word (signed) type data into time type data.</td>
<td>5-42</td>
</tr>
<tr>
<td>REAL_TO_INT(_E)</td>
<td>1, 2</td>
<td>Converts single-precision real type data into word type data.</td>
<td>5-45</td>
</tr>
<tr>
<td>REAL_TO_DINT(_E)</td>
<td>1, 2</td>
<td>Converts single-precision real type data into double word type data.</td>
<td>5-48</td>
</tr>
<tr>
<td>LREAL_TO_INT(_E)</td>
<td>1, 2</td>
<td>Converts double-precision real type data into word type data.</td>
<td>5-51</td>
</tr>
<tr>
<td>LREAL_TO_DINT(_E)</td>
<td>1, 2</td>
<td>Converts double-precision real type data into double word type data.</td>
<td>5-54</td>
</tr>
<tr>
<td>REAL_TO_BCD(_E)</td>
<td>1, 2</td>
<td>Converts single-precision real type data into BCD type data.</td>
<td>5-57</td>
</tr>
<tr>
<td>LREAL_TO_BCD(_E)</td>
<td>1, 2</td>
<td>Converts double-precision real type data into BCD type data.</td>
<td>5-60</td>
</tr>
<tr>
<td>REAL_TO_LREAL(_E)</td>
<td>1, 2</td>
<td>Converts single-precision real type data into double-precision real type data.</td>
<td>5-63</td>
</tr>
<tr>
<td>LREAL_TO_REAL(_E)</td>
<td>1, 2</td>
<td>Converts double-precision real type data into single-precision real type data.</td>
<td>5-66</td>
</tr>
<tr>
<td>REAL_TO_STR(_E)</td>
<td>1, 2</td>
<td>Converts single-precision real type data into string type data (exponential form).</td>
<td>5-69</td>
</tr>
<tr>
<td>WORD_TO_BOOL(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into bit type data.</td>
<td>5-44</td>
</tr>
<tr>
<td>DWORD_TO_BOOL(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into bit type data.</td>
<td>5-47</td>
</tr>
<tr>
<td>WORD_TO_INT(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into word type data.</td>
<td>5-49</td>
</tr>
<tr>
<td>DWORD_TO_DINT(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into double word type data.</td>
<td>5-52</td>
</tr>
<tr>
<td>WORD_TO_DWORD(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into double word type data.</td>
<td>5-55</td>
</tr>
<tr>
<td>DWORD_TO_DWORD(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into double word type data.</td>
<td>5-58</td>
</tr>
<tr>
<td>WORD_TO_TIME(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into time type data.</td>
<td>5-61</td>
</tr>
<tr>
<td>DWORD_TO_TIME(_E)</td>
<td>1, 2</td>
<td>Converts word (unsigned) or double word (unsigned) type data into time type data.</td>
<td>5-64</td>
</tr>
<tr>
<td>STR_TO_BOOL(_E)</td>
<td>1, 2</td>
<td>Converts string type data into bit type data.</td>
<td>5-67</td>
</tr>
</tbody>
</table>

### 2.2 Function Tables

#### 2.2.1 Type conversion functions
<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR_TO_INT(E)</td>
<td></td>
<td>Converts string type data into word (signed) or double word (signed) type data.</td>
<td>5-81</td>
</tr>
<tr>
<td>STR_TO_DINT(E)</td>
<td></td>
<td></td>
<td>5-84</td>
</tr>
<tr>
<td>STR_TO_REAL(E)</td>
<td></td>
<td>Converts string type data into single-precision real type data.</td>
<td></td>
</tr>
<tr>
<td>STR_TO_WORD(E)</td>
<td></td>
<td>Converts string type data into word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data.</td>
<td>5-88</td>
</tr>
<tr>
<td>STR_TO_DWORD(E)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STR_TO_TIME(E)</td>
<td></td>
<td>Converts string type data into time type data.</td>
<td>5-92</td>
</tr>
<tr>
<td>STR_TO_BCD(E)</td>
<td></td>
<td>Converts string type data into BCD type data.</td>
<td>5-95</td>
</tr>
<tr>
<td>BCD_TO_INT(E)</td>
<td></td>
<td>Converts BCD type data into word (signed) or double word (signed) type data.</td>
<td>5-100</td>
</tr>
<tr>
<td>BCD_TO_DINT(E)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCD_TO_STR(E)</td>
<td></td>
<td>Converts BCD type data into string type data.</td>
<td>5-104</td>
</tr>
<tr>
<td>TIME_TO_BOOL(E)</td>
<td></td>
<td>Converts time type data into bit type data.</td>
<td>5-107</td>
</tr>
<tr>
<td>TIME_TO_INT(E)</td>
<td></td>
<td>Converts time type data into word (signed) or double word (signed) type data.</td>
<td>5-109</td>
</tr>
<tr>
<td>TIME_TO_DINT(E)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME_TO_TIME(E)</td>
<td></td>
<td>Converts time type data into string type data.</td>
<td>5-112</td>
</tr>
<tr>
<td>TIME_TO_WORD(E)</td>
<td></td>
<td>Converts time type data into word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data.</td>
<td>5-114</td>
</tr>
<tr>
<td>TIME_TO_DWORD(E)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BITARR_TO_INT(E)</td>
<td>n, n</td>
<td>Converts specified number of bits from bit array into word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, or double word (unsigned)/32-bit string type data.</td>
<td>5-117</td>
</tr>
<tr>
<td>BITARR_TO_DINT(E)</td>
<td>n, n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT_TO_BITARR(E)</td>
<td>n, n</td>
<td>Outputs low-order n bits of word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, or double word (unsigned)/32-bit string type data.</td>
<td>5-119</td>
</tr>
<tr>
<td>DINT_TO_BITARR(E)</td>
<td>n, n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPY_BITARR(E)</td>
<td>n, n</td>
<td>Copies specified number of bits from bit array.</td>
<td>5-121</td>
</tr>
<tr>
<td>GET_BIT_OF_INT(E)</td>
<td>n, n</td>
<td>Reads a value of specified bit of word (signed) type data.</td>
<td>5-123</td>
</tr>
<tr>
<td>SET_BIT_OF_INT(E)</td>
<td>n, n</td>
<td>Writes a value to the specified bit of word (signed) type data.</td>
<td>5-125</td>
</tr>
<tr>
<td>CPY_BIT_OF_INT(E)</td>
<td>n1, n2, n</td>
<td>Copies a specified bit of word (signed) type data to the specified bit of another word (signed) type data.</td>
<td>5-127</td>
</tr>
<tr>
<td>GET_BOOL_ADDR</td>
<td></td>
<td>Converts the type of data into bit type.</td>
<td>5-129</td>
</tr>
<tr>
<td>GET_INT_ADDR</td>
<td></td>
<td>Converts the type of data into word (signed) type.</td>
<td></td>
</tr>
<tr>
<td>GET_WORD_ADDR</td>
<td></td>
<td>Converts the type of data to word (unsigned)/16-bit string type.</td>
<td></td>
</tr>
</tbody>
</table>
### 2.2.2 Standard functions of one numeric variable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(_E)</td>
<td>@, @</td>
<td>Outputs the absolute value of an input value.</td>
<td>5-131</td>
</tr>
</tbody>
</table>

### 2.2.3 Standard arithmetic functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_E</td>
<td>@, @,..., @</td>
<td>Outputs the sum (∑) of input values.</td>
<td>5-135</td>
</tr>
<tr>
<td>MUL_E</td>
<td>@, @,..., @</td>
<td>Outputs the product (∏) of input values.</td>
<td>5-138</td>
</tr>
<tr>
<td>SUB_E</td>
<td>@, @,..., @</td>
<td>Outputs the difference (−) between input values.</td>
<td>5-141</td>
</tr>
<tr>
<td>DIV_E</td>
<td>@, @,..., @</td>
<td>Outputs the quotient (÷) of input values.</td>
<td>5-144</td>
</tr>
<tr>
<td>MOD(_E)</td>
<td>@, @,..., @</td>
<td>Outputs the remainder after division of input values (mod).</td>
<td>5-147</td>
</tr>
<tr>
<td>EXPT(_E)</td>
<td>@, @,..., @</td>
<td>Outputs the exponentiation of an input value.</td>
<td>5-150</td>
</tr>
<tr>
<td>MOVE(_E)</td>
<td>@, @,..., @</td>
<td>Moves the input value to @.</td>
<td>5-153</td>
</tr>
</tbody>
</table>

### 2.2.4 Standard bitwise Boolean functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND_E</td>
<td>@, @,..., @</td>
<td>Outputs the Boolean AND of input values.</td>
<td>5-157</td>
</tr>
<tr>
<td>OR_E</td>
<td>@, @,..., @</td>
<td>Outputs the Boolean OR of input values.</td>
<td>5-157</td>
</tr>
<tr>
<td>XOR_E</td>
<td>@, @,..., @</td>
<td>Outputs the Boolean exclusive OR of input values.</td>
<td>5-157</td>
</tr>
<tr>
<td>NOT(_E)</td>
<td>@, @,..., @</td>
<td>Outputs the Boolean NOT of input values.</td>
<td>5-157</td>
</tr>
</tbody>
</table>

### 2.2.5 Standard selection functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL(_E)</td>
<td>@, @,..., @</td>
<td>Outputs the value selected from the input values.</td>
<td>5-162</td>
</tr>
<tr>
<td>MAXIMUM(_E)</td>
<td>@, @,..., @</td>
<td>Outputs the maximum value of the input values.</td>
<td>5-165</td>
</tr>
<tr>
<td>MINIMUM(_E)</td>
<td>@, @,..., @</td>
<td>Outputs the minimum value of the input values.</td>
<td>5-168</td>
</tr>
<tr>
<td>LIMITATION(_E)</td>
<td>@, @,..., @</td>
<td>Outputs the input value controlled by the upper and lower limit control.</td>
<td>5-168</td>
</tr>
<tr>
<td>MUX(_E)</td>
<td>n, @,..., @</td>
<td>Outputs one of the multiple input values.</td>
<td>5-171</td>
</tr>
</tbody>
</table>
### 2.2.6 Standard comparison functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT_E</td>
<td>s1, s2, ..., sN (Number of pins variable)</td>
<td>Outputs the comparison value of an input value.</td>
<td>5-174</td>
</tr>
<tr>
<td>GE_E</td>
<td>s1, s2, ..., sN (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQ_E</td>
<td>s1, s2, ..., sN (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LE_E</td>
<td>s1, s2, ..., sN (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT_E</td>
<td>s1, s2, ..., sN (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE_E</td>
<td>s1, s2, sN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2.2.7 Standard character string functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID(E)</td>
<td>s1, n1, n2, s2</td>
<td>Outputs the specified number of characters, extracted from the specified start position in the input character string.</td>
<td>5-178</td>
</tr>
<tr>
<td>CONCAT(E)</td>
<td>s1, s2, s3</td>
<td>Concatenates the character strings and outputs the operation result.</td>
<td>5-181</td>
</tr>
<tr>
<td>INSERT(E)</td>
<td>s1, s2, n, s3 (Number of pins variable)</td>
<td>Inserts a character string between other character strings and outputs the operation result.</td>
<td>5-184</td>
</tr>
<tr>
<td>DELETE(E)</td>
<td>s1, n1, n2, s2</td>
<td>Deletes the specified range in a character string and outputs the operation result.</td>
<td>5-187</td>
</tr>
<tr>
<td>REPLACE(E)</td>
<td>s1, s2, n1, n2, s3</td>
<td>Replaces the specified range in a character string with the specified character string and outputs the operation result.</td>
<td>5-190</td>
</tr>
</tbody>
</table>

### 2.2.8 Functions of time data types

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TIME(E)</td>
<td>s1, s2, s3</td>
<td>Outputs the sum (s1 + s2) of the input values (time type).</td>
<td>5-193</td>
</tr>
<tr>
<td>SUB_TIME(E)</td>
<td>s1, s2, s3</td>
<td>Outputs the difference (s1 - s2) of input values (time type).</td>
<td>5-196</td>
</tr>
<tr>
<td>MUL_TIME(E)</td>
<td>s1, s2, s3</td>
<td>Outputs the product (s1 × s2) of input values (time type).</td>
<td>5-199</td>
</tr>
<tr>
<td>DIV_TIME(E)</td>
<td>s1, s2, s3</td>
<td>Outputs the quotient (s1 ÷ s2) of input values (time type).</td>
<td>5-202</td>
</tr>
</tbody>
</table>

### 2.2.9 Standard bistable function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR(E)</td>
<td>s1, s2, s3</td>
<td>Discerns two input values and outputs 1 (TRUE) or 0 (FALSE). (Set-dominant)</td>
<td>5-204</td>
</tr>
<tr>
<td>RS(E)</td>
<td>s1, s2, s3</td>
<td>Discerns two input values and outputs 1 (TRUE) or 0 (FALSE). (Reset-dominant)</td>
<td>5-207</td>
</tr>
</tbody>
</table>
## 2.2.10 Standard edge detection function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TRIG(_E)</td>
<td>③, ②</td>
<td>Detects the rising edge of a signal and outputs pulse signals.</td>
<td>5-210</td>
</tr>
<tr>
<td>F_TRIG(_E)</td>
<td>③, ②</td>
<td>Detects the falling edge of a signal and outputs pulse signals.</td>
<td>5-213</td>
</tr>
</tbody>
</table>

## 2.2.11 Standard counter function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTU(_E)</td>
<td>③, ②, n, ③, ②</td>
<td>Counts the number of times that the signal turns ON.</td>
<td>5-215</td>
</tr>
<tr>
<td>CTD(_E)</td>
<td>③, ②, n, ③, ②</td>
<td>Counts down the number of times that the signal turns ON.</td>
<td>5-218</td>
</tr>
<tr>
<td>CTUD(_E)</td>
<td>③, ②, n, ③, ②</td>
<td>Counts/counts down the number of times that the signal turns ON.</td>
<td>5-221</td>
</tr>
<tr>
<td>COUNTER_FB_M</td>
<td>③, ②, ③, ②</td>
<td>Counts the number of times that the signal turns ON from ③ to ②.</td>
<td>5-225</td>
</tr>
</tbody>
</table>

## 2.2.12 Standard timer function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP(_E)</td>
<td>③, n, ③, ②</td>
<td>Holds the signal ON for the specified time.</td>
<td>5-227</td>
</tr>
<tr>
<td>TP_HIGH(_E)</td>
<td>③, n, ③, ②</td>
<td>Holds the signal ON for the specified time.</td>
<td>5-227</td>
</tr>
<tr>
<td>TON(_E)</td>
<td>③, n, ③, ②</td>
<td>Turns ON the signal after the specified time.</td>
<td>5-230</td>
</tr>
<tr>
<td>TON_HIGH(_E)</td>
<td>③, n, ③, ②</td>
<td>Turns ON the signal after the specified time.</td>
<td>5-230</td>
</tr>
<tr>
<td>TOF(_E)</td>
<td>③, n, ③, ②</td>
<td>Turns OFF the signal after the specified time.</td>
<td>5-233</td>
</tr>
<tr>
<td>TOF_HIGH(_E)</td>
<td>③, n, ③, ②</td>
<td>Turns OFF the signal after the specified time.</td>
<td>5-233</td>
</tr>
<tr>
<td>TIMER_10_FB_M</td>
<td>③, ③, ③, ②</td>
<td>Turns ON the signal after the specified time counted from input value ③ to ②.</td>
<td>5-236</td>
</tr>
<tr>
<td>TIMER_100_FB_M</td>
<td>③, ③, ③, ②</td>
<td>Turns ON the signal after the specified time counted from input value ③ to ②.</td>
<td>5-236</td>
</tr>
<tr>
<td>TIMER_HIGH_FB_M</td>
<td>③, ③, ③, ②</td>
<td>Turns ON the signal after the specified time counted from input value ③ to ②.</td>
<td>5-236</td>
</tr>
<tr>
<td>TIMER_LOW_FB_M</td>
<td>③, ③, ③, ②</td>
<td>Turns ON the signal after the specified time counted from input value ③ to ②.</td>
<td>5-236</td>
</tr>
<tr>
<td>TIMER_CONT_FB_M</td>
<td>③, ③, ③, ②</td>
<td>Turns ON the signal after the specified time counted from input value ③ to ②.</td>
<td>5-236</td>
</tr>
<tr>
<td>TIMER_CONT_FB_M</td>
<td>③, ③, ③, ②</td>
<td>Turns ON the signal after the specified time counted from input value ③ to ②.</td>
<td>5-236</td>
</tr>
</tbody>
</table>

### POINT

The function and function block of the application functions execute the operation with the combination of multiple sequence instructions. Therefore, if the interrupt occurs in the application function operations, unintended operation results may occur.

For using an interrupt program, use Disable interrupt/ Enable interrupt (DI/EI instruction) as necessary.
## 2.3 Operator Tables

### 2.3.1 Arithmetic operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured ladder/FBD</td>
<td>ST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>+</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td>Outputs the sum ($\theta + \theta + \cdots + \theta$) of input values.</td>
</tr>
<tr>
<td>MUL</td>
<td>*</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td>Outputs the product ($\theta \times \theta \times \cdots \times \theta$) of input values.</td>
</tr>
<tr>
<td>SUB</td>
<td>-</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$</td>
<td>Outputs the difference ($\theta - \theta$) between input values.</td>
</tr>
<tr>
<td>DIV</td>
<td>/</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$</td>
<td>Outputs the quotient ($\theta \div \theta$) of input values.</td>
</tr>
<tr>
<td>MOD</td>
<td>&amp;</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td>Outputs the remainder after division of input values ($\theta \div \theta$).</td>
</tr>
<tr>
<td>**</td>
<td>**</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$</td>
<td>Outputs the exponentiation of an input value.</td>
</tr>
</tbody>
</table>

### 2.3.2 Logical operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured ladder/FBD</td>
<td>ST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td>Outputs the Boolean AND of input values.</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td>Outputs the Boolean OR of input values.</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td>Outputs the Boolean exclusive OR of input values.</td>
</tr>
<tr>
<td>NOT</td>
<td>NOT</td>
<td>$\theta \cdot \theta$</td>
<td>Outputs the Boolean NOT of input values.</td>
</tr>
</tbody>
</table>

### 2.3.3 Comparison operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured ladder/FBD</td>
<td>ST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GT</td>
<td>&gt;</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>&gt;=</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td></td>
</tr>
<tr>
<td>EQ</td>
<td>=</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td>Outputs the comparison value of an input value.</td>
</tr>
<tr>
<td>LE</td>
<td>&lt;=</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td></td>
</tr>
<tr>
<td>LT</td>
<td>&lt;</td>
<td>$\theta \cdot \theta \cdots \theta \cdot \theta$ (Number of pins variable)</td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>&lt;&gt;</td>
<td>$\theta \cdot \theta$</td>
<td></td>
</tr>
</tbody>
</table>
3.1 Configuration of Functions .............................................. 3-2
3.2 Input Pins Variable Function ......................................... 3-3
3.1 Configuration of Functions

Instructions available in the CPU module can be divided into a function name and an argument.

The application of a function name and an argument are as follows:

- Function name ....... Indicates the function.
- Argument .............. Indicates the I/O data used in the function.

Arguments are classified into source data, destination data, executing condition, and execution result.

(1) Source ♦

(a) A source is data used in an operation.

(b) The following source types are available depending on the device specified in a function:

   • Constant ................. Specifies a numeric value used in an operation. Constants are set during programming so that they cannot be changed while the program is being executed. Perform index modification when using them as variable data.

   • Bit device and word device ....... Specifies the device in which the data used in the operation are stored. Data must be stored to the specified device before executing the operation. By changing the data to be stored to the specified device while a program is being executed, the data used in the function can be changed.

(c) Contacts cannot be input directly to sources that use bit devices.

(2) Destination ♦

(a) Data after the operation are stored to a destination.

(b) Set a device in which data are to be stored to a destination.

(c) Coils cannot be directly connected to destinations that store bit devices.

(3) Executing condition (EN)

(a) An input variable EN inputs an executing condition of a function.

(4) Execution result (ENO)

(a) An output variable ENO outputs an execution result.

POINT

For details of the configuration of functions for labels and structures, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
3.2 Input Pins Variable Function

Some functions allow the number of input pins to be changed.

To change the number of input pins, select the target function and change the number.

For the number of input pins change operation GX Works2 Version 1 Operating Manual (Structured Project)
4

HOW TO READ FUNCTIONS
Chapter 5 provides detailed explanation on each function in the layout as shown below.

### 5.1 Type Conversion Functions

1. **Bit type → word (signed), double word (signed) type conversion**
   - BOOL_TO_INT(E), BOOL_TO_DINT(E)

2. **BOOL_TO_INT(E)**

3. **BOOL_TO_DINT(E)**

4. ... (Diagram of function)

5. **Structured ladder (LD)

6. **Function processing**
   - 1) BOOL_TO_INT, BOOL_TO_INT_E:
     - Converts bit type data input to p into word (signed) type data, and outputs the operation result from q.
     - When the input value is FALSE, 0 is output in word (signed) type data.
     - When the input value is TRUE, 1 is output in word (signed) type data.

   ![Function processing diagram]

7. **Operation Error**
   - No operation error occurs in the execution of the BOOL_TO_INT(E) and BOOL_TO_DINT(E) functions.

8. **Program Example**
   - 1) The program which converts bit type data input to p into word (signed) type data, and outputs the operation result from q.

   ![Program example]

---

1. Indicates a section number and an outline of a function.
2. Indicates a function to be explained.
③ Indicates the CPU modules that can use the function.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Basic model QCPU" /></td>
<td>The basic icon indicates that the CPU module can use the corresponding function.</td>
</tr>
<tr>
<td><img src="image" alt="High Performance model QCPU" /></td>
<td>The icon with a Ver. symbol indicates that the CPU module can use the corresponding function under certain restrictions (function version and software version).</td>
</tr>
<tr>
<td><img src="image" alt="Process CPU" /></td>
<td>The icon with <img src="image" alt="Not influencing" /> indicates that the CPU module cannot use the corresponding function.</td>
</tr>
</tbody>
</table>

④ Indicates the function names.

⑤ Indicates the function names that can be described.

⑥ Indicates the description format of the function in the structured ladder/FBD/ST language.

⑦ Indicates the names of input and output arguments, and the data type of each argument. For details of the data type, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

⑧ Indicates the processing performed by the function.

⑨ Indicates whether to exist the related error. When an error exists, conditions that cause an error are described.

⑩ Indicates program examples in the structured ladder/FBD/ST language.
5 APPLICATION FUNCTIONS

5.1 Type Conversion Functions ................................................. 5-2
5.2 Standard Functions of One Numeric Variable ......................... 5-131
5.3 Standard Arithmetic Functions ........................................... 5-135
5.4 Standard Bitwise Boolean Functions .................................. 5-157
5.5 Standard Selection Functions ............................................. 5-162
5.6 Standard Comparison Functions ........................................... 5-174
5.7 Standard Character String Functions ................................... 5-178
5.8 Functions of Time Data Type ............................................. 5-193
5.9 Standard Bistable Function Blocks ..................................... 5-204
5.10 Standard Edge Detection Function Blocks ............................ 5-210
5.11 Standard Counter Function Blocks ..................................... 5-215
5.12 Standard Timer Function Blocks ........................................ 5-227
5.1 Type Conversion Functions

5.1.1 Bit type → word (signed), double word (signed) type conversion

BOOL_TO_INT(_E), BOOL_TO_DINT(_E)

<table>
<thead>
<tr>
<th>Structure</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s</td>
<td>d</td>
</tr>
</tbody>
</table>

Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop) :Bit
Input argument, s(BOOL): Input :Bit
Output argument, ENO: Execution result (TRUE: Normal execution, FALSE: Error or stop) :Bit
Output argument, d: Output :Word (signed), double word (signed)

Function Operation processing

(1) BOOL_TO_INT, BOOL_TO_INT_E

Converts bit type data input to into word (signed) type data, and outputs the operation result from .
When the input value is FALSE, 0 is output in word (signed) type data.
When the input value is TRUE, 1 is output in word (signed) type data.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

_E: With EN/ENO

Indicates any of the following functions.
BOOL_TO_INT BOOL_TO_INT_E
BOOL_TO_DINT BOOL_TO_DINT_E

Structured ladder/FBD

Basic High-performance Process Redundant Universal LCPU
5.1 Type Conversion Functions

5.1.1 Bit type

→ word (signed), double word (signed) type conversion

(2) BOOL_TO_DINT, BOOL_TO_DINT_E

Converts bit type data input into double word (signed) type data, and outputs the operation result from.

When the input value is FALSE, 0 is output in double word (signed) type data.

When the input value is TRUE, 1 is output in double word (signed) type data.

<table>
<thead>
<tr>
<th>Bit type</th>
<th>Double word (signed) type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>0</td>
</tr>
<tr>
<td>TRUE</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from.

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③ Operation output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE¹</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*¹: When FALSE is output from ENO, the data output from ③ is undefined.
In this case, create a program so that the data output from ③ is not used.

Operation Error

No operation error occurs in the execution of the BOOL_TO_INT(_E) and BOOL_TO_DINT(_E) functions.

Program Example

(1) The program which converts bit type data input into word (signed) type data, and outputs the operation result from.

(a) Function without EN/ENO (BOOL_TO_INT)

[Structured ladder/FBD]

```
[ST]
g_int1 := BOOL_TO_INT(g_bool1);
```

5.1 Type Conversion Functions

5.1.1 Bit type \( \rightarrow \) word (signed), double word (signed) type conversion

(b) Function with EN/ENO (BOOL_TO_INT_E)

[Structured ladder/FBD]

```
[ST]
g_boole3 := BOOL_TO_INT_E(g_bool1, g_bool2, g_int1);
```

(2) The program which converts bit type data input to \( \circ \) into double word (signed) type data, and outputs the operation result from \( \circ \).

(a) Function without EN/ENO (BOOL_TO_DINT)

[Structured ladder/FBD]

```
[ST]
g_dint1 := BOOL_TO_DINT(g_bool1);
```
5.1.2 Bit type → string type conversion

**BOOL_TO_STR(E)**

_E: With EN/ENO

---

**Function**

**Operation processing**

Converts bit type data input to ① into string type data, and outputs the operation result from ②.

When the input value is FALSE, 0 is output in string type data.
When the input value is TRUE, 1 is output in string type data.

<table>
<thead>
<tr>
<th>Bit type</th>
<th>String type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>TRUE</td>
<td>&quot;1&quot;</td>
</tr>
</tbody>
</table>
**Operation result**

(1) Function without EN/ENO
An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.

**Operation Error**
No operation error occurs in the execution of the BOOL_TO_STR(E) function.

**Program Example**

The program which converts bit type data input to ③ into string type data, and outputs the operation result from ③.

(a) Function without EN/ENO (BOOL_TO_STR)

[Structured ladder/FBD]

![Structured ladder/FBD diagram]

[ST]

g_string1 := BOOL_TO_STR(g_bool1);
5.1.3 Bit type → word (unsigned)/16-bit string, double word (unsigned)/32-bit string type conversion

BOOL_TO_WORD(_E), BOOL_TO_DWORD(_E)

_E: With EN/ENO

**Operation processing**

1. **BOOL_TO_WORD, BOOL_TO_WORD_E**

   Converts bit type data input to [②] into word (unsigned)/16-bit string type data, and outputs the operation result from [③].
   When the input value is FALSE, 0H is output in word (unsigned)/16-bit string type data.
   When the input value is TRUE, 1H is output in word (unsigned)/16-bit string type data.

   ![Conversion Diagram]
(2) BOOL_TO_DWORD_E, BOOL_TO_DWORD_E

Converts bit type data input to ① into double word (unsigned)/32-bit string type data, and outputs the operation result from ②.

When the input value is FALSE, 0H is output in double word (unsigned)/32-bit string type data.
When the input value is TRUE, 1H is output in double word (unsigned)/32-bit string type data.

<table>
<thead>
<tr>
<th>①</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>0H</td>
</tr>
<tr>
<td>TRUE</td>
<td>1H</td>
</tr>
</tbody>
</table>

Bit type Double word (unsigned)/32-bit string type

Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from ②.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ② is undefined. In this case, create a program so that the data output from ② is not used.
### Operation Error

No operation error occurs in the execution of the BOOLTOWORD(E) and BOOLTODWORD(E) functions.

### Program Example

1. The program which converts bit type data input to into word (unsigned)/16-bit string type data, and outputs the operation result from .
   
   (a) Function without EN/ENO (BOOLTOWORD)
   
   **[Structured ladder/FBD]**

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

   (b) Function with EN/ENO (BOOLTOWORD_E)
   
   **[Structured ladder/FBD]**

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

2. The program which converts bit type data input to into double word (unsigned)/32-bit string type data, and outputs the operation result from .
   
   (a) Function without EN/ENO (BOOLTODWORD)
   
   **[Structured ladder/FBD]**

   ```
   g_dword1 := BOOLTODWORD (g_bool1);
   ```
5.1.4 Bit type → time type conversion

**BOOL_TO_TIME(_E)**

Function

**Operation processing**

Converts bit type data input into time type data, and outputs the operation result from.

- When the input value is FALSE, 0 is output in time type data.
- When the input value is TRUE, 1 is output in time type data.
**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from $\odot$.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>$\odot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE$^1$</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

$^1$: When FALSE is output from ENO, the data output from $\odot$ is undefined. In this case, create a program so that the data output from $\odot$ is not used.

**Operation Error**

No operation error occurs in the execution of the BOOL_TO_TIME(_E) function.

**Program Example**

The program which converts bit type data input to $\odot$ into time type data, and outputs the operation result from $\odot$.

(a) Function without EN/ENO (BOOL_TO_TIME)

[Structured ladder/FBD]

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

(b) Function with EN/ENO (BOOL_TO_TIME_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := BOOL_TO_TIME_E (g_bool1, g_bool2, g_time1);
```
5.1.5 Word (signed) type → double word (signed) type conversion

**INT_TO_DINT(_E)**

_E: With EN/ENO

---

**Structured ladder/FBD**

```
INT_TO_DINT_E (EN, s, d)
```

**Input argument,**

- **EN:** Executing condition (TRUE: Execution, FALSE: Stop)
- **s(_INT):** Input

**Output argument,**

- **ENO:** Execution result (TRUE: Normal, FALSE: Error)
- **d:** Output

---

**Function**

**Operation processing**

Converts word (signed) type data input to ③ into double word (signed) type data, and outputs the operation result from ②.

③ 1234 → ② 1234

Word (signed) type

Double word (signed) type
5.1 Type Conversion Functions

5.1.5 Word (signed) type → double word (signed) type conversion

**Operation result**

1. **Function without EN/ENO**
   
   An operation is executed and the operation value is output from ③.

2. **Function with EN/ENO**
   
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE¹</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*¹: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.

**Operation Error**

No operation error occurs in the execution of the INT_TO_DINT(_E) function.

**Program Example**

The program which converts word (signed) type data input to ③ into double word (signed) type data, and outputs the operation result from ③.

(a) **Function without EN/ENO (INT_TO_DINT)**

[Structured ladder/FBD]

(ST)

\[ g\_{\text{dint1}} := \text{INT\_TO\_DINT}\; (g\_{\text{int1}}); \]

(b) **Function with EN/ENO (INT_TO_DINT_E)**

[Structured ladder/FBD]

(ST)

\[ g\_{\text{bool3}} := \text{INT\_TO\_DINT\_E}\; (g\_{\text{bool1}}, g\_{\text{int1}}, g\_{\text{dint1}}); \]
5.1.6  Double word (signed) type → word (signed) type conversion

**DINT_TO_INT**(E)

_E:_ With EN/ENO

---

**Function**

**Operation processing**

Converts double word (signed) type data input to ② into word (signed) type data, and outputs the operation result from ③.

1. Double word (signed) type
2. Word (signed) type
5.1 Type Conversion Functions

5.1.6 Double word (signed) type → word (signed) type conversion

Operation result

1. Function without EN/ENO
   An operation is executed and the operation value is output from \(\circ\).

2. Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>(\circ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \(\circ\) is undefined. In this case, create a program so that the data output from \(\circ\) is not used.

POINT

When the DINT_TO_INT(_E) function is executed, low-order 16-bit data of double word (signed) type data input to \(\circ\) are converted into word (signed) type data. High-order 16-bit data are discarded.

Operation Error

No operation error occurs in the execution of the DINT_TO_INT(_E) function.

Program Example

The program which converts double word (signed) type data input to \(\circ\) into word (signed) type data, and outputs the operation result from \(\circ\).

(a) Function without EN/ENO (DINT_TO_INT)

[Structured ladder/FBD]

```
[ST]
g_int1 := DINT_TO_INT(g_dint1);
```

(b) Function with EN/ENO (DINT_TO_INT_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := DINT_TO_INT_E (g_bool1, g_dint1, g_int1);
```
5.1.7 Word (signed), double word (signed) type → bit type conversion

**INT_TO_BOOL(_E), DINT_TO_BOOL(_E)**

_E: With EN/ENO

### Function

#### Operation processing

1. **INT_TO_BOOL, INT_TO_BOOL_E**

Converts word (signed) type data input to (2) into bit type data, and outputs the operation result from (3).

When the input value is 0, FALSE is output in bit type data.
When the input value is other than 0, TRUE is output in bit type data.

<table>
<thead>
<tr>
<th>0</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1567</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
5.1 Type Conversion Functions

5.1.7 Word (signed), double word (signed) type → bit type conversion

(2) DINT_TO_BOOL, DINT_TO_BOOL_E

Converts double word (signed) type data input to ② into bit type data, and outputs the operation result from ③.

When the input value is 0, FALSE is output in bit type data.
When the input value is other than 0, TRUE is output in bit type data.

<table>
<thead>
<tr>
<th>②</th>
<th>12345678</th>
</tr>
</thead>
<tbody>
<tr>
<td>③</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Double word (signed) type

Bit type

Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined.
In this case, create a program so that the data output from ③ is not used.

Operation Error

No operation error occurs in the execution of the INT_TO_BOOL(E) and DINT_TO_BOOL(E) functions.

Program Example

(1) The program which converts word (signed) type data input to ② into bit type data, and outputs the operation result from ③.

(a) Function without EN/ENO (INT_TO_BOOL)

[Structured ladder/FBD]

```
1
q_int := INT_TO_BOOL(g_int1);
```

[ST]

```
g Bool1 := INT_TO_BOOL(g_int1);
```
(b) Function with EN/ENO (INT_TO_BOOL_E)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool3}} := \text{INT\_TO\_BOOL\_E}\left(g_{\text{bool1}}, g_{\text{int1}}, g_{\text{bool2}}\right);
\end{array}
\]

(2) The program which converts double word (signed) type data input to \( \odot \) into bit type data, and outputs the operation result from \( \odot \).

(a) Function without EN/ENO (DINT_TO_BOOL)

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool1}} := \text{DINT\_TO\_BOOL}\left(g_{\text{dint1}}\right);
\end{array}
\]
5.1.8 Word (signed), double word (signed) type → single-precision real type conversion

INT_TO_REAL(E), DINT_TO_REAL(E)

 Function

Operation processing

(1) INT_TO_REAL, INT_TO_REAL_E

Converts word (signed) type data input to ② into single-precision real type data, and outputs the operation result from ③.

1234  ↔  1234.0

Word (signed) type  Single-precision real type
(2) DINT_TO_REAL, DINT_TO_REAL_E

(a) Converts double word (signed) type data input to  into single-precision real type data, and outputs the operation result from  .

<table>
<thead>
<tr>
<th>16543521</th>
<th>16543521.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double word (signed) type</td>
<td>Single-precision real type</td>
</tr>
</tbody>
</table>

(b) The number of significant figures of single-precision real type 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.

Operation result

(1) Function without EN/ENO
   An operation is executed and the operation value is output from  .

(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE ¹</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*¹: When FALSE is output from ENO, the data output from  is undefined. In this case, create a program so that the data output from  is not used.
5.1 Type Conversion Functions

5.1.8 Word (signed), double word (signed) type → single-precision real type conversion

Operation Error

No operation error occurs in the execution of the INT_TO_REAL(_E) and DINT_TO_REAL(_E) functions.

Program Example

(1) The program which converts word (signed) type data input to \( \circ \) into single-precision real type data, and outputs the operation result from \( \circ \).

(a) Function without EN/ENO (INT_TO_REAL)

[Structured ladder/FBD]

\[
\text{g_real1} := \text{INT_TO_REAL}(\text{g_int1});
\]

(b) Function with EN/ENO (INT_TO_REAL_E)

[Structured ladder/FBD]

\[
\text{g_bool3} := \text{INT_TO_REAL_E}(\text{g_bool1}, \text{g_int1}, \text{g_real1});
\]

(2) The program which converts double word (signed) type data input to \( \circ \) into single-precision real type data, and outputs the operation result from \( \circ \).

(a) Function without EN/ENO (DINT_TO_REAL)

[Structured ladder/FBD]

\[
\text{g_real1} := \text{DINT_TO_REAL}(\text{g_dint1});
\]
5.1.9 Word (signed), double word (signed) type → double-precision real type conversion

**Function**

**Operation processing**

1. **INT_TO_LREAL, INT_TO_LREAL_E**

   Converts word (signed) type data input to ③ into double-precision real type data, and outputs the operation result from ④.

   ![Structured ladder/FBD](image)

   - **EN**: Executing condition (TRUE: Execution, FALSE: Stop) :Bit
   - **ENO**: Execution result (TRUE: Normal, FALSE: Error) :Bit
   - **s**: Input :Word (signed), double word (signed)
   - **d**: Output :Double-precision real

   - **_E**: With EN/ENO

   ```
   INT_TO_LREAL(s); ENO:= INT_TO_LREAL_E(s, d);
   ```

   Indicates any of the following functions.
   - INT_TO_LREAL
   - INT_TO_LREAL_E
   - DINT_TO_LREAL
   - DINT_TO_LREAL_E

---

5-22

5.1 Type Conversion Functions

5.1.9 Word (signed), double word (signed) type → double-precision real type conversion
(2) DINT_TO_LREAL, DINT_TO_LREAL_E

Converts double word (signed) type data input to ① into double-precision real type data, and outputs the operation result from ③.


double word (signed) type

<table>
<thead>
<tr>
<th>①</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>16543521</td>
<td>16543521.0</td>
</tr>
</tbody>
</table>

Double-precision real type

---

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN (Operation execution)</th>
<th>ENO (Operation stop)</th>
<th>③ Operation output value</th>
<th>③ Undefined value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Operation output value</td>
<td>Undefined value</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE*1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
5.1 Type Conversion Functions

5.1.9 Word (signed), double word (signed) type \(\rightarrow\) double-precision real type conversion

**Operation Error**

No operation error occurs in the execution of the INT_TO_LREAL(_E) and DINT_TO_LREAL(_E) functions.

**Program Example**

1. The program which converts word (signed) type data input to \(\oplus\) into double-precision real type data, and outputs the operation result from \(\oplus\).
   (a) Function without EN/ENO (INT_TO_LREAL)

   [Structured ladder/FBD]

   ```plaintext
   g_lreal1 := INT_TO_LREAL(g_int1);
   ```

   [ST]

   ```plaintext
   g_lreal1 := INT_TO_LREAL(g_int1);
   ```

   (b) Function with EN/ENO (INT_TO_LREAL_E)

   [Structured ladder/FBD]

   ```plaintext
   g_bool3 := INT_TO_LREAL_E(g_bool1, g_int1, g_lreal2);
   ```

   [ST]

   ```plaintext
   g_bool3 := INT_TO_LREAL_E(g_bool1, g_int1, g_lreal2);
   ```

2. The program which converts double word (signed) type data input to \(\oplus\) into double-precision real type data, and outputs the operation result from \(\oplus\).
   (a) Function without EN/ENO (DINT_TO_LREAL)

   [Structured ladder/FBD]

   ```plaintext
   g_lreal1 := DINT_TO_LREAL(g_dint1);
   ```

   [ST]

   ```plaintext
   g_lreal1 := DINT_TO_LREAL(g_dint1);
   ```
5.1.10 Word (signed), double word (signed) type → string type conversion

**INT_TO_STR(_E), DINT_TO_STR(_E)**

_E: With EN/ENO

**Structured ladder/FBD**

```
EN  ENO
s   d
```

**ST**

```
ENO := INT_TO_STR_E (EN, s, d);
```

**Operation processing**

1. **INT_TO_STR, INT_TO_STR_E**
   
   a. Converts word (signed) type data input to s into string type data, and outputs the operation result from d.

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

   When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored.

   b. '20H (space)' is stored in 'Sign data' when the input value is positive; '2DH (-)' is stored when negative.
### 5.1.10 Word (signed), double word (signed) type → string type conversion

#### (c) If the number of significant figures is less, '20H (space)' is stored to high-order digits.

(Example) Inputting −123

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>String 1st word</th>
</tr>
</thead>
<tbody>
<tr>
<td>20H (space)</td>
<td>2D (−)</td>
<td>2nd word</td>
</tr>
<tr>
<td>31H (1)</td>
<td>20H (space)</td>
<td>3rd word</td>
</tr>
<tr>
<td>33H (3)</td>
<td>32H (2)</td>
<td>4th word</td>
</tr>
<tr>
<td>00H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Word (signed) type

#### (d) When SM701 (signal for switching the number of output characters) is OFF, "00" is stored to the end of the character string.

(2) DINT_TO_STR, DINT_TO_STR_E

(a) Converts double word (signed) type data input to  into string type data, and outputs the operation result from  .

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>String 1st word</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of billions place</td>
<td>Sign data</td>
<td>2nd word</td>
</tr>
<tr>
<td>ASCII code of ten-millions place</td>
<td>ASCII code of hundred-millions place</td>
<td>3rd word</td>
</tr>
<tr>
<td>ASCII code of hundreds place</td>
<td>ASCII code of millions place</td>
<td>4th word</td>
</tr>
<tr>
<td>ASCII code of thousands place</td>
<td>ASCII code of ten-thousands place</td>
<td>5th word</td>
</tr>
<tr>
<td>ASCII code of tens place</td>
<td>ASCII code of hundreds place</td>
<td>6th word</td>
</tr>
<tr>
<td>00H</td>
<td>ASCII code of units place</td>
<td></td>
</tr>
</tbody>
</table>

Double word (signed) type

When SM701 (signal for switching the number of output characters) is OFF, "00" is stored.

(b) '20H (space)' is stored in 'Sign data' when the input value is positive; '2DH (-)' is stored when negative

(c) If the number of significant figures is less, '20H (space)' is stored to high-order digits.

(Example) Inputting −123456

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>String 1st word</th>
</tr>
</thead>
<tbody>
<tr>
<td>20H (space)</td>
<td>2D (−)</td>
<td>2nd word</td>
</tr>
<tr>
<td>20H (space)</td>
<td>20H (space)</td>
<td>3rd word</td>
</tr>
<tr>
<td>31H (1)</td>
<td>20H (space)</td>
<td>4th word</td>
</tr>
<tr>
<td>33H (3)</td>
<td>32H (2)</td>
<td>5th word</td>
</tr>
<tr>
<td>35H (5)</td>
<td>34H (4)</td>
<td>6th word</td>
</tr>
<tr>
<td>00H</td>
<td>36H (6)</td>
<td></td>
</tr>
</tbody>
</table>

Double word (signed) type

(d) When SM701 (signal for switching the number of output characters) is OFF, "00" is stored to the end of the character string.
Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \circ \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE'1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \circ \) is undefined. In this case, create a program so that the data output from \( \circ \) is not used.

Operation Error

No operation error occurs in the execution of the \text{INT\_TO\_STR\(_E\)} and \text{DINT\_TO\_STR\(_E\)} functions.

Program Example

(1) The program which converts word (signed) type data input to \( \circ \) into string type data, and outputs the operation result from \( \circ \).

(a) Function without EN/ENO (INT\_TO\_STR)

[Structured ladder/FBD]

```
1
```

```
2
```

[ST]

\[
g\text{\_string1} := \text{INT\_TO\_STR}(g\text{\_int1});
\]

(b) Function with EN/ENO (INT\_TO\_STR\(_E\))

[Structured ladder/FBD]

```
1
```

```
2
```

[ST]

\[
g\text{\_bool3} := \text{INT\_TO\_STR\(_E\)} (g\text{\_bool1}, g\text{\_int1}, g\text{\_string1});
\]
(2) The program which converts double word (signed) type data input to \( \odot \) into string type data, and outputs the operation result from \( \odot \)

(a) Function without EN/ENO (DINT_TO_STR)

[Structured ladder/FBD]

```
3
\[ g_{\text{dint1}} := \text{DINT\_TO\_STR}(g_{\text{dint1}}); \]
```

[ST]
5.1.11 Word (signed), double word (signed) type → word (unsigned)/16-bit string type conversion

INT_TO_WORD(_E), DINT_TO_WORD(_E)

Function

Operation processing

(1) INT_TO_WORD, INT_TO_WORD_E

Converts word (signed) type data input to ① into word (unsigned)/16-bit string type data, and outputs the operation result from ③.

```
① 22136
   Word (signed) type
```

```
③ 5678h
   Word (unsigned)/16-bit string type
```
(2) **DINT_TO_WORD, DINT_TO_WORD_E**

Converts double word (signed) type data input to \( \mathcal{Q} \) into word (unsigned)/16-bit string type data, and outputs the operation result from \( \mathcal{R} \).

\[
\begin{array}{c}
\mathcal{Q} \\
12345678
\end{array}
\rightarrow
\begin{array}{c}
\mathcal{R} \\
614EH
\end{array}
\]

Double word (signed) type → Word (unsigned)/16-bit string type

12345678

\[
\begin{array}{c}
0000000111110000010101011001
\end{array}
\]

High-order 16-bit data is discarded.

**Operation result**

1. **Function without EN/ENO**
   
   An operation is executed and the operation value is output from \( \mathcal{R} \).

2. **Function with EN/ENO**
   
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \mathcal{R} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \mathcal{R} \) is undefined. In this case, create a program so that the data output from \( \mathcal{R} \) is not used.

**POINT**

When the DINT_TO_WORD(E) function is executed, low-order 16-bit data of double word (signed) type data input to input variable \( \mathcal{Q} \) are converted into word (unsigned)/16-bit string type data. High-order word (unsigned)/16-bit string type data are discarded.
### 5.1 Type Conversion Functions

#### 5.1.11 Word (signed), double word (signed) type \(\rightarrow\) word (unsigned)/16-bit string type conversion

---

#### Operation Error

No operation error occurs in the execution of the INT_TO_WORD(E) and DINT_TO_WORD(E) functions.

#### Program Example

(1) The program which converts word (signed) type data input to \(\oplus\) into word (unsigned)/16-bit string type data, and outputs the operation result from \(\odot\).

(a) Function without EN/ENO (INT_TO_WORD)

[Structured ladder/FBD]

```
[ST]
g_word1 := INT_TO_WORD(g_int1);
```

(b) Function with EN/ENO (INT_TO_WORD_E)

[Structured ladder/FBD]

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

(2) The program which converts double word (signed) type data input to \(\oplus\) into word (unsigned)/16-bit string type data, and outputs the operation result from \(\odot\).

(a) Function without EN/ENO (DINT_TO_WORD)

[Structured ladder/FBD]

```
[ST]
g_word1 := DINT_TO_WORD(g_dint1);
```
5.1.12 Word (signed), double word (signed) type → double word (unsigned)/32-bit string type conversion

**INT_TO_DWORD(_E), DINT_TO_DWORD(_E)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INT_TO_DWORD, INT_TO_DWORD_E</strong></td>
<td>Converts word (signed) type data input into double word (unsigned)/32-bit string type data, and outputs the operation result from .</td>
</tr>
</tbody>
</table>

```
-325 ➔ 0000FEBH
```

```
-325 1111111010111011
1000000000000000 1111110101110110000FE
```

Data conversion

```
Always filled with 0's.
```

Structured ladder/FBD

```
IN INT_TO_DWORD_E
EN ENO
s d
```

ST

```
ENO:=INT_TO_DWORD_E[EN, s, d];
```

<table>
<thead>
<tr>
<th>Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop)</th>
<th>:Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_INT, _DINT: Input</td>
<td>:Word (signed), double word (signed)</td>
</tr>
<tr>
<td>Output argument, ENO: Execution result (TRUE: Normal, FALSE: Error)</td>
<td>:Bit</td>
</tr>
<tr>
<td>d: Output</td>
<td>:Double word (unsigned)/32-bit string</td>
</tr>
</tbody>
</table>
(2) **DINT_TO_DWORD, DINT_TO_DWORD_E**

Converts double word (signed) type data input to $①$ into double word (unsigned)/32-bit string type data, and outputs the operation result from $②$.  

![Diagram of data conversion](image)

**Double word (signed) type** → **Double word (unsigned)/32-bit string type**

---

**Operation result**

(1) **Function without EN/ENO**  
An operation is executed and the operation value is output from $②$.  

(2) **Function with EN/ENO**  
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>$②$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE$^1$</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from $②$ is undefined. In this case, create a program so that the data output from $②$ is not used.
5.1 Type Conversion Functions

5.1.12 Word (signed), double word (signed) type → double word (unsigned)/32-bit string type conversion

### Operation Error

No operation error occurs in the execution of the INT_TO_DWORD(_E) and DINT_TO_DWORD(_E) functions.

### Program Example

(1) The program which converts word (signed) type data input to ⨁ into double word (unsigned)/32-bit string type data, and outputs the operation result from ⨁.

(a) Function without EN/ENO (INT_TO_DWORD)

```
[g_int1] INT_TO_DWORD(g_int1)

g_dword1 := INT_TO_DWORD(g_int1);
```

(b) Function with EN/ENO (INT_TO_DWORD_E)

```
[g_bool1, g_int1] INT_TO_DWORD_E(g_bool1, g_int1, g_dword1)

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


(2) The program which converts double word (signed) type data input to ⨂ into double word (unsigned)/32-bit string type data, and outputs the operation result from ⨃.

(a) Function without EN/ENO (DINT_TO_DWORD)

```
[g_dint1] DINT_TO_DWORD(g_dint1)

g_dword1 := DINT_TO_DWORD(g_dint1);
```
5.1.13 Word (signed), double word (signed) type → BCD type conversion

INT_TO_BCD(_E), DINT_TO_BCD(_E)

Function Operation processing

(1) INT_TO_BCD, INT_TO_BCD_E

(a) Converts word (signed) type data input to  into BCD type data, and outputs the operation result from .

(b) The value to be input to  is word (signed) type data within the range from 0 to 9999.
5.1 Type Conversion Functions

5.1.13 Word (signed), double word (signed) type \(\rightarrow\) BCD type conversion

(2) DINT_TO_BCD, DINT_TO_BCD_E

(a) Converts double word (signed) type data input to \(\textcircled{3}\) into BCD type data, and outputs the operation result from \(\textcircled{4}\).

\[
\begin{array}{c|c}
\text{Double word (signed) type} & \text{Double word (unsigned)/32-bit string type} \\
\hline
\text{99999999} & \text{99999999h} \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
\text{Ten} & \text{Millions} & \text{Place} & \text{Tens} & \text{Hundreds} & \text{Place} & \text{Thousands} & \text{Place} & \text{Units} & \text{Place} \\
\hline
\text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} \\
\text{BCD} & \text{conversion} \\
\end{array}
\]

(b) The value to be input to \(\textcircled{3}\) is double word (signed) type data within the range from 0 to 99999999.

(c) Word (unsigned)/16-bit string type, double word (unsigned)/32-bit string type data can be specified for \(\textcircled{3}\). Bit type cannot be specified.

\[\text{POINT}\]

The output from \(\textcircled{4}\) cannot be used with connecting to the input of double word (unsigned)/32-bit string type data. In this case, use the DBCD instruction.

---

**Operation result**

(1) Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(\textcircled{4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>(\textcircled{4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \(\textcircled{4}\) is undefined. In this case, create a program so that the data output from \(\textcircled{4}\) is not used.
Operation Error

An operation error occurs when the value input exceeds 9999 or 99999999 respectively in the execution of the INT_TO_BCD(_E) or DINT_TO_BCD(_E) function. (Error code: 4100)

Program Example

(1) The program which converts word (signed) type data input to \(\circ\) into BCD type data, and outputs the operation result from \(\circ\).
   (a) Function without EN/ENO (INT_TO_BCD)
      [Structured ladder/FBD]
      
      \[
      \begin{align*}
      \text{[ST]} \\
      \text{g_word1 := INT_TO_BCD(g_int1)};
      \end{align*}
      \]
   (b) Function with EN/ENO (INT_TO_BCD_E)
      [Structured ladder/FBD]
      
      \[
      \begin{align*}
      \text{[ST]} \\
      \text{g_bool3 := INT_TO_BCD_E(g_bool1, g_int1, g_word1)};
      \end{align*}
      \]

(2) The program which converts double word (signed) type data input to \(\circ\) into BCD type data, and outputs the operation result from \(\circ\).
   (a) Function without EN/ENO (DINT_TO_BCD)
      [Structured ladder/FBD]
      
      \[
      \begin{align*}
      \text{[ST]} \\
      \text{g_dword1 := DINT_TO_BCD(g_dint1)};
      \end{align*}
      \]
5.1.14 Word (signed), double word (signed) type → time type conversion

**INT_TO_TIME(_E), DINT_TO_TIME(_E)**

\[ \text{INT_TO_TIME(_E), DINT_TO_TIME(_E)} \]

\[ _E: \text{With EN/ENO} \]

**Function**

**Operation processing**

Converts word (signed) / double word (signed) type data input to ① into time type data, and outputs the operation result from ③.

\[ 1234 \rightarrow \text{T#1s234ms} \]

Word (signed) type

Time type
5.1 Type Conversion Functions

5.1.14 Word (signed), double word (signed) type → time type conversion

Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \circ \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*2</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*2: When FALSE is output from ENO, the data output from \( \circ \) is undefined. In this case, create a program so that the data output from \( \circ \) is not used.

Operation Error

No operation error occurs in the execution of the INT_TO_TIME(_E) and DINT_TO_TIME(_E) functions.

Program Example

(1) The program which converts word (signed) type data input to \( \circ \) into time type data, and outputs the operation result from \( \circ \).

(a) Function without EN/ENO (INT_TO_TIME)

[Structured ladder/FBD]

```
[ST]
g_time1 := INT_TO_TIME(g_int1);
```

(b) Function with EN/ENO (INT_TO_TIME_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := INT_TO_TIME_E(g_bool1, g_int1, g_time1);
```
(2) The program which converts double word (signed) type data input to \( \circ \) into time type data, and outputs the operation result from \( \odot \).

(a) Function without EN/ENO (DINT_TO_TIME)

[Structured ladder/FBD]

\[
\text{g\_time1} := \text{DINT\_TO\_TIME(g\_dint1)};
\]

[ST]
5.1.15 Single-precision real type \(\rightarrow\) word (signed), double word (signed) type conversion

REAL_TO_INT(_E), REAL_TO_DINT(_E)

\[ s \rightarrow d \]

\_E: With EN/ENO

Function

Operation processing

(1) REAL_TO_INT, REAL_TO_INT_E

(a) Converts single-precision real type data input to \(s\) into word (signed) type data, and outputs the operation result from \(d\).

\[
\begin{align*}
&1234.0 \\
&\text{Single-precision real type}
\end{align*}
\]

\[
\begin{align*}
&1234 \\
&\text{Word (signed) type}
\end{align*}
\]

(b) The value to be input to \(s\) is single-precision real type data, within the range from \(-32768\) to \(32767\).

(c) The converted data is the value rounded single-precision real type data to the first digit after the decimal point.
(2) REAL_TO_DINT, REAL_TO_DINT_E

(a) Converts single-precision real type data input to  into double word (signed) type data, and outputs the operation result from  .

\[
\begin{array}{c|c}
\text{Single-precision real type} & \text{Double word (signed) type} \\
16543521.0 & 16543521 \\
\end{array}
\]

(b) The value to be input to  is single-precision real type data within the range from \(-2^{31}\) to \(2^{31}\). However, a rounding error may occur when setting the input value by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

(c) The converted data is the value rounded single-precision real type data to the first digit after the decimal point.

**Operation result**

(1) Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^*1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from  is undefined. In this case, create a program so that the data output from  is not used.
5.1 Type Conversion Functions

5.1.15 Single-precision real type \rightarrow word (signed), double word (signed) type conversion

Operation Error

An operation error occurs in the following cases.

- REAL_TO_INT(_E): The input value is outside the range of \(-32768\) to \(32767\).
  (Error code: 4100)

- REAL_TO_DINT(_E): The input value is outside the range of \(-2147483648\) to \(2147483647\).
  (Error code: 4100)

Program Example

1. The program which converts single-precision real type data input to \(\odot\) into word (signed) type data, and outputs the operation result from \(\oplus\).
   
   (a) Function without EN/ENO (REAL_TO_INT)
   [Structured ladder/FBD]

   \[
   \text{g\_int1} := \text{REAL\_TO\_INT}(\text{g\_real1});
   \]

   (b) Function with EN/ENO (REAL_TO_INT_E)
   [Structured ladder/FBD]

   \[
   \text{g\_bool3} := \text{REAL\_TO\_INT\_E}(\text{g\_bool1}, \text{g\_real1}, \text{g\_int1});
   \]

2. The program which converts single-precision real type data input to \(\odot\) into double word (signed) type data, and outputs the operation result from \(\oplus\).
   
   (a) Function without EN/ENO (REAL_TO_DINT)
   [Structured ladder/FBD]

   \[
   \text{g\_dint1} := \text{REAL\_TO\_DINT}(\text{g\_real1});
   \]
5.1.16 Double-precision real type → word (signed), double word (signed) type conversion

LREAL_TO_INT(_E), LREAL_TO_DINT(_E)

Function

Operation processing

(1) LREAL_TO_INT, LREAL_TO_INT_E

  (a) Converts double-precision real type data input to ② into word (signed) type data, and outputs the operation result from ③.

  ![Structured ladder/FBD diagram]

  1234.0

  Double-precision real type

  1234

  Word (signed) type

  (b) The value to be input to ② is double-precision real type data, within the range from -32768 to 32767.

  (c) The converted data is the value rounded double-precision real type data to the first digit after the decimal point.
5.1 Type Conversion Functions

5.1.16 Double-precision real type → word (signed), double word (signed) type conversion

(2) LREAL_TO_DINT, LREAL_TO_DINT_E

(a) Converts double-precision real type data input to \( \circ \) into double word (signed) type data, and outputs the operation result from \( \circ \).

\[
\begin{array}{c|c}
\text{Double-precision real type} & \text{Double word (signed) type} \\
16543521.0 & 16543521 \\
\end{array}
\]

(b) The value to be input to \( \circ \) is double-precision real type data within the range from \(-2147483648\) to \(2147483647\). However, rounding error may occur when setting the input value by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

(c) The converted data is the value rounded double-precision real type data to the first digit after the decimal point.

Operation result

(1) Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>( \circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN (Operation execution)</th>
<th>ENO (No operation error)</th>
<th>( \circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE'</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \circ \) is undefined. In this case, create a program so that the data output from \( \circ \) is not used.
5-46

5.1 Type Conversion Functions

5.1.16 Double-precision real type → word (signed), double word (signed) type conversion

Operation Error

An operation error occurs in the following cases.

- The input value is -0 or outside the following range. (Error code: 4140)
  \[ 0, 2^{-1022} \leq |x| < 2^{1024} \]
- \( \text{LREAL}_\text{TO}_\text{INT}_\text{E}(\_E) \): The input value is outside the range of -32768 to 32767. (Error code: 4140)
- \( \text{LREAL}_\text{TO}_\text{DINT}_\text{E}(\_E) \): The input value is outside the range of -2147483648 to 2147483647. (Error code: 4140)

Program Example

1. The program which converts double-precision real type data input to \( \oplus \) into word (signed) type data, and outputs the operation result from \( \ominus \).
   (a) Function without EN/ENO (LREAL_TO_INT)
   [Structured ladder/FBD]
   
   ```
   g_int1 := \text{LREAL}_\text{TO}_\text{INT}(g\_lreal1);
   ```
   
   [ST]
   
   ```
   g\_int1 := \text{LREAL}_\text{TO}_\text{INT}(g\_lreal1);
   ```
   
   (b) Function with EN/ENO (LREAL_TO_INT_E)
   [Structured ladder/FBD]
   
   ```
   g\_bool3 := \text{LREAL}_\text{TO}_\text{INT}_\text{E}(g\_bool1, g\_lreal1, g\_int1);
   ```
   
   2. The program which converts double-precision real type data input to \( \oplus \) into double word (signed) type data, and outputs the operation result from \( \ominus \).
   (a) Function without EN/ENO (LREAL_TO_DINT)
   [Structured ladder/FBD]
   
   ```
   g\_dint1 := \text{LREAL}_\text{TO}_\text{DINT}(g\_lreal1);
   ```
   
   [ST]
   
   ```
   g\_dint1 := \text{LREAL}_\text{TO}_\text{DINT}(g\_lreal1);
   ```
5.1.17 Single-precision real type → double-precision real type conversion

REAL_TO_LREAL(E)

_E: With EN/ENO

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL_TO_LREAL(E)</td>
<td>EN</td>
</tr>
<tr>
<td>s(REAL)</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Input argument, EN:</td>
<td></td>
</tr>
<tr>
<td>s(REAL):</td>
<td></td>
</tr>
<tr>
<td>Output argument, ENO:</td>
<td></td>
</tr>
<tr>
<td>d:</td>
<td></td>
</tr>
<tr>
<td>Execution condition</td>
<td></td>
</tr>
<tr>
<td>(TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>:Single-precision real</td>
<td></td>
</tr>
<tr>
<td>Execution result</td>
<td></td>
</tr>
<tr>
<td>(TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>:Double-precision real</td>
<td></td>
</tr>
</tbody>
</table>

**Function**

**Operation processing**

1. Converts single-precision real type data input to ① into double-precision real type data, and outputs the operation result from ③.

   ![Single-precision real type](1234.0) → ![Double-precision real type](1234.0)

2. Rounding error may occur when specifying the input value to ① by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
Operation result

(1) Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ② is undefined. In this case, create a program so that the data output from ② is not used.
5.1 Type Conversion Functions

5.1.17 Single-precision real type \( \rightarrow \) double-precision real type conversion

---

### Operation Error

An operation error occurs in the following cases.  
- The input value is -0 or outside the following range.  
  \(0, 2^{-126} \leq x < 2^{127}\)  
  (Error code: 4140)

- The operation result is outside the following range (an overflow occurrence).  
  \(2^{1024} \geq |\text{operation result}|\)  
  (Error code: 4141)

### Program Example

The program which converts single-precision real type data input to \(\odot\) into double-precision real type data, and outputs the operation result from \(\odot\).

(a) Function without EN/ENO (REAL_TO_LREAL)

[Structured ladder/FBD]

```
1
```


```
g_lreal1 := REAL_TO_LREAL(g_real1);
```

(b) Function with EN/ENO (REAL_TO_LREAL_E)

[Structured ladder/FBD]

```
2
```

```
g_bool3 := REAL_TO_LREAL_E(g_bool1, g_real1, g_lreal1);
```

---
5.1.18 Double-precision real type → single-precision real type conversion

**LREAL_TO_REAL(_E)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Converts double-precision real type data input to [①] into single-precision real type data, and outputs the operation result from [③].</td>
</tr>
</tbody>
</table>

![Structured ladder/FBD and ST diagram]

- **Input argument**: 
  - EN: Executing condition (TRUE: Execution, FALSE: Stop)
  - s_LREAL: Input
- **Output argument**: 
  - d: Output
  - ENO: Execution result (TRUE: Normal, FALSE: Error)

- **EN, ENO**: The function uses the EN, ENO signals. If both EN and ENO are set to TRUE, ENO is output.

- **Indicates any of the following functions**: 
  - LREAL_TO_REAL
  - LREAL_TO_REAL_E

(2) Rounding error may occur when setting the input value to [①] by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
Operation result

(1) Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^1\): When FALSE is output from ENO, the data output from (2) is undefined. In this case, create a program so that the data output from (2) is not used.
5.1 Type Conversion Functions

5.1.18 Double-precision real type → single-precision real type conversion

**Operation Error**

An operation error occurs in the following cases.

- The input value is -0 or outside the following range. (Error code: 4140)
  \[ 0, 2^{-1022} \leq | \text{input value} | < 2^{1024} \]
- The operation result is outside the following range (an overflow occurrence). (Error code: 4141)
  \[ 2^{128} \leq | \text{operation result} | \]

**Program Example**

The program which converts double-precision real type data input to \( \text{③} \) into single-precision real type data, and outputs the operation result from \( \text{④} \).

(a) Function without EN/ENO (LREAL_TO_REAL)
[Structured ladder/FBD]

```plaintext
[ST]
g_real1 := LREAL_TO_REAL(g_lreal1);
```

(b) Function with EN/ENO (LREAL_TO_REAL_E)
[Structured ladder/FBD]

```plaintext
[ST]
g_bool3 := LREAL_TO_REAL_E(g_bool1, g_lreal1, g_real1);
```
5.1.19 Single-precision real type → string type conversion

REAL_TO_STR(_E)

_E: With EN/ENO

Function Operation processing

(1) Converts single-precision real type data input to \( s \) into string type (exponential form) data, and outputs the operation result from \( d \).

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL_TO_STR(_E)</td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s(_REAL)</td>
<td></td>
</tr>
<tr>
<td>ENO:=REAL_TO_STR_E(EN, s, d);</td>
<td></td>
</tr>
</tbody>
</table>

Input argument, \( s(_REAL) \):
- \( s \): Single-precision real

Output argument, \( d \):
- \( d \): String(13)

High-order byte
- 20h(space): Sign data (integer part)
- 2En(\( \times \)): ASCII code of integer part
- ASCII code of two decimal places
- ASCII code of four decimal places
- ASCII code of five decimal places
- ASCII code of exponent part's tens place
- ASCII code of exponent part's units place

Low-order byte
- 00h(NUL): ASCII code of sign data (exponent part)

When SM701 (signal for switching the number of output character) is OFF, "00h" is stored.
(2) The character string data after conversion is output from output variable \( \textcircled{5} \) in the following manner.

(a) The number of digits is fixed respectively for the integer part, fractional part, and exponent part. (Integer part: 1 digit, fractional part: 5 digits, exponent part: 2 digits)

\[ -12.3456 \quad \text{Single-precision real type} \]

\( \text{Integer part (1 digit)} \) | Fractional part (5 digits) | Exponent part (2 digits)
---|---|---
| 2E (.) | 45H (E) | 20H (space)

All together 13 digits

(b) ‘20H’ (space) is stored in ‘Sign data’ (integer part) when the input value is positive; ‘2DH’ (−) is stored when negative.

(c) Fractional part is rounded to 5 decimal places.

\[ -12.345678 \quad \text{Single-precision real type} \]

\( \text{Fractional part (5 digits)} \) | Rounded off.
---|---
| 2E (.) | 45H (E) | 20H (space)

All together 13 digits

(d) If the number of significant figures is less, ‘30H’ (0) is stored to fractional part.

\[ -12.34 \quad \text{Single-precision real type} \]

\( \text{Fractional part (5 digits)} \)

\( 30H (0) \)

All together 13 digits

(e) ‘2BH’ (+) is stored in the ‘Sign data’ (exponent part) if the exponent is positive; ‘2DH’ (−) is stored when negative.

\[ -12.3456 \quad \text{Single-precision real type} \]

\( \text{Exponent part (2 digits)} \)

\( 30H (0) \)

All together 13 digits

(f) ‘30H’ (0) is stored to tens place in the exponent part if exponent part has only one digit.

\[ -12.34 \quad \text{Single-precision real type} \]

\( \text{Exponent part (2 digits)} \)

\( 30H (0) \)

All together 13 digits

(3) When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string (7th word).

(4) Rounding error may occur when specifying the input value to \( \textcircled{5} \) by programming tool.

For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
5.1 Type Conversion Functions

5.1.19 Single-precision real type → string type conversion

Operation result

(1) Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^*1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^*1\): When FALSE is output from ENO, the data output from ② is undefined.
In this case, create a program so that the data output from ② is not used.
5.1 Type Conversion Functions

5.1.19 Single-precision real type → string type conversion

**Operation Error**

An operation error occurs in the following case.

- The input value is outside the range of \(-3.40282 \times 10^{38}\) to \(-1.17549 \times 10^{-38}\), 0 or \(1.17549 \times 10^{-38}\) to \(3.40282 \times 10^{38}\) (Error code: 4100)

**Program Example**

The program which converts single-precision real type data input to \(\odot\) into string type (exponential form) data, and outputs the operation result from \(\odot\).  

(a) Function without EN/ENO (REAL_TO_STR)

[Structured ladder/FBD]

```
[ST]
g_string1 := REAL_TO_STR(g_real1);
```

(b) Function with EN/ENO (REAL_TO_STR_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := REAL_TO_STR_E(g_bool1, g_real1, g_string1);
```
5.1.20 Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type → bit type conversion

**WORD_TO_BOOL(_E), DWORD_TO_BOOL(_E)**

- **WORD_TO_BOOL(_E)**
  - Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop)
  - s(_WORD, _DWORD): Input
  - d: Output

- **DWORD_TO_BOOL(_E)**
  - Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop)
  - Input
  - d: Output

**Function**

**Operation processing**

1. **WORD_TO_BOOL, WORD_TO_BOOL_E**
   - Converts word (unsigned)/16-bit string type data input to d into bit type data, and outputs the operation result from d.
   - When the input value is 0H, FALSE is output.
   - When the input value is other than 0H, TRUE is output.

```
0H  ➔  FALSE
1567H  ➔  TRUE
```

Word (unsigned)/16-bit string type  ➔  Bit type
(2) DWORD_TO_BOOL, DWORD_TO_BOOL_E

Converts double word (unsigned)/32-bit string type data input to  into bit type data, and outputs the operation result from .
When the input value is 0H, FALSE is output.
When the input value is other than 0H, TRUE is output.

<table>
<thead>
<tr>
<th>Double word (unsigned)/32-bit string type</th>
<th>Bit type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>FALSE</td>
</tr>
<tr>
<td>12345678H</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

**Operation result**

(1) Function without EN/ENO
An operation is executed and the operation value is output from  .

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from  is undefined. In this case, create a program so that the data output from  is not used.
No operation error occurs in the execution of the WORD_TO_BOOL(_E) and DWORD_TO_BOOL(_E) functions.

**Program Example**

1. The program which converts word (unsigned)/16-bit string type data input to ① into bit type data, and outputs the operation result from ③.
   
   (a) Function without EN/ENO (WORD_TO_BOOL)
   
   [Structured ladder/FBD]
   
   [ST]
   
   ```
   g_bool1 := WORD_TO_BOOL(g_word1);
   ```

   (b) Function with EN/ENO (WORD_TO_BOOL_E)
   
   [Structured ladder/FBD]
   
   [ST]
   
   ```
   g_bool3 := WORD_TO_BOOL_E(g_bool1, g_word1, g_bool2);
   ```

2. The program which converts double word (unsigned)/32-bit string type data input to ② into bit type data, and outputs the operation result from ③.
   
   (a) Function without EN/ENO (DWORD_TO_BOOL)
   
   [Structured ladder/FBD]
   
   [ST]
   
   ```
   g_bool1 := DWORD_TO_BOOL(g_dword1);
   ```
5.1.21 Word (unsigned)/16-bit string type → word (signed), double word (signed) type conversion

WORD_TO_INT(_E), WORD_TO_DINT(_E)

Function

Operation processing

(1) WORD_TO_INT, WORD_TO_INT_E

Converts word (unsigned)/16-bit string type data input to ① into word (signed) type data, and outputs the operation result from ② .

5678H → 22136

Word (unsigned)/16-bit string type

Word (signed) type
(2) **WORD_TO_DINT, WORD_TO_DINT_E**

Converts word (unsigned)/16-bit string type data input to \(\textcircled{2}\) into double word (signed) type data, and outputs the operation result from \(\textcircled{3}\).

Word (unsigned)/16-bit string type \(\rightarrow\) Double word (signed) type

<table>
<thead>
<tr>
<th>5678h</th>
<th>22136</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101011001111000</td>
<td>0101011001111000</td>
</tr>
</tbody>
</table>

Data conversion

Always filled with 0s.

**Operation result**

1. **Function without EN/ENO**
   - An operation is executed and the operation value is output from \(\textcircled{3}\).

2. **Function with EN/ENO**
   - The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>(\textcircled{3})</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^1\): When FALSE is output from ENO, the data output from \(\textcircled{3}\) is undefined. In this case, create a program so that the data output from \(\textcircled{3}\) is not used.
### Operation Error

No operation error occurs in the execution of the WORD_TO_INT(_E) and WORD_TO_DINT(_E) functions.

### Program Example

(1) The program which converts word (unsigned)/16-bit string type data input to \( \circ \) into word (signed) type data, and outputs the operation result from \( \circ \).

(a) Function without EN/ENO (WORD_TO_INT)

```
[Structured ladder/FBD]
```

```
1

WORD_TO_INT

\[g\text{\_int1} := \text{WORD\_TO\_INT}(\text{g\text\_word1});\]
```

(b) Function with EN/ENO (WORD_TO_INT_E)

```
[Structured ladder/FBD]
```

```
2

\[\text{g\text\_bool3} := \text{WORD\_TO\_INT\_E}(\text{g\text\_bool1}, \text{g\text\_word1}, \text{g\text\_int1});\]
```

(2) The program which converts word (unsigned)/16-bit string type data input to \( \circ \) into double word (signed) type data, and outputs the operation result from \( \circ \).

(a) Function without EN/ENO (WORD_TO_DINT)

```
[Structured ladder/FBD]
```

```
1

WORD_TO_DINT

\[g\text{\_dint1} := \text{WORD\_TO\_DINT}(\text{g\text\_word1});\]
```
5.1.22 Double word (unsigned)/32-bit string type → word (signed), double word (signed) type conversion

DWORD_TO_INT(_E), DWORD_TO_DINT(_E)

_FUNCTIONAL

DWORD_TO_INT(_E), DWORD_TO_DINT(_E)  
_E: With EN/ENO

Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop)

s(DWORD): Input

Output argument, ENO: Execution result (TRUE: Normal, FALSE: Error)

d: Output

_FUNCTIONAL

DWORD_TO_INT(_E)
DWORD_TO_DINT(_E)

☆ Function

Operation processing

(1) DWORD_TO_INT, DWORD_TO_INT_E

Converts double word (unsigned)/32-bit string type data input to \(s\) into word (signed) type data, and outputs the operation result from \(d\).

\[
\begin{align*}
&\text{Double word (unsigned)/32-bit string type} \\
&\text{Word (signed) type}
\end{align*}
\]

High-order 16-bit data is discarded.
5.1 Type Conversion Functions

5.1.22 Double word (unsigned)/32-bit string type → word (signed), double word (signed) type conversion

(2) DWORD_TO_DINT, DWORD_TO_DINT_E

Converts double word (unsigned)/32-bit string type data input to ① into double word (signed) type data, and outputs the operation result from ②.

\[
\begin{array}{c|c|c}
\text{DWORD_TO_DINT, DWORD_TO_DINT_E} & & \\
\hline
\text{Operation result} & & \\
\hline
\text{(1) Function without EN/ENO} & & \\
\quad \text{An operation is executed and the operation value is output from ②.} & & \\
\hline
\text{(2) Function with EN/ENO} & & \\
\quad \text{The following table shows the executing conditions and operation results.} & & \\
\hline
\text{EN} & \text{ENO} & \text{②} \\
\text{TRUE (Operation execution)} & \text{TRUE} & \text{Operation output value} \\
\text{FALSE (Operation stop)} & \text{FALSE}^{*1} & \text{Undefined value} \\
\end{array}
\]

*1: When FALSE is output from ENO, the data output from ② is undefined. In this case, create a program so that the data output from ② is not used.

**POINT**

When the DINT_TO_INT(E) function is executed, low-order 16-bit data of double word (unsigned)/32-bit string type data input to ③ are converted into word (signed) type data. High-order 16-bit data are discarded.
5.1 Type Conversion Functions

5.1.22 Double word (unsigned)/32-bit string type → word (signed), double word (signed) type conversion

Operation Error

No operation error occurs in the execution of the DWORD_TO_INT(_E) and DWORD_TO_DINT(_E) functions.

Program Example

(1) The program which converts double word (unsigned)/32-bit string type data input to word (signed) type data, and outputs the operation result from.

(a) Function without EN/ENO (DWORD_TO_INT)

[Structured ladder/FBD]

[ST]
\[ g\_int1 := \text{DWORD}\_\text{TO}\_\text{INT}(g\_dword1); \]

(b) Function with EN/ENO (DWORD_TO_INT_E)

[Structured ladder/FBD]

[ST]
\[ g\_bool3 := \text{DWORD}\_\text{TO}\_\text{INT}\_\text{E}(g\_bool1, g\_dword1, g\_int1); \]

(2) The program which converts double word (unsigned)/32-bit string type data input to double word (signed) type data, and outputs the operation result from.

(a) Function without EN/ENO (DWORD_TO_DINT)

[Structured ladder/FBD]

[ST]
\[ g\_dint1 := \text{DWORD}\_\text{TO}\_\text{DINT}(g\_dword1); \]

5.1 Type Conversion Functions

5.1.22 Double word (unsigned)/32-bit string type → word (signed), double word (signed) type conversion

5-65
5.1.23 Word (unsigned)/16-bit string type → double word (unsigned)/32-bit string type conversion

**WORD_TO_DWORD(_E)**

Operation processing

Converts word (unsigned)/16-bit string type data input to ① into double word (unsigned)/32-bit string type data, and outputs the operation result from ②.

After data conversion, high-order 16 bits are filled with 0s.

![Structured ladder/FBD](image)

<table>
<thead>
<tr>
<th>Input argument,</th>
<th>Output argument,</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN:</td>
<td>ENO:</td>
</tr>
<tr>
<td>s:</td>
<td>d:</td>
</tr>
</tbody>
</table>

- **EN**: Executing condition (TRUE: Execution, FALSE: Stop) :Bit
- **ENO**: Execution result (TRUE: Normal, FALSE: Error) :Bit
- **s**: Word (unsigned)/16-bit string
- **d**: Double word (unsigned)/32-bit string
Operation result

(1) Function without EN/ENO
   An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE^1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

^1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
### Operation Error

No operation error occurs in the execution of the WORD_TO_DWORD(_E) function.

### Program Example

The program which converts word (unsigned)/16-bit string type data input to \( ① \) into double word (unsigned)/32-bit string type data, and outputs the operation result from \( ② \).

(a) Function without EN/ENO (WORD_TO_DWORD)

[Structured ladder/FBD]

\[
g_{\text{dword1}} := \text{WORD_TO_DWORD}(g_{\text{word1}});
\]

(b) Function with EN/ENO (WORD_TO_DWORD_E)

[Structured ladder/FBD]

\[
g_{\text{bool3}} := \text{WORD_TO_DWORD_E}(g_{\text{bool1}}, g_{\text{word1}}, g_{\text{dword1}});
\]
5.1.24 Double word (unsigned)/32-bit string type → word (unsigned)/16-bit string type conversion

**DWORD_TO_WORD(E)**

_E: With EN/ENO

**Function**

**Operation processing**

Converts double word (unsigned)/32-bit string type data input to 1 into word (unsigned)/16-bit string type data, and outputs the operation result from 2.

12345678H → 5678H

Double word (unsigned)/32-bit string type

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Word (unsigned)/16-bit string type

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

High-order 16-bit data is discarded.
Operation result

(1) Function without EN/ENO
   An operation is executed and the operation value is output from ②.

(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ② is undefined.

In this case, create a program so that the data output from ② is not used.

POINT

When the DWORD_TO_WORD(_E) function is executed, low-order 16-bit data of double word (unsigned)/32-bit string type data input to ① are converted into word (unsigned)/16-bit string type data. High-order 16-bit data are discarded.
5.1 Type Conversion Functions

5.1.24 Double word (unsigned)/32-bit string type → word (unsigned)/16-bit string type conversion

Program Example

The program which converts double word (unsigned)/32-bit string type data input to \( \odot \) into word (unsigned)/16-bit string type data, and outputs the operation result from \( \odot \).

(a) Function without EN/ENO (DWORD_TO_WORD)
[Structured ladder/FBD]

```plaintext
ST

\( g_{\text{word1}} := \text{DWORD_TO_WORD}(g_{\text{dword1}}); \)
```

(b) Function with EN/ENO (DWORD_TO_WORD_E)
[Structured ladder/FBD]

```plaintext
ST

\( g_{\text{bool3}} := \text{DWORD_TO_WORD_E}(g_{\text{bool1}}, g_{\text{dword1}}, g_{\text{word1}}); \)
```

Operation Error

No operation error occurs in the execution of the DWORD_TO_WORD(E) function.
5.1.25 Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type → string type conversion

WORD_TO_STR(_E), DWORD_TO_STR(_E)

Function

Operation processing

(1) WORD_TO_STR, WORD_TO_STR_E

(a) Converts word (unsigned)/16-bit string type data input to ② into string type data, and outputs the operation result from ③.

① 0H

② 1567H

③ ' 0 '

Word (unsigned)/16-bit string type

String type

(b) When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string.
(2) DWORD_TO_STR, DWORD_TO_STR_E

(a) Converts double word (unsigned)/32-bit string type data input to \( \circ \) into string type data, and outputs the operation result from \( \odot \).

(b) When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string.

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( \odot \).

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \odot )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^1\): When FALSE is output from ENO, the data output from \( \odot \) is undefined. In this case, create a program so that the data output from \( \odot \) is not used.
5.1 Type Conversion Functions

5.1.25 Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type → string type conversion

---

**Operation Error**

No operation error occurs in the execution of the `WORD_TO_STR(_E)` and `DWORD_TO_STR(_E)` functions.

---

**Program Example**

(1) The program which converts word (unsigned)/16-bit string type data input to [ ] into string type data, and outputs the operation result data from [ ] .

(a) Function without EN/ENO (`WORD_TO_STR`)

```
[Structured ladder/FBD]
```

```
[ST]
g_string1 := WORD_TO_STR (g_word1);
```

(b) Function with EN/ENO (`WORD_TO_STR_E`)

```
[Structured ladder/FBD]
```

```
[ST]
g_bool3 := WORD_TO_STR_E (g_bool1, g_word1, g_string1);
```

(2) The program which converts double word (unsigned)/32-bit string type data input to [ ] into string type data, and outputs the operation result data from [ ] .

(a) Function without EN/ENO (`DWORD_TO_STR`)

```
[Structured ladder/FBD]
```

```
[ST]
g_string1 := DWORD_TO_STR (g_dword1);
```

---

---
Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type → time type conversion

WORD_TO_TIME(_E), DWORD_TO_TIME(_E)

Function

Operation processing

(1) WORD_TO_TIME, WORD_TO_TIME_E

Converts word (unsigned)/16-bit string type data to time type data, and outputs the operation result from.

\[
\begin{align*}
\text{Input argument,} & \quad \text{EN:} \quad \text{Executing condition (TRUE: Execution, FALSE: Stop)} \\
& \quad \text{s (_WORD, _DWORD):} \quad \text{Input} \\
\text{Output argument,} & \quad \text{ENO:} \quad \text{Execution result (TRUE: Normal, FALSE: Error)} \\
& \quad d: \quad \text{Output}
\end{align*}
\]

Word (unsigned)/16-bit string type → Time type

\[
\begin{align*}
0H & \Rightarrow \text{T#0ms} \\
1234H & \Rightarrow \text{T#1s234ms}
\end{align*}
\]
(2) DWORD_TO_TIME, DWORD_TO_TIME_E

Converts double word (unsigned)/32-bit string type data input to double word (unsigned)/32-bit string type into time type data, and outputs the operation result from the operation value.

<table>
<thead>
<tr>
<th>Double word (unsigned)/32-bit string type</th>
<th>Time type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>T#0ms</td>
</tr>
<tr>
<td>1234567H</td>
<td>T#20m34s567ms</td>
</tr>
</tbody>
</table>

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from the operation value.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
**Operation Error**

No operation error occurs in the execution of the \texttt{WORD\_TO\_TIME(\_E)} and \texttt{DWORD\_TO\_TIME(\_E)} functions.

**Program Example**

(1) The program which converts word (unsigned)/16-bit string type data input to \(\odot\) into time type data, and outputs the operation result from \(\odot\).

(a) Function without EN/ENO (\texttt{WORD\_TO\_TIME})

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{time1}} := \texttt{WORD\_TO\_TIME} (g_{\text{word1}}) ;
\end{array}
\]

(b) Function with EN/ENO (\texttt{WORD\_TO\_TIME(E)})

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{bool3}} := \texttt{WORD\_TO\_TIME(E)} (g_{\text{bool1}}, g_{\text{word1}}, g_{\text{time1}}) ;
\end{array}
\]

(2) The program which converts double word (unsigned)/32-bit string type data input to \(\odot\) into time type data, and outputs the operation result from \(\odot\).

(a) Function without EN/ENO (\texttt{DWORD\_TO\_TIME})

[Structured ladder/FBD]

\[
\begin{array}{c}
g_{\text{time1}} := \texttt{DWORD\_TO\_TIME} (g_{\text{dword1}})
\end{array}
\]
5.1.27 String type → bit type conversion

**STR_TO_BOOL(_E)**

---

**Function**

**Operation processing**

Converts string type data input to \( \text{'0'} \) into bit type data, and outputs the operation result from \( \text{'1'} \).

When the input value is 0, FALSE is output in bit type data.
When the input value is other than 0, TRUE is output in bit type data.
**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from ②.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE¹</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ② is undefined. In this case, create a program so that the data output from ② is not used.
Operation Error

No operation error occurs in the execution of the STR_TO_BOOL(_E) function.

Program Example

The program which converts string type data input to \( \circ \) into bit type data, and outputs the operation result from \( \ominus \).

(a) Function without EN/ENO (STR_TO_BOOL)

```
[Structured ladder/FBD]

ST

g_bool1 := STR_TO_BOOL (g_string1);
```

(b) Function with EN/ENO (STR_TO_BOOL_E)

```
[Structured ladder/FBD]

ST

g_bool3 := STR_TO_BOOL_E (g_bool1, g_string1, g_bool2);
```
5.1.28 String type \rightarrow word (signed), double word (signed) type conversion

**STR_TO_INT(_E), STR_TO_DINT(_E)**

### Function

**Operation processing**

1. **STR_TO_INT, STR_TO_INT_E**
   
   (a) Converts string type data input into word (signed) type data, and outputs the operation result from \(\circ\).

<table>
<thead>
<tr>
<th>String 1st word</th>
<th>ASCII code of ten-thousands place</th>
<th>High-order byte</th>
<th>(\circ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>ASCII code of hundreds place</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd word</td>
<td>ASCII code of units place</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th word</td>
<td>ASCII code of tens place</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   00H (Indicates the end of the character string.)

   Low-order byte

<table>
<thead>
<tr>
<th>ASCII code of thousands place</th>
<th>Sign data</th>
</tr>
</thead>
</table>

   Word (signed) type

(b) The value to be input to \(\circ\) is string type data within the following range.

   ASCII code: '30H' to '39H', '20H', '2DH', and '00H'

   String type data: '-32768' to '32767'
(2) STR_TO_DINT, STR_TO_DINT_E

(a) Converts string type data input to ② into double word (signed) type data, and outputs the operation result from ③.

<table>
<thead>
<tr>
<th>String 1st word</th>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of billions place</td>
<td>Sign data</td>
<td></td>
</tr>
</tbody>
</table>

| 2nd word | ASCII code of ten-millions place | ASCII code of hundred-millions place |
| 3rd word | ASCII code of hundred-thousands place | ASCII code of millions place |
| 4th word | ASCII code of thousands place | ASCII code of ten-thousands place |
| 5th word | ASCII code of tens place | ASCII code of hundreds place |
| 6th word | 00H | ASCII code of units place |

 Indicates the end of the character string.

(b) The value to be input to ② is string type data within the following range.
ASCII code: '30H' to '39H', '20H', '2DH', and '00H'
String type data: -2147483648 to 2147483647

**Operation result**

(1) Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ② is undefined. In this case, create a program so that the data output from ② is not used.
5.1 Type Conversion Functions

5.1.28 String type → word (signed), double word (signed) type conversion

### Operation Error

An operation error occurs in the following cases.

- The input value is other than '30H' to '39H', '20H', '2DH', and '00H' of ASCII code.  
  (Error code: 4100)

- The input value is outside the following ranges of ASCII code.  
  (Error code: 4100)
  STR_TO_INT(_E): −32768 to 32767
  STR_TO_DINT(_E): −2147483648 to 2147483647

### Program Example

(1) The program which converts string type data input to ☰ into word (signed) type data, and outputs the operation result from ☰.

(a) Function without EN/ENO (STR_TO_INT)

[Structured ladder/FBD]

```
[ST]
g_int1 := STR_TO_INT (g_string1);
```

(b) Function with EN/ENO (STR_TO_INT_E)

[Structured ladder/FBD]

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

(2) The program which converts string type data input to ☰ into double word (signed) type data, and outputs the operation result from ☰.

(a) Function without EN/ENO (STR_TO_DINT)

[Structured ladder/FBD]

```
[ST]
g_dint1 := STR_TO_DINT (g_string1);
```
5.1.29 String type → single-precision real type conversion

**STR_TO_REAL(_E)**

This function is used in the Basic model QCPU with a serial number (first five digits) of "04122" or later.

**Function**

**Operation processing**

1. Converts string type (decimal form/exponential form) data input to (1) into single-precision real type data, and outputs the operation result from (2).

<table>
<thead>
<tr>
<th>String 1st word</th>
<th>ASCII code of 1st character</th>
<th>Sign date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>ASCII code of 3rd character</td>
<td>ASCII code of 2nd character</td>
</tr>
<tr>
<td>3rd word</td>
<td>ASCII code of 5th character</td>
<td>ASCII code of 4th character</td>
</tr>
<tr>
<td>4th word</td>
<td>ASCII code of 7th character</td>
<td>ASCII code of 6th character</td>
</tr>
<tr>
<td>5th word</td>
<td>ASCII code of 9th character</td>
<td>ASCII code of 8th character</td>
</tr>
<tr>
<td>6th word</td>
<td>ASCII code of 11th character</td>
<td>ASCII code of 10th character</td>
</tr>
<tr>
<td>7th word</td>
<td>00H (Indicates the end of the character string.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR_TO_REAL</td>
<td>STR_TO_REAL_E</td>
</tr>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s</td>
<td>d</td>
</tr>
</tbody>
</table>

(EN, s, d): Indicates any of the following functions.

ST

EN: Executing condition (TRUE: Execution, FALSE: Stop): Bit
s(_STRING): Input : String (24)
ENO: Execution result (TRUE: Normal, FALSE: Error): Bit
d: Output : Single-precision real
(2) Both string type data in decimal form and exponential form can be converted to single-precision real type data.

(a) Decimal form

<table>
<thead>
<tr>
<th>String 1st word</th>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>31H(1)</td>
<td>2Dh(-)</td>
</tr>
<tr>
<td>3rd word</td>
<td>33H(3)</td>
<td>2Eh(-)</td>
</tr>
<tr>
<td>4th word</td>
<td>30H(0)</td>
<td>35H(5)</td>
</tr>
<tr>
<td>5th word</td>
<td>34H(4)</td>
<td>33H(3)</td>
</tr>
<tr>
<td>6th word</td>
<td>00H</td>
<td></td>
</tr>
</tbody>
</table>

\[ -1.35034 \] Single-precision real type

(b) Exponential form

<table>
<thead>
<tr>
<th>String 1st word</th>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd word</td>
<td>31H(1)</td>
<td>2Dh(-)</td>
</tr>
<tr>
<td>3rd word</td>
<td>33H(3)</td>
<td>2Eh(-)</td>
</tr>
<tr>
<td>4th word</td>
<td>30H(0)</td>
<td>35H(5)</td>
</tr>
<tr>
<td>5th word</td>
<td>34H(4)</td>
<td>33H(3)</td>
</tr>
<tr>
<td>6th word</td>
<td>2Dh(-)</td>
<td>45H(E)</td>
</tr>
<tr>
<td>7th word</td>
<td>00H</td>
<td></td>
</tr>
</tbody>
</table>

\[ -1.35034E-10 \] Single-precision real type

(3) As the number of significant figures of string type data is 6, the 7th and later digits excluding the sign, decimal point, and exponent part are cut and converted.

(a) Decimal form

\[ \ldots 305312023 \] Cut \[ \rightarrow \] \[ -1.35034 \] Single-precision real type

(b) Exponential form

\[ \ldots 305312023 \] Cut \[ \rightarrow \] \[ -1.35034E-10 \] Single-precision real type

(4) When a sign is not specified or ‘2BH’ (+) is specified for a sign in decimal form, string type data is converted as a positive value. When ‘2DH’ (−) is specified for a sign, string type data is converted as a negative value.

(5) When a sign is not specified or ‘2BH’ (+) is specified for a sign of the exponent part in exponential form, string type data is converted as a positive value. When ‘2DH’ (−) is specified for a sign of the exponential part, string type data is converted as a negative value.
5.1 Type Conversion Functions

5.1.29 String type → single-precision real type conversion

(6) When '20h' (space) or '30h' (0) exists before the first 0 in string type data, the conversion is executed ignoring '20h' and '30h'.

(a) Decimal form

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>35</td>
<td>03</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Neglected

Single-precision real type

-1.35034

(7) When '30h' (0) exists between 'E' and a numeric value in string type data (exponential form), the conversion is executed ignoring '30h'.

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>35</td>
<td>03</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Neglected

Single-precision real type

-1.35034E-10

(8) When '20h' (space) exists in the character string, the conversion is executed ignoring '20h'.

(9) String type data can contain up to 24 characters.

'20h' (space) and '30h' (0) in the character string are counted as one character.

(10) The value to be input to \(\odot\) is string type data within the following range.

ASCII code: '30h' to '39h', '45h', '2Bh', '2Dh', '2EH', '20h', and '00h'

**Operation result**

(1) Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(\odot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>(\odot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \(\odot\) is undefined. In this case, create a program so that the data output from \(\odot\) is not used.
### Operation Error

An operation error occurs in the following cases.

- Any characters other than '30H' to '39H' exist in the integer or fractional part.  
  (Error code: 4100)

- Two or more 2EH exist.  
  (Error code: 4100)

- Any characters other than ‘45H(E), ’2BH(+)’ or ‘45H(E), ’2DH(–)’ exist in the exponent part, or more than one exponent parts exist.  
  (Error code: 4100)

- The data after conversion is outside the range of -3.4028238 to -1.17549-38, 0 or 1.17549-38 to 3.4028238  
  (Error code: 4100)

- The number of characters is 0 or exceeding 24.  
  (Error code: 4100)

### Program Example

The program which converts string type data input to into single-precision real type data, and outputs the operation result from .

(a) Function without EN/ENO (STR_TO_REAL)

[Structured ladder/FBD]

```
ST

[ST]
g_real1 := STR_TO_REAL (g_string1);
```

(b) Function with EN/ENO (STR_TO_REAL_E)

[Structured ladder/FBD]

```
ST

[ST]
g_bool3 := STR_TO_REAL_E (g_bool1, g_string1, g_real1);
```
5.1.30 String type → word (unsigned)/16-bit string, double word (unsigned)/32-bit string type conversion

**STR_TO_WORD(_E), STR_TO_DWORD(_E)**

_E: With EN/ENO

---

**Function Operation processing**

1. **STR_TO_WORD, STR_TO_WORD_E**

   Converts string type data input to ⑩ into word (unsigned)/16-bit string type data, and outputs the operation result from ⑪.

   - **String type**: "0000"
   - **Word (unsigned)/16-bit string type**: 0H

   - **String type**: "0012"
   - **Word (unsigned)/16-bit string type**: 12H
(2) STR_TO_DWORD, STR_TO_DWORD_E

Converts the string type data input to ① into double word (unsigned)/32-bit string type data, and outputs the operation result from ③.

- "00000000" → 0h
- "00000012" → 12h

String type  Double word (unsigned)/32-bit string type

**Operation result**

1. **Function without EN/ENO**
   An operation is executed and the operation value is output from ③.

2. **Function with EN/ENO**
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
5.1 Type Conversion Functions

5.1.30 String type → word (unsigned)/16-bit string, double word (unsigned)/32-bit string type conversion

### Operation Error

These functions consist of the following instructions.

- **STR_TO_WORD(_E)** : HABIN
- **STR_TO_DWORD(_E)** : DHABIN

In any of the following cases, an operation error occurs, the error flag (SM0) is turned ON, and the corresponding error code is stored to SD0.

- The ASCII code for each number specified for is outside the range of 30H to 39H, 41H to 46H. (Error code: 4100)
- The device specified for exceeds the corresponding device range. (For Universal model QCPU and LCPU) (Error code: 4101)

### Program Example

(1) The program which converts string type data input to into word (unsigned)/16-bit string type data, and outputs the converted data from .

(a) Function without EN/ENO (STR_TO_WORD)

[Structured ladder/FBD]

```
[ST]
  g_string1 := "0012";
  g_word1 := STR_TO_WORD (g_string1);
```

(b) Function with EN/ENO (STR_TO_WORD_E)

[Structured ladder/FBD]

```
[ST]
  g_string1 := "0012";
  g_bool3 := STR_TO_WORD_E (g_bool1, g_string1, g_word1);
```
(2) The program which converts string type data input to ① into double word (unsigned)/32-bit string type data, and outputs the operation result from ②.

(a) Function without EN/ENO (STR_TO_DWORD)

[Structured ladder/FBD]

[ST]

```plaintext
  g_string1 := "00000012";
  g_dword1 := STR_TO_DWORD (g_string1);
```
5.1.31 String type → time type conversion

**STR_TO_TIME(_E)**

- **Function Operation processing**
  - Converts string type data input to ⎮ into time type data, and outputs the operation result from ⎮.

### Example

- **String type**
  - "00000000"
  - "01234567"

- **Time type**
  - 0ms
  - T#20m34s567ms
Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined.
In this case, create a program so that the data output from ③ is not used.

Operation Error

This function consists of the following instruction.

STR_TO_TIME(_E) : DDABIN

In any of the following cases, an operation error occurs, the error flag (SM0) is turned ON, and the corresponding error code is stored to SD0.

• The ASCII code for each number specified for ③ is outside the range of 30H to 39H, 20H, and 00H. (Error code: 4100)

• The ASCII data specified for ③ is outside of the range shown below.
  -2147483648 to 4147483647 (Error code: 4100)
Program Example

The program which converts string type data input to \( \circ \) into time type data, and outputs the operation result from \( \bullet \).

(a) Function without EN/ENO (STR_TO_TIME)

```
[Structured ladder/FBD]
```

```
[ST]
g_string1 := "01234567";
g_time1 := STR_TO_TIME (g_string1);
```

(b) Function with EN/ENO (STR_TO_TIME_E)

```
[Structured ladder/FBD]
```

```
[ST]
g_string1 := "01234567";
g_bool3 := STR_TO_TIME_E (g_bool1, g_string1, g_time1);
```
### 5.1.32 String type → BCD type conversion

**STR_TO_BCD**

**Function**

**Operation processing**

1. When word (unsigned)/16-bit string type is specified for output argument.
   
   (a) Converts string type 4-character-string data input to (5) into BCD type data, and outputs the operation result from (3).

   ![String type to BCD type conversion](image)

   (b) When '20H' (space) exists in the character string, the conversion is executed ignoring '20H'.

   (c) '20H' (space) and '30H' (0) in the character string are counted as one character.

   (d) The value to be input to (5) is string type data within the following range.

   ASCII code: '30H' to '39H', '20H', and '00H'

---

**Input argument**, **EN**: Executing condition (TRUE: Execution, FALSE: Stop) : Bit

**Input argument**, **s(STRING)**: String (8)

**Output argument**, **ENO**: Execution result (TRUE: Normal, FALSE: Error) : Bit

**Output argument**, **d**: ANY BIT

---

**Structured ladder/FBD**

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s(STRING)</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ST**

ENO := STR_TO_BCD_E(EN, s, d);

---

5.1 Type Conversion Functions

5.1.32 String type → BCD type conversion
(e) When input character string has less than 4 letters, convert it with 4 letters supplementing with 0 to the end of the character string. Therefore, when converting character string ("0001" for "1") with less than 4 letters to BCD data, input the zero padding character strings.

(f) When the character string has more than 4 letters, the conversion target is the forth character from the left of the character string data.

<table>
<thead>
<tr>
<th>Entered character string</th>
<th>Converted character string</th>
<th>Output (BCD type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1&quot;</td>
<td>&quot;1000&quot;</td>
<td>1000H(4096)</td>
</tr>
<tr>
<td>&quot;12&quot;</td>
<td>&quot;1200&quot;</td>
<td>1200H(4608)</td>
</tr>
<tr>
<td>&quot;123&quot;</td>
<td>&quot;1230&quot;</td>
<td>1230H(4656)</td>
</tr>
<tr>
<td>&quot;1234&quot;</td>
<td>&quot;1234&quot;</td>
<td>1230H(4656)</td>
</tr>
<tr>
<td>&quot;12345&quot;</td>
<td>&quot;1234&quot;</td>
<td>1230H(4656)</td>
</tr>
</tbody>
</table>

(2) When double word (unsigned)/32-bit string type is specified for output argument.

(a) Converts string type 8-character-string data input to \(\circ\) into BCD type data, and outputs the operation result from \(\bullet\).

(b) When '20H' (space) exists in the character string, the conversion is executed ignoring '20H'.

(c) '20H' (space) and '30H' (0) in the character string are counted as one character.

(d) The value to be input to \(\circ\) is string type data within the following range.
ASCII code: '30H' to '39H', '20H', and '00H'

(3) Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type can be specified for \(\circ\).
Bit type cannot be specified.

**POINT**

Output from \(\circ\) cannot be used with connecting to input of function and operator in double word (unsigned)/32-bit string type. In this case, use the DDABCD instruction.
**Operation result**

(1) Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from  is undefined. In this case, create a program so that the data output from  is not used.

---

**Operation Error**

An operation error occurs in the following cases.

- The input character string is outside the range of ASCII code '30H' to '39H', "20H", and "00H". (Error code 4100)
Program Example

(1) The program which converts string type data input to \(\odot\) into BCD type data, and outputs the operation result from \(\odot\).

(a) Function without EN/ENO (STR_TO_BCD)

[Structured ladder/FBD]

```
[ST]
g_string1:="0012";
g_word1 := STR_TO_BCD (g_string1);
```

(b) Function without EN/ENO (STR_TO_BCD_E)

[Structured ladder/FBD]

```
[ST]
g_string1:="0012";
g_bool3 := STR_TO_BCD_E (g_bool1, g_string1, g_word1);
```
(2) The program which converts string type data input to \( \mathcal{S} \) into BCD type data in double word (unsigned)/32-bit string type data, and outputs the operation result from \( \mathcal{S} \).

(a) Function without EN/ENO (STR_TO_BCD)

[Structured ladder/FBD]

```
ST
g_string1 := "00000012";
g_dword1 := STR_TO_BCD (g_string1);
```

(b) Function without EN/ENO (STR_TO_BCD_E)

[Structured ladder/FBD]

```
ST
G_string1 := "00000012";
g_bool3 := STR_TO_BCD_E (g_bool1, g_string1, g_dword1);
```
5.1.33 BCD type → word (signed), double word (signed) type conversion

BCD_TO_INT(_E), BCD_TO_DINT(_E)

Function

Operation processing

(1) BCD_TO_INT, BCD_TO_INT_E

(a) Converts BCD type data input to (③) into word (signed) type data, and outputs the operation result from (④).

(b) The value to be input to (③) is word (unsigned)/16-bit string type data within the range from 0H to 9999H (0 to 9 for each digit).
(2) BCD_TO_DINT, BCD_TO_DINT_E

(a) Converts BCD type data input to (2) into double word (signed) type data, and outputs the operation result from (3).

- When word (unsigned)/16-bit string is specified for (2)

![Diagram](image)

- When double word (unsigned)/32-bit string is specified for (2)

![Diagram](image)

(b) The value to be input to (2) is word (unsigned)/16-bit string data within the range from 0H to 9999H (0 to 9 for each digit), double word (unsigned)/32-bit string data within the range from 0H to 99999999H (0 to 9 for each digit).

(c) Word (unsigned)/16-bit string, double word (unsigned)/32-bit string type can be specified for (2). Bit type cannot be specified.
Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \mathcal{O} \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \mathcal{O} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Op execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Op stop)</td>
<td>FALSE*¹</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*¹: When FALSE is output from ENO, the data output from \( \mathcal{O} \) is undefined.
In this case, create a program so that the data output from \( \mathcal{O} \) is not used.

Operation Error

These functions consist of the following common instructions.

- \( \text{BCD\_TO\_INT\(_E\)} \) : BIN
- \( \text{BCD\_TO\_DINT\(_E\)} \) : BIN, WAND

In any of the following cases, an operation error occurs, the error flag (SM0) is turned ON, and the corresponding error code is stored to SD0.

- Values other than 0 to 9 are specified for each digit of \( \mathcal{O} \).
  (Error code: 4100)

  The error above can be suppressed by turning ON SM722.
  However, the instruction is not executed regardless of whether SM722 is turned ON or OFF if the specified value is out of the available range.
  For the BINP and DBINP instruction, the next operation will not be performed until the command (executing condition) is turned from OFF to ON regardless of the presence or absence of an error.
5.1 Type Conversion Functions

5.1.33 BCD type → word (signed), double word (signed) type conversion

Program Example

(1) The program which converts BCD type data input to \( \circ \) into word (signed) type data, and outputs the operation result from \( \bullet \):

(a) Function without EN/ENO (BCD_TO_INT)

[Structured ladder/FBD]

\[
\text{BCD} \rightarrow \text{BCD} \\
g_{\text{int1}} := \text{BCD\_TO\_INT} (g_{\text{word1}});
\]

(b) Function with EN/ENO (BCD\_TO\_INT\_E)

[Structured ladder/FBD]

\[
\text{BCD} \rightarrow \text{BCD} \\
g_{\text{bool3}} := \text{BCD\_TO\_INT\_E} (g_{\text{bool1}}, g_{\text{word1}}, g_{\text{int1}});
\]

(2) The program which converts BCD type data input to \( \circ \) into double word (signed) type data, and outputs the operation result from \( \bullet \):

(a) Function without EN/ENO (BCD\_TO\_DINT)

[Structured ladder/FBD]

\[
\text{BCD} \rightarrow \text{BCD} \\
g_{\text{dint1}} := \text{BCD\_TO\_DINT} (g_{\text{dword1}});
\]
5.1.34 BCD type → string type conversion

**BCD_TO_STR(_E)**

_E: With EN/ENO

---

**Function**

**Operation processing**

(1) Converts BCD type data input to (①) into string type data, and outputs the operation result from (②).

(a) When word (unsigned)/16-bit string type is specified for (①).

```
① BCD type
② String type
```

Word (unsigned)/16-bit string type

(b) When double word (unsigned)/32-bit string type is specified for (①).

```
① BCD type
② String type
```

Double word (unsigned)/32-bit string type

(2) Word (unsigned)/16-bit string type, double word (unsigned)/32-bit string type data can be specified for (①). Bit type cannot be specified.

(3) When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string.
5.1 Type Conversion Functions

5.1.34 BCD type → string type conversion

Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from 📌.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>📌</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from 📌 is undefined. In this case, create a program so that the data output from 📌 is not used.

Operation Error

These functions consist of the following common instructions.

When word (unsigned)/16-bit string type is specified for 📌: BCDDA
When double word (unsigned)/32-bit string type is specified for 📌: DBCDDA

In any of the following cases, an operation error occurs, the error flag (SM0) is turned ON, and the corresponding error code is stored to SD0.

- When word (unsigned)/16-bit string type is specified for 📌, 📌 is outside the range of 0 to 9999. (Error code: 4100)
- When double word (unsigned)/32-bit string type is specified for 📌, 📌 is outside the range of 0 to 99999999. (Error code: 4100)
- The device specified for 📌 exceeds the corresponding device range. (For Universal model QCPU and LCPU) (Error code: 4101)
5.1 Type Conversion Functions

5.1.34 BCD type → string type conversion

Program Example

(1) The program which converts word (unsigned)/16-bit string type data input to \( \odot \) into string type data, and outputs the operation result from \( \odot \).

(a) Function without EN/ENO (BCD_TO_STR)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_word1} & \quad \text{BCD} \\
\end{align*}
\]

[ST]

\[
g\_string1 := \text{BCD\_TO\_STR} (g\_word1);
\]

(b) Function with EN/ENO (BCD_TO_STR_E)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_bool3} & \quad \text{BCD\_TO\_STR\_E} \\
\text{g\_word1} & \quad \text{BCD} \\
\text{g\_bool1} & \quad \text{BCD} \\
\end{align*}
\]

[ST]

\[
g\_bool3 := \text{BCD\_TO\_STR\_E} (g\_bool1, g\_word1, g\_string1);
\]

(2) The program which converts double word (unsigned)/32-bit string type data input to \( \odot \) into string type data, and outputs the operation result from \( \odot \).

(a) Function without EN/ENO (BCD_TO_STR)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_dword1} & \quad \text{BCD} \\
\end{align*}
\]

[ST]

\[
g\_string1 := \text{BCD\_TO\_STR} (g\_dword1);
\]

(b) Function with EN/ENO (BCD_TO_STR_E)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g\_bool3} & \quad \text{BCD\_TO\_STR\_E} \\
\text{g\_dword1} & \quad \text{BCD} \\
\text{g\_bool1} & \quad \text{BCD} \\
\end{align*}
\]

[ST]

\[
g\_bool3 := \text{BCD\_TO\_STR\_E} (g\_bool1, g\_dword1, g\_string1);
\]
5.1.35 Time type → bit type conversion

**TIME_TO_BOOL(_E)**

```
TIME_TO_BOOL(_E)
```

_E: With EN/ENO

**Function**

**Operation processing**

Converts time type data input to ② into bit type data, and outputs the operation result from ②.

When the input value is 0ms, FALSE is output in bit type data.
When the input value is other than 0ms, TRUE is output in bit type data.

<table>
<thead>
<tr>
<th>Time type</th>
<th>Bit type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T#0ms</td>
<td>FALSE</td>
</tr>
<tr>
<td>T#20m34s567ms</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
5.1 Type Conversion Functions

5.1.35 Time type → bit type conversion

Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \mathcal{O} \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \mathcal{O} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE (^*1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^*1\): When FALSE is output from ENO, the data output from \( \mathcal{O} \) is undefined. In this case, create a program so that the data output from \( \mathcal{O} \) is not used.

Operation Error

No operation error occurs in the execution of the TIME_TO_BOOL(_E) function.

Program Example

The program which converts time type data input to \( \mathcal{O} \) into bit type data, and outputs the operation result from \( \mathcal{O} \).

(a) Function without EN/ENO (TIME_TO_BOOL)

[Structured ladder/FBD]

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

(b) Function with EN/ENO (TIME_TO_BOOL_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := TIME_TO_BOOL_E (g_bool1, g_time1, g_bool2);
```
5.1.36 Time type → word (signed), double word (signed) type conversion

TIME_TO_INT(_E), TIME_TO_DINT(_E)

TIME_TO_INT(_E), TIME_TO_DINT(_E)

(_E: With EN/ENO)

Structured ladder/FBD

ST

Input argument, s(_TIME): Executing condition (TRUE: Execution, FALSE: Stop) :Bit
Output argument, ENO: Execution result (TRUE: Normal execution, FALSE: Error or stop) :Bit
EN: Executing condition (TRUE: Execution, FALSE: Stop) :Bit
ENO: Execution result (TRUE: Normal execution, FALSE: Error or stop) :Bit
s: Input :Time
d: Output :Word (signed), double word (signed)

Function

Operation processing

(1) TIME_TO_INT, TIME_TO_INT_E

(a) Converts time type data input to into word (signed) type data, and outputs the operation result from .

(b) When converting to word (signed) type data, high-order 16-bit (1 word) data of time type is discarded.
(2) TIME_TO_DINT, TIME_TO_DINT_E

Converts time type data input to ① into double word (signed) type data, and outputs the operation result from ②.

\[
\begin{array}{c|c}
\text{Time type} & \text{Double word (signed) type} \\
\hline
20m34s567ms & 1234567 \\
\end{array}
\]

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from ②.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ② is undefined. In this case, create a program so that the data output from ② is not used.
5.1 Type Conversion Functions

5.1.36 Time type → word (signed), double word (signed) type conversion

Operation Error

No operation error occurs in the execution of the TIME_TO_INT(_E) and TIME_TO_DINT(_E) functions.

Program Example

(1) The program which converts time type data input to \( \text{\textbullet} \) into word (signed) type data, and outputs the operation result from \( \text{\textbullet} \).

(a) Function without EN/ENO (TIME_TO_INT)

[Structured ladder/FBD]

[ST]
\[
g_{\text{int1}} := \text{TIME_TO_INT} (g_{\text{time1}});
\]

(b) Function with EN/ENO (TIME_TO_INT_E)

[Structured ladder/FBD]

[ST]
\[
g_{\text{bool3}} := \text{TIME_TO_INT_E} (g_{\text{bool1}}, g_{\text{time1}}, g_{\text{int1}});
\]

(2) The program which converts time type data input to \( \text{\textbullet} \) into double word (signed) type data, and outputs the operation result from \( \text{\textbullet} \).

(a) Function without EN/ENO (TIME_TO_DINT)

[Structured ladder/FBD]

[ST]
\[
g_{\text{dint1}} := \text{TIME_TO_DINT} (g_{\text{time1}});
\]
5.1.37 Time type → string type conversion

TIME_TO_STR(E)

Function

Operation processing

(1) Converts time type data input to \( T \) into string type data, and outputs the operation result from \( d \).

\[
\begin{align*}
\text{Time type} & \rightarrow \text{String type} \\
T\#20\text{m}34\text{s}56\text{ms} & \rightarrow "\_\_\_\_\_\_\_1234567890\_\_\_\_\_\_\_" \\
\text{Time type} & \rightarrow \text{String type} \\
T\#-20\text{m}34\text{s}56\text{ms} & \rightarrow "\_\_\_\_\_\_\_1234567890\_\_\_\_\_\_\_"
\end{align*}
\]

(2) When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string.

(3) The operation results stored to \( d \) are as follows.

(a) For the first character, 20H (space) is stored if the BIN data is positive, and 2DH (-) is stored if it is negative.

(b) 20H (space) is stored to the left of significant figures.
**5.1 Type Conversion Functions**

5.1.37 Time type → string type conversion

**Operation result**

1. **Function without EN/ENO**
   
   An operation is executed and the operation value is output from the device.

2. **Function with EN/ENO**
   
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>Operation output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Op. ex.)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Op. stop)</td>
<td>FALSE *1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from the device is undefined. In this case, create a program so that the data output from the device is not used.

**Operation Error**

These functions consist of the following common instructions:

**TIME_TO_STR(_E):DBINDA**

In any of the following cases, an operation error occurs, the error flag (SM0) is turned ON, and the corresponding error code is stored to SDO.

- The device specified for exceeds the corresponding device range.
  
  (For Universal model QCPU and LCPU)  
  
  (Error code: 4101)

**Program Example**

The program which converts time type data input to the device into string type data, and outputs the operation result from the device.

(a) **Function without EN/ENO (TIME_TO_STR)**

[Structured ladder/FBD]

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

[ST]

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

(b) **Function with EN/ENO (TIME_TO_STR_E)**

[Structured ladder/FBD]

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

[ST]

```
g_bool3 := TIME_TO_STR_E (g_bool1, g_time1, g_string1);
```
5.1.38 Time type → word (unsigned)/16-bit string, double word (unsigned)/32-bit string type conversion

TIME_TO_WORD(_E), TIME_TO_DWORD(_E)

Function

Operation processing

1. TIME_TO_WORD, TIME_TO_WORD_E

Converts time type data input to \( \text{①} \) into word (unsigned)/16-bit string type data, and outputs the operation result from \( \text{②} \).

\[
\begin{align*}
\text{①} & : \text{Time type} \\
\text{②} & : \text{Word (unsigned)/16-bit string type}
\end{align*}
\]

2. When converting to word (unsigned)/16-bit string type data, high-order 16-bit (1 word) data is discarded.
(3) TIME_TO_DWORD, TIME_TO_DWORD_E

Converts time type data input to ① into double word (unsigned)/32-bit string type data, and outputs the operation result from ③.

<table>
<thead>
<tr>
<th>Time type</th>
<th>Double word (unsigned)/32-bit string type</th>
</tr>
</thead>
<tbody>
<tr>
<td>T#20m34s567ms</td>
<td>12345678</td>
</tr>
</tbody>
</table>

Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
Operation Error

No operation error occurs in the execution of the TIME_TO_WORD(_E) and TIME_TO_DWORD(_E) functions.

Program Example

(1) The program which converts time type data input to \( \mathbb{O} \) into word (unsigned)/16-bit string type data, and outputs the operation result from \( \mathbb{O} \).

(a) Function without EN/ENO (TIME_TO_WORD)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g_word1 := TIME_TO_WORD (g_time1);}
\end{align*}
\]

(b) Function with EN/ENO (TIME_TO_WORD_E)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g_bool3 := TIME_TO_WORD_E (g_bool1, g_time1, g_word1);} 
\end{align*}
\]

(2) The program which converts time type data input to \( \mathbb{O} \) into double word (unsigned)/32-bit string type data, and outputs the operation result from \( \mathbb{O} \).

(a) Function without EN/ENO (TIME_TO_DWORD)

[Structured ladder/FBD]

\[
\begin{align*}
\text{g_dword1 := TIME_TO_DWORD (g_time1);}
\end{align*}
\]
5.1.39 Bit array → word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, double word (unsigned)/32-bit string type conversion

BITARR_TO_INT(\_E), BITARR_TO_DINT(\_E)

BITARR_TO_INT(\_E), BITARR_TO_DINT(\_E)

\_E: With EN/ENO

Function

Operation processing

(1) BITARR_TO_INT, BITARR_TO_INT_E

Converts number of bits specified for n starting from the bit array element input to \( \Theta \) into word (signed) type or word (unsigned)/16-bit string type data, and outputs the operation result from \( \Theta \).

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

0 is set for the output bits higher than the specified number of bits.

(2) BITARR_TO_DINT, BITARR_TO_DINT_E

Converts number of bits specified for n starting from the bit array element input to \( \Theta \) into double word (signed) type or double word (unsigned)/32-bit string type data, and outputs the operation result from \( \Theta \).

Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified for n.

0 is set for the output bits higher than the specified number of bits.
Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from $\circ$.

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>$\odot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>*1 Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from $\odot$ is undefined. In this case, create a program so that the data output from $\odot$ is not used.

Operation Error

No operation error occurs in the execution of the BITARR_TO_INT(_E) and BITARR_TO_DINT(_E) functions.

Program Example

The program which converts 8 bits from 0 of bit array input to $\odot$ into word (signed) type data, and outputs the operation result from $\odot$.

(a) Function without EN/ENO (BITARR_TO_INT)
[Structured ladder/FBD]

```
[ST]
g_int1 := BITARR_TO_INT(g_bool4[0], 8);
```

(b) Function with EN/ENO (BITARR_TO_INT_E)
[Structured ladder/FBD]

```
[ST]
g_bool2 := BITARR_TO_INT_E(g_bool1, g_bool4[0], 8, g_int1);
```
5.1.40 Word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, double word (unsigned)/32-bit string type → bit array conversion

`INT_TO_BITARR(_E), DINT_TO_BITARR(_E)`

**Operation processing**

1. **INT_TO_BITARR, INT_TO_BITARR_E**
   - Outputs low-order n bits of word (signed) type or word (unsigned)/16-bit string type data specified for 4 to 16.
   - Only a constant 4, 8, 12 or 16 can be specified for n.
   - The output bits higher than the specified number of bits do not change.

2. **DINT_TO_BITARR, DINT_TO_BITARR_E**
   - Outputs low-order n bits of double word (signed) type or double word (unsigned)/32-bit string type data specified for 4 to 32.
   - Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified for n.
   - The output bits higher than the specified number of bits do not change.
5.1 Type Conversion Functions

5.1.40 Word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, double word (unsigned)/32-bit string type → bit array conversion

**Operation result**

1. **Function without EN/ENO**
   - An operation is executed and the operation value is output from ③.

2. **Function with EN/ENO**
   - The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>*1 Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.

**Operation Error**

No operation error occurs in the execution of the INT_TO_BITARR(_E) and DINT_TO_BITARR(_E) functions.

**Program Example**

The program which outputs low-order 4 bits of word (signed) type data input to ② to ③.

(a) **Function without EN/ENO (INT_TO_BITARR)**

[Structured ladder/FBD]

[ST]

```plaintext
g_bool4[0] := INT_TO_BITARR(g_int1, 4);
```

(b) **Function with EN/ENO (INT_TO_BITARR_E)**

[Structured ladder/FBD]

[ST]

```plaintext
g_bool2 := INT_TO_BITARR_E(g_bool1, g_int1, 4, g_bool4[0]);
```
5.1.41 Bit array copy

**CPY_BITARR(_E)**

_&_E: With EN/ENO

---

### Structure ladder/FBD

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation processing</th>
<th>Operation result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPY_BITARR(_E)</strong></td>
<td>Outputs n bits of bit array input to to .</td>
<td></td>
</tr>
</tbody>
</table>

#### Operation processing

Outputs n bits of bit array input to to .

#### Operation result

1. **Function without EN/ENO**
   
   An operation is executed and the operation value is output from .

2. **Function with EN/ENO**
   
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE *1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from is undefined. In this case, create a program so that the data output from is not used.
Operation Error

No operation error occurs in the execution of the CPY_BITARR(E) function.

Program Example

The program which outputs 12 bits from num1 element of bit string input to ③ to num2 and the following bits of ③.

(a) Function without EN/ENO (CPY_BITARR)

[Structured ladder/FBD]

[g_bool5[num2] := CPY_BITARR(g_bool4[num1], 12);

(b) Function with EN/ENO (CPY_BITARR_E)

[Structured ladder/FBD]

[g_bool2 := CPY_BITARR_E(g_bool1, g_bool4[num1], 12, g_bool5[num2]);
5.1.42 Specified bit read of word (signed) type data

GET_BIT_OF_INT(_E)

_FUNCTION_

GET_BIT_OF_INT(_E)

_FUNCTION_

_GET_BIT_OF_INT(_E)_

_E: With EN/ENO

FUNCTION

Operation processing

Reads a value of nth bit of _s_, and outputs the operation result from _d_.

Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from _d_.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th><em>d</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from _d_ is undefined. In this case, create a program so that the data output from _d_ is not used.
**Operation Error**

No operation error occurs in the execution of the GET_BIT_OF_INT(_E) function.

**Program Example**

The program which reads a value of 5th bit of data input to , and outputs the operation result from .

(a) Function without EN/ENO (GET_BIT_OF_INT)

[Structured ladder/FBD]

```
[ST]
g_bool3 := GET_BIT_OF_INT(g_int1, 5);
```

(b) Function with EN/ENO (GET_BIT_OF_INT_E)

[Structured ladder/FBD]

```
[ST]
g_bool2 := GET_BIT_OF_INT_E(g_bool1, g_int1, 5, g_bool3);
```
5.1.43 Specified bit write of word (signed) type data

**SET_BIT_OF_INT(\_E)**

\_E: With EN/ENO

---

**Function**

**Operation processing**

Writes a value specified for \( \circ \) to the nth bit of \( \bigcirc \).

**Operation result**

1. Function without EN/ENO
   An operation is executed and the operation value is output from \( \bigcirc \).

2. Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \bigcirc )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE *1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \bigcirc \) is undefined. In this case, create a program so that the data output from \( \bigcirc \) is not used.
**Operation Error**

No operation error occurs in the execution of the SET_BIT_OF_INT(_E) function.

**Program Example**

The program which writes a value specified for $\odot$ to the 3rd bit of $\odot$.

(a) Function without EN/ENO (SET_BIT_OF_INT)

[Structured ladder/FBD]

```
[ST]
g_int3 := SET_BIT_OF_INT(g_bool1, 3);
```

(b) Function with EN/ENO (SET_BIT_OF_INT_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := SET_BIT_OF_INT_E(g_bool2, g_bool1, 3, g_int3);
```
5.1.44 Specified bit copy of word (signed) type data

**Function**

**Operation processing**
Copies a value of (n1)th bit of input  to the (n2)th bit of output .

**Operation result**

1. Function without EN/ENO
   An operation is executed and the operation value is output from .

2. Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE *1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from  is undefined. In this case, create a program so that the data output from  is not used.
**Operation Error**

No operation error occurs in the execution of the CPY_BIT_OF_INT(_E) function.

**Program Example**

The program which writes a value of 5th bit of \( \circ \) to the 3rd bit of \( \circ \).

(a) Function without EN/ENO (CPY_BIT_OF_INT)

[Structured ladder/FBD]

```
[ST]
  g_int3 := CPY_BIT_OF_INT(g_int1, 5, 3);
```

(b) Function with EN/ENO (CPY_BIT_OF_INT_E)

[Structured ladder/FBD]

```
[ST]
  g_bool3 := CPY_BIT_OF_INT_E(g_bool2, g_int1, 5, 3, g_int3);
```
5.1.45 Nonessential type conversion

GET_BOOL_ADDR, GET_INT_ADDR, GET_WORD_ADDR

<table>
<thead>
<tr>
<th>Function name</th>
<th>Input data type</th>
<th>Output data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_BOOL_ADDR</td>
<td>Bit Array of bit</td>
<td>Bit</td>
</tr>
<tr>
<td>GET_INT_ADDR</td>
<td>Word (signed) Double word (signed) Word (unsigned)/16-bit string Single-precision real number String Time type</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>GET_WORD_ADDR</td>
<td>Array of word (signed) Array of double word (signed) Array of word (unsigned)/16-bit string Array of double word (unsigned)/32-bit string Array of real number Array of time type</td>
<td>Word (unsigned)/16-bit string</td>
</tr>
</tbody>
</table>

(2) Rounding error may occur when specifying the input value to GET_BOOL_ADDR, GET_INT_ADDR, GET_WORD_ADDR by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
5.1.45 Nonessential type conversion

**Operation result**

An operation is executed and the operation value is output from ③.

**Operation Error**

No operation error occurs in the execution of the GET_BOOL_ADDR, GET_INT_ADDR, and GET_WORD_ADDR functions.

**Program Example**

The program which directly handles 32-bit input variable Var_D10 as 16-bit input data without the type conversion.

GET_INT_ADDR

[Structured ladder/FBD]

[ST]

Var_D100 := GET_INT_ADDR(Var_D10);
5.2 Standard Functions of One Numeric Variable

5.2.1 Absolute value

ABS(_E)

Function

Operation processing

1. Outputs the absolute value of word (signed), double word (signed), single-precision real or double-precision real type data input to \( \odot \) from \( \odot \) in the same data type as that of \( \odot \). Assuming that the input value is \( A \) and the operation output value is \( B \), the relationship is expressed by the following equality.

\[ B = |A| \]

2. The value to be input to \( \odot \) is word (signed), double word (signed), single-precision real or double-precision real type data.

3. When the data type of \( \odot \) is word (signed) type and the input value is -32768, -32768 is output from \( \odot \).

When the data type of \( \odot \) is double word (signed) type and the input value is -2147483648, -2147483648 is output from \( \odot \).

(No operation error occurs. In case of ABS_E, TRUE is output from ENO.)
(4) Rounding error may occur when specifying single-precision real or double-precision real type data to \( \oplus \) by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

**Operation result**

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \oplus \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN (Operation execution)</th>
<th>ENO</th>
<th>( \oplus )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^*1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^*1\): When FALSE is output from ENO, the data output from \( \oplus \) is undefined. In this case, create a program so that the data output from \( \oplus \) is not used.
Operation Error

No operation error occurs in the execution of the ABS(_E) function.

Program Example

(1) The program which outputs the absolute value of word (signed) type data input to ③ from ② in the same data type as that of ①.
   (a) Function without EN/ENO (ABS)

   [Structured ladder/FBD]

   
   
   [ST]
   \[ g\text{-int2} := \text{ABS}(g\text{-int1}); \]

   (b) Function with EN/ENO (ABS_E)

   [Structured ladder/FBD]

   
   
   [ST]
   \[ g\text{-bool3} := \text{ABS}_E(g\text{-bool1}, g\text{-int1}, g\text{-int2}); \]

(2) The program which outputs the absolute value of double word (signed) type data input to ⑤ from ④ in the same data type as that of ③.
   (a) Function without EN/ENO (ABS)

   [Structured ladder/FBD]

   
   
   [ST]
   \[ g\text{-dint2} := \text{ABS}(g\text{-dint1}); \]

   (b) Function with EN/ENO (ABS_E)

   [Structured ladder/FBD]

   
   
   [ST]
   \[ g\text{-bool2} := \text{ABS}_E(g\text{-bool1}, g\text{-dint1}, g\text{-dint2}); \]
5.2 Standard Functions of One Numeric Variable

5.2.1 Absolute value

(3) The program which outputs the absolute value of single-precision real type data input to \( g_1 \) from \( g_2 \) in the same data type as that of \( g_2 \).

(a) Function without EN/ENO (ABS)

[Structured ladder/FBD]

\[
g_{\text{real2}} := \text{ABS}(g_{\text{real1}});
\]

(b) Function with EN/ENO (ABS_E)

[Structured ladder/FBD]

\[
g_{\text{bool2}} := \text{ABS}_E(g_{\text{bool1}}, g_{\text{real1}}, g_{\text{real2}});
\]

(4) The program which outputs the absolute value of double-precision real type data input to \( g_1 \) from \( g_2 \) in the same data type as that of \( g_2 \).

(a) Function without EN/ENO (ABS)

[Structured ladder/FBD]

\[
g_{\text{lreal2}} := \text{ABS}(g_{\text{lreal1}});
\]

(b) Function with EN/ENO (ABS_E)

[Structured ladder/FBD]

\[
g_{\text{bool2}} := \text{ABS}_E(g_{\text{bool1}}, g_{\text{lreal1}}, g_{\text{lreal2}});
\]
5.3 Standard Arithmetic Functions

5.3.1 Addition

ADD_E

**Function**

**Operation processing**

(1) Performs addition \( \sum \) on word (signed), double word (signed), single-precision real or double-precision real type data input to \( \oplus \) to \( \oplus \), and outputs the operation result from \( \ominus \) in the same data type as that of \( \oplus \) to \( \ominus \).

(Example) Word (signed) type data

\[
\begin{align*}
1234 & \oplus 5678 \\
\text{Word (signed) type} & \to 6912 \\
\text{Word (signed) type} & \to \ominus 
\end{align*}
\]

(2) The values to be input to \( \oplus \) to \( \ominus \) are word (signed), double word (signed), single-precision real or double-precision real type data.

(3) The number of pins for \( \ominus \) can be changed in the range from 2 to 28.
5.3 Standard Arithmetic Functions

5.3.1 Addition

(4) If an underflow/overflow occurs in the operation result, data is output from \( \circ \) as follows.

(a) Word (signed) type data
   No operation error occurs even if an underflow/overflow occurs.
   In case of ADD_E, TRUE is output from ENO.
   \[
   32767 + 2 = -32767 \\
   (7FFFH) (0002H) (8001H)
   \]
   Since the highest-order bit is 1, the result value is negative.

   \[
   -32767 + (-2) = 32766 \\
   (8000H) (FFFFH) (7FFEH)
   \]
   Since the highest-order bit is 0, the result value is positive.

(b) Double word (signed) type data
   No operation error occurs even if an underflow/overflow occurs.
   In case of ADD_E, TRUE is output from ENO.
   \[
   2147483647 + 2 = -2147483647 \\
   (7FFFFFFH) (0002H) (80000001H)
   \]
   Since the highest-order bit is 1, the result value is negative.

   \[
   -2147483648 + (-2) = 2147483646 \\
   (80000000H) (FFFFH) (7FFFFFFEH)
   \]
   Since the highest-order bit is 0, the result value is positive.

(5) Rounding error may occur when specifying single-precision real or double-precision real type data to \( \circ \) through \( \circ \) by programming tool.
   For the precautions on setting the input value by the programming tool, refer to
   MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

Operation result

(1) Function without EN/ENO
   An operation is executed and the operation value is output from \( \circ \).

(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \circ \) is undefined.
   In this case, create a program so that the data output from \( \circ \) is not used.
5.3 Standard Arithmetic Functions

5.3.1 Addition

Operation Error

No operation error occurs in the execution of the ADD_E function.

Program Example

The program which performs addition (\( \oplus \)) on double word (signed) type data input to \( A \) and \( B \), and outputs the operation result from \( C \) in the same data type as that of \( A \) and \( B \).

[Structured ladder/FBD]

[ST]

\[
g_{bool3} := \text{ADD}_E(g_{bool1}, g_{dint1}, g_{dint2}, g_{dint3});
\]
5.3.2 Multiplication

MUL_E

Function

Operation processing

1. Performs multiplication \((\circlearrowleft \times \circlearrowleft \times \cdots \times \circlearrowright)\) on word (signed), double word (signed), single-precision real or double-precision real type data input to \(\circlearrowright\) to \(\circlearrowright\), and outputs the operation result from \(\circlearrowright\) in the same data type as that of \(\circlearrowleft\) to \(\circlearrowright\).

(Example) Word (signed) type data

\[
\begin{array}{ccc}
\text{Word (signed) type} & \times & \text{Word (signed) type} \\
100 & \times & 15 \\
\end{array}
\Rightarrow \begin{array}{c}
\text{Word (signed) type} \\
1500 \\
\end{array}
\]

2. The values to be input to \(\circlearrowright\) to \(\circlearrowright\) are word (signed), double word (signed), single-precision real or double-precision real type data.

3. The number of pins for \(\circlearrowleft\) can be changed in the range from 2 to 28.
(4) If an underflow/overflow occurs in the operation result, data is output from \( \text{d} \) as follows.

(a) Word (signed) type data
   
   No operation error occurs even if an underflow/overflow occurs.
   
   In case of MUL_E, TRUE is output from ENO.
   
   Even if the operation result exceeds the word (signed) type data range, data is output in
   word (signed) type.
   
   (Although the operation result is 32-bit data, data is output in word (signed) type with the
   high-order 16 bits discarded.)
   
   If the operation result exceeds the word (signed) type data range, convert the input
   values to the double word (signed) type data by the INT_TO_DINT function and perform
   the operation using the converted data.

(b) Double word (signed) type data
   
   No operation error occurs even if an underflow/overflow occurs.
   
   In case of MUL_E, TRUE is output from ENO.
   
   Even if the operation result exceeds the double word (signed) data range, data is output
   in double word (signed) type.
   
   (Although the operation result is 64-bit data, data is output in double word (signed) type
   with the high-order 32 bits discarded.)
   
   If the operation result exceeds the double word (signed) type data range, convert the
   input values to the single-precision real type data by the DINT_TO_REAL function and
   perform the operation using the converted data.

(5) Rounding error may occur when specifying single-precision real or double-precision real
   type data to \( \text{d} \) through \( \text{s} \) by programming tool.
   
   For the precautions on setting the input value by the programming tool, refer to
   MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

**Operation result**

(1) Function without EN/ENO
   
   An operation is executed and the operation value is output from \( \text{d} \).

(2) Function with EN/ENO
   
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \text{d} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^1\): When FALSE is output from ENO, the data output from \( \text{d} \) is undefined.
   
   In this case, create a program so that the data output from \( \text{d} \) is not used.

**POINT**

If the operation result exceeds the data type range, convert the data type of the
input data before the operation.
Operation Error

No operation error occurs in the execution of the MUL_E function.

Program Example

The program which performs multiplication \((s1 \times s2)\) on double word (signed) type data input to \(s1\) and \(s2\), and outputs the operation result from \(s1\) in the same data type as that of \(s1\) and \(s2\).

[Structured ladder/FBD]

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

[ST]

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

**Function**

**Operation processing**

(1) Performs subtraction \( \text{①} - \text{②} \) on word (signed), double word (signed), single-precision real or double-precision real type data input to \( \text{①} \) and \( \text{②} \), and outputs the operation result from \( \text{③} \) in the same data type as that of \( \text{①} \) and \( \text{②} \).

(Example) Word (signed) type data

\[
\begin{align*}
\text{①} & : 12345 \\
\text{②} & : 6789 \\
\text{③} & : 5556
\end{align*}
\]

(2) The values to be input to \( \text{①} \) and \( \text{②} \) are word (signed), double word (signed), single-precision real or double-precision real type data.
(3) If an underflow/overflow occurs in the operation result, data is output from ③ as follows.

(a) Word (signed) type data
No operation error occurs even if an underflow/overflow occurs.
In case of SUB_E, TRUE is output from ENO.

\[ 32767 - (-2) = 32767 \]
\[ (7FFFFH)(FFFEH)(8001H) \]
Since the highest-order bit is 1, the result value is negative.

\[ -32767 - 2 = 32766 \]
\[ (8000H)(0002H)(7FFEH) \]
Since the highest-order bit is 0, the result value is positive.

(b) Double word (signed) type data
No operation error occurs even if an underflow/overflow occurs.
In case of SUB_E, TRUE is output from ENO.

\[ 2147483647 - (-2) = 2147483647 \]
\[ (7FFFFFFFH)(FFFEH)(80000001H) \]
Since the highest-order bit is 1, the result value is negative.

\[ -2147483648 - 2 = 2147483646 \]
\[ (80000000H)(0002H)(7FFFFFFEH) \]
Since the highest-order bit is 0, the result value is positive.

(4) Rounding error may occur when specifying single-precision real or double-precision real type data to ③ by programming tool.
For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

### Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined.
In this case, create a program so that the data output from ③ is not used.
### Operation Error

No operation error occurs in the execution of the SUB_E function.

### Program Example

The program which performs subtraction \((\overline{\circ} - \overline{\circ})\) on double word (signed) type data input to \(\overline{\circ}\) and \(\overline{\circ}\), and outputs the operation result from \(\overline{\circ}\) in the same data type as that of \(\overline{\circ}\) and \(\overline{\circ}\).

[Structured ladder/FBD]

![Structured ladder/FBD diagram]

[ST]

\[
g_{\text{bool}3} := \text{SUB}_E(g_{\text{bool}1}, g_{\text{dint}1}, g_{\text{dint}2}, g_{\text{dint}3});
\]
5.3.4 Division

**DIV_E**

(_E: With EN/ENO)

---

**Operation processing**

1. Performs division (\( \div \)) on word (signed), double word (signed), single-precision real or double-precision real type data input to \( \odot \) and \( \odot \), and outputs the quotient of the operation result from \( \odot \) in the same data type as that of \( \odot \) and \( \odot \).

   **Example** Word (signed) type data

   \[
   \begin{array}{c}
   5 \div 2 = 2.5
   \\
   \text{Word (signed) type}
   \\
   \text{Word (signed) type}
   \\
   \text{(Quotient)}
   \\
   \text{(Remainder)}
   \\
   \end{array}
   \]

2. The values to be input to \( \odot \) and \( \odot \) are word (signed), double word (signed), single-precision real or double-precision real type data.

   (The value to be input to \( \odot \) must be other than 0.)

3. Rounding error may occur when specifying single-precision real or double-precision real type data to \( \odot \), \( \odot \) by programming tool.

   For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
Operation result

(1) Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>Operation output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>Operation output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
5.3 Standard Arithmetic Functions

5.3.4 Division

**Operation Error**

An operation error occurs in the following case.

- The value to be input to \( \div \) is 0. (Division by 0) (Error code: 4100)

**Program Example**

The program which performs division \((\div)\) on double word (signed) type data input to \( \div \) and \( \div \), and outputs the quotient of the operation result from \( \div \) in the same data type as that of \( \div \) and \( \div \).

**Structured ladder/FBD**

```
\[
g_\text{bool3} := \text{DIV}_E(g_\text{bool1}, g_\text{dint1}, g_\text{dint2}, g_\text{dint3});
\]
```

**ST**

```
g_\text{bool3} := \text{DIV}_E(g_\text{bool1}, g_\text{dint1}, g_\text{dint2}, g_\text{dint3});
```
5.3.5 Modules operation

MOD(_E)

_E: With EN/ENO

Function

Operation processing

(1) Performs division \((\div)\) on word (signed) or double word (signed) type data input to \(\textcircled{1}\) and \(\textcircled{2}\), and outputs the remainder of the operation result from \(\textcircled{3}\) in the same data type as that of \(\textcircled{1}\) and \(\textcircled{2}\).

(Example) Word (signed) type data

\[
\begin{array}{c}
\textcircled{1} 5 \\
\text{Word (signed) type}
\end{array}
\div
\begin{array}{c}
\textcircled{2} 2 \\
\text{Word (signed) type}
\end{array}
\rightarrow
\begin{array}{c}
\textcircled{4} 2 \\
\text{No output}
\end{array}
\begin{array}{c}
\textcircled{3} 1 \\
\text{Word (signed) type}
\end{array}
\]

(2) The values to be input to \(\textcircled{1}\) and \(\textcircled{2}\) are word (signed) or double word (signed) type data. (Note that the value to be input to \(\textcircled{2}\) must be other than 0.)
### 5.3 Standard Arithmetic Functions

#### 5.3.5 Modules operation

**Operation result**

(1) Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation error</td>
<td>Operation output value</td>
</tr>
<tr>
<td>Operation error</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
### Operation Error

An operation error occurs in the following case.

- The value to be input to \( \div \) is 0. (Division by 0) (Error code: 4100)

### Program Example

The program which performs division (\( \div \)) on double word (signed) type data input to \( \div \) and \( \div \), and outputs the remainder of the operation result from \( \div \) in the same data type as that of \( \div \) and \( \div \).

(a) Function without EN/ENO (MOD)

[Structured ladder/FBD]

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

(b) Function with EN/ENO (MOD_E)

[Structured ladder/FBD]

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

EXPT(_E)

_E: With EN/ENO

Function

Operation processing

1. Performs exponentiation \( \times \) on single-precision real or double-precision real type data input to \( \times \) and word (signed), double word (signed), single-precision real or double-precision real type data input to \( \times \), and outputs the operation result from \( \times \).

\[
\begin{array}{c|c|c|c|c}
\text{Input argument,} & \text{EN:} & \text{Input condition (TRUE: Execution, FALSE: Stop)} & \text{Input} & \text{Output} \\
\text{s1(_IN1):} & \text{EN:} & \text{Input condition} & \text{s1} & \text{d} \\
\text{s2(_IN2):} & \text{ENO:} & \text{Execution result (TRUE: Normal, FALSE: Error)} & \text{s2} & \text{ENO} \\
\end{array}
\]

2. Rounding error may occur when specifying single-precision real or double-precision real type data to \( \times, \) \( \times \) by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
Operation result

(1) Function without EN/ENO
   An operation is executed and the operation value is output from ②.

(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th></th>
<th>EN</th>
<th>ENO</th>
<th>②</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td></td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE*1</td>
<td></td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ② is undefined. In this case, create a program so that the data output from ② is not used.

Operation Error

These functions consist of the following common instructions.
When ① is single-precision real number, ② is word (signed): LOG, FLT
When ① is single-precision real number, ② is double word (signed): LOG, DFLT
When ① is single-precision real number, ② is single-precision real number: LOG
When ① is single-precision real number, ② is double-precision real number: LOGD, DFLT
When ① is double-precision real number, ② is word (signed): LOGD
When ① is double-precision real number, ② is double word (signed): LOGD, FLTD
When ① is double-precision real number, ② is single-precision real number: LOGD, DFLT
When ① is double-precision real number, ② is double-precision real number: LOGD

For details of an error which occurs when the function is executed, refer to MELSEC-Q/L Structured Programming Manual (Common Instructions).
Program Example

The program which performs exponentiation and outputs the operation result from $\otimes$ in the same data type as that of $\oplus$ and $\otimes$.

(a) Function without EN/ENO (EXPT)

[Structured ladder/FBD]

[ST]

\[ g_{\text{real2}} := \text{EXPT}(g_{\text{real1}}, g_{\text{int1}}); \]

(b) Function with EN/ENO (EXPT_E)

[Structured ladder/FBD]

[ST]

\[ g_{\text{bool3}} := \text{EXPT}_E(g_{\text{bool1}}, g_{\text{real1}}, g_{\text{int1}}, g_{\text{real2}}); \]
5.3.7 Move operation

MOVE(_E)

_E: With EN/ENO

Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop)

s1(_IN1): Input

Output argument, ENO: Execution result (TRUE: Normal, FALSE: Error)

d: Output

Structured ladder/FBD

ST

ENO:=MOVE_E(EN, s, d);

Indicates any of the following functions.
MOVE
MOVE_E

:Bit
:ANY
:Bit
:ANY
Function

Operation processing

1. Moves the data input for \( a \) from \( b \) in the same data type as that of \( c \).

2. The values to be specified to \( a \) and \( b \) are word (signed), double word (signed), word(unsigned)/16-bit string, double word (unsigned)/32-bit string, single-precision real, double-precision real, string, or time type data. Only the same data type can be specified for \( a \) and \( b \).

3. Rounding error may occur when specifying single-precision real or double-precision real type data to \( a \) by programming tool.

For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \Box \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \Box )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^1\): When FALSE is output from ENO, the data output from \( \Box \) is undefined.
In this case, create a program so that the data output from \( \Box \) is not used.
### Operation Error

No operation error occurs in the execution of the MOVE(_E) function.

### Program Example

The program which moves the word (signed) type data input to \( \odot \) to \( \oplus \).

(a) Function without EN/ENO (MOVE)

[Structured ladder/FBD]

[ST]

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

(b) Function with EN/ENO (MOVE_E)

[Structured ladder/FBD]

[ST]

```
g_bool3 := MOVE_E(g_bool1, g_int1, g_int2);
```
5.4 Standard Bitwise Boolean Functions

5.4.1 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT

\[ \text{AND}_E, \text{OR}_E, \text{XOR}_E, \text{NOT}_E(\_E) \]

- **Function**

**Operation processing**

1. **AND_E**
   - (a) Performs Boolean AND on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to $s_1$ to $s_28$($\_\text{IN1}$):input (s1 only for NOT(\_E)) by bit by bit, and outputs the operation result from $d$ in the same data type as that of $s_1$ to $s_28$.
   - (Example) Word (unsigned)/16-bit string type data
     
     \[
     s_1: \begin{array}{cccccccccccc}
     1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
     \end{array}
     \]
     
     AND
     
     \[
     s_2: \begin{array}{cccccccccccc}
     0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
     \end{array}
     \]
     
     Boolean AND
     
     \[
     d: \begin{array}{cccccccccccc}
     0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
     \end{array}
     \]
5.4 Standard Bitwise Boolean Functions

5.4.1 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT

(b) The number of pins of variable 's' can be changed in the range from 2 to 28.

(2) OR_E

(a) Performs Boolean OR on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to \( \mathcal{E} \) to \( \mathcal{E} \) bit by bit, and outputs the operation result from \( \mathcal{E} \) in the same data type as that of \( \mathcal{E} \) to \( \mathcal{E} \).

(Example) Word (unsigned)/16-bit string type data

\[
\begin{array}{cccccccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\( \text{OR} \)

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}
\]

Boolean OR

\[
\begin{array}{cccccccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}
\]

(b) The number of pins of variable 's' can be changed in the range from 2 to 28.

(3) XOR_E

(a) Performs Boolean exclusive OR on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to \( \mathcal{E} \) to \( \mathcal{E} \) bit by bit, and outputs the operation result from \( \mathcal{E} \) in the same data type as that of \( \mathcal{E} \) to \( \mathcal{E} \).

(Example) Word (unsigned)/16-bit string type data

\[
\begin{array}{cccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

\( \text{OR} \)

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}
\]

Boolean exclusive OR

\[
\begin{array}{cccccccccccc}
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}
\]

(b) The number of pins of variable 's' can be changed in the range from 2 to 28.

(c) When three or more variables 's' exist, XOR is performed between \( \mathcal{E} \) and \( \mathcal{E} \) first, and XOR is successively performed between the result and \( \mathcal{E} \). When the expression includes \( \mathcal{E} \), XOR is performed between the result of XOR with \( \mathcal{E} \) and \( \mathcal{E} \).

In this manner, XOR is repeated by the number of variables 's' in the order with \( \mathcal{E} \), \( \mathcal{E} \) and so on.

(Example) Bit type data

\[
\begin{array}{c|c|c|c}
\text{For 3 INs} & \text{FALSE} & \text{TRUE} & \text{FALSE} \\
\text{XOR} & \text{TRUE} & \text{TRUE} & \text{TRUE} \\
\text{Result} & \text{FALSE} & \text{TRUE} & \text{FALSE} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{For 4 INs} & \text{TRUE} & \text{FALSE} & \text{TRUE} \end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{For 5 INs} & \text{FALSE} & \text{TRUE} & \text{FALSE} \end{array}
\]

Hereafter, XOR is repeated by the number of s.
(4) NOT, NOT_E

(a) Performs Boolean NOT on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to \( s_1 \) bit by bit, and outputs the operation result from \( s_2 \) in the same data type as that of \( s_1 \).

(Example) Word (unsigned)/16-bit string type data

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\]

NOT

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

(b) The value to be input to variables \( s_3 \) to \( s_6 \) is bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data.

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( s_7 \).

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( s_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE (^1)</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

\(^1\): When FALSE is output from ENO, the data output from \( s_7 \) is undefined. In this case, create a program so that the data output from \( s_7 \) is not used.
**Operation Error**

No operation error occurs in the execution of the AND_E, OR_E, XOR_E, and NOT(_E) functions.

**Program Example**

(1) The program which performs Boolean AND on bit, word (unsigned)/16-bit string type data input to @ to @ bit by bit, and outputs the operation result from @ in the same data type as that of @ to @.

[Structured ladder/FBD]

[ST]

\[
g_{\text{bool2}} := \text{AND}_E (g_{\text{bool1}}, g_{\text{word1}}, g_{\text{word2}}, g_{\text{word3}});
\]
5.4 Standard Bitwise Boolean Functions

5.4.1 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT

(2) The program which performs Boolean OR on bit, word (unsigned)/16-bit string type data input to \( g_1 \) to \( g_3 \) bit by bit, and outputs the operation result from \( g_3 \) in the same data type as that of \( g_3 \) to \( g_3 \).

[Structured ladder/FBD]

[ST]
\[
g_{\text{bool}2} := \text{OR}_E(g_{\text{bool}1}, g_{\text{word}1}, g_{\text{word}2}, g_{\text{word}3});
\]

(3) The program which performs Boolean XOR on bit, word (unsigned)/16-bit string type data input to \( g_1 \) to \( g_3 \) bit by bit, and outputs the operation result from \( g_3 \) in the same data type as that of \( g_3 \) to \( g_3 \).

[Structured ladder/FBD]

[ST]
\[
g_{\text{bool}2} := \text{XOR}_E(g_{\text{bool}1}, g_{\text{word}1}, g_{\text{word}2}, g_{\text{word}3});
\]

(4) The program which performs Boolean NOT on bit, word (unsigned)/16-bit string type data input to \( g_3 \) bit by bit, and outputs the operation result from \( g_3 \) in the same data type as that of \( g_3 \).

(a) Function without EN/ENO (NOT)

[Structured ladder/FBD]

[ST]
\[
g_{\text{word}2} := \text{NOT} (g_{\text{word}1});
\]

(b) Function with EN/ENO (NOT_E)

[Structured ladder/FBD]

[ST]
\[
g_{\text{bool}2} := \text{NOT}_E (g_{\text{bool}1}, g_{\text{word}1}, g_{\text{word}2});
\]
5.5 Standard Selection Functions

5.5.1 Selection

SEL(_E)

Function

Operation processing

(1) Selects either of values input to  and  according to the bit type data input to , and outputs the operation result from  in the same data type as that of  and .

When the input value of  is FALSE, the value input to  is output from  .

When the input value of  is TRUE, the value input to  is output from  .

(Example)  and  are word (signed) type data

SEL(_E) _E: With EN/ENO

Structured ladder/FBD

ST

Indicates any of the following functions.

SEL  SEL_E
(2) The input value to $\text{SEL}(_E)$ is data value of bit type.

(3) The input value to $\text{SEL}(_E)$ is data value of bit type/word (signed) type/word (unsigned) type/16-bit string type/double word (signed) type/double word (unsigned) type/32-bit string type/single-precision real number type/double-precision real number type/string type/time type/structured data type/array type.

(4) Rounding error may occur when specifying single-precision real or double-precision real type data to $\text{SEL}(_E)$ by programming tool.
   For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

**Operation result**

(1) Function without EN/ENO
   An operation is executed and the operation value is output from $\text{SEL}(_E)$.

(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>$\text{SEL}(_E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE$^1$</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from $\text{SEL}(_E)$ is undefined. In this case, create a program so that the data output from $\text{SEL}(_E)$ is not used.
5.5 Standard Selection Functions

5.5.1 Selection

**Operation Error**

No operation error occurs in the execution of the SEL(_E) function.

**Program Example**

The program which selects either of values input to \(\text{SEL} \) and \(\text{SEL} \) according to the value input to \(\text{SEL} \), and outputs the operation result from \(\text{SEL} \) in the same data type as that of \(\text{SEL} \) and \(\text{SEL} \).

(a) Function without EN/ENO (SEL)

[Structured ladder/FBD]

\[
g_{\text{word3}} := \text{SEL} (g_{\text{bool1}}, g_{\text{word1}}, g_{\text{word2}});
\]

(b) Function with EN/ENO (SEL_E)

[Structured ladder/FBD]

\[
g_{\text{bool3}} := \text{SEL}_E (g_{\text{bool1}}, g_{\text{bool2}}, g_{\text{word1}}, g_{\text{word2}}, g_{\text{word3}});
\]
5.5.2 Maximum/Minimum selection

MAXIMUM(_E), MINIMUM(_E)

MAXIMUM(_E)
MINIMUM(_E)

_E: With EN/ENO

Function

Operation processing

(1) MAXIMUM, MAXIMUM_E

Selects the maximum value to be output among the bit, word (signed), double word (signed), word(unsigned)/16-bit string, double word(unsigned)/32-bit string, single-precision real, double-precision real, string, or time type data input to _E to _E, and outputs the operation result from _E in the same data type as that of _E to _E. (Example) Word (signed) type data
(2) MINIMUM, MINIMUM_E

Selects the minimum value to be output among the word (signed), double word (signed) or single-precision real type data input to $s1$ to $s2$, and outputs the operation result from $d$ in the same data type as that of $s1$ to $s2$.

(Example) Word (signed) type data

![Diagram of MINIMUM function]

(3) The values to be input to $s1$ to $s2$ are bit, word (signed), double word (signed), word(unsigned)/16-bit string, double word (unsigned)/32-bit string, single-precision real, double-precision real, string, or time type data.

(4) Rounding error may occur when specifying single-precision real or double-precision real type data to $s1$ through $s2$ by programming tool.
For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

(5) The number of pins of variable 's' can be changed in the range from 2 to 28.

(6) If word (unsigned) type/16-bit string type/double word (unsigned) type/32-bit string type is specified for $s2$, warning C9026 occurs.

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from $d$.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from $d$ is undefined. In this case, create a program so that the data output from $d$ is not used.
Operation Error

No operation error occurs in the execution of the MAXIMUM(_E) and MINIMUM(_E) function.

Program Example

(1) The program which outputs the maximum value of the word (signed) data input to variables \( \text{g} \text{-int}1 \) to \( \text{g} \text{-int}2 \) from \( \text{g} \text{-int}3 \) in the same data type as that of \( \text{g} \text{-int}3 \) to \( \text{g} \text{-int}4 \).

(a) Function without EN/ENO (MAXIMUM)

[Structured ladder/FBD]

```plaintext
ST
\text{g} \text{-int}3 := \text{MAXIMUM(} \text{g} \text{-int1, g} \text{-int2)};
```

(b) Function with EN/ENO (MAXIMUM_E)

[Structured ladder/FBD]

```plaintext
ST
\text{g} \text{-bool3 := MAXIMUM}_E( \text{g} \text{-bool1, g} \text{-int1, g} \text{-int2, g} \text{-int3);}
```

(2) The program which outputs the minimum value of the word (signed) data input to variables \( \text{g} \text{-int}1 \) to \( \text{g} \text{-int}4 \) from \( \text{g} \text{-int}3 \) in the same data type as that of \( \text{g} \text{-int}3 \) to \( \text{g} \text{-int4} \).

(a) Function without EN/ENO (MINIMUM)

[Structured ladder/FBD]

```plaintext
ST
\text{g} \text{-int3 := MINIMUM(} \text{g} \text{-int1, g} \text{-int2);}
```
5.5.3 Upper/Lower limit control

**LIMITATION(_E)**

**Operation processing**

1. Selects the value to be output among the bit, word (signed), double word (signed), word (unsigned)/16-bit string, double word (unsigned)/32-bit string, or single-precision real type, double-precision real, string, or time type data input to \(_s_1\), \(_s_2\), and \(_s_3\) according to their values, and outputs the operation result from \(_d\) in the same data type as that of \(_s_3\) to \(_s_3\).

   (a) When the input value of \(_s_2\) > the input value of \(_s_1\), outputs the input value \(_s_1\) from \(_d\).

   (b) When the input value of \(_s_2\) < the input value of \(_s_3\), outputs the input value \(_s_3\) from \(_d\).

   (c) When the input value of \(_s_1\) ≥ the input value of \(_s_2\) ≥ the input value of \(_s_3\), outputs the input value of \(_s_2\) from \(_d\).
(Example) Word (signed) type data

2. The values to be input to $s_1$, $s_2$, and $s_3$ are bit, word (signed), double word (signed), word (unsigned)/16-bit string, double word (unsigned)/32-bit string, single-precision real, double-precision real, string, or time type data. (the input value of $s_3$ < the input value of $s_2$)

3. Rounding error may occur when specifying single-precision real or double-precision real type data to $s_1$, $s_2$, or $s_3$ by programming tool.
For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

4. If word (unsigned) type/16-bit string type/double word (unsigned) type/32-bit string type is specified for $s_2$, warning C9026 occurs.

**Operation result**

1. Function without EN/ENO
   An operation is executed and the operation value is output from $s_1$.

2. Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>$s_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE$^1$</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

$^1$: When FALSE is output from ENO, the data output from $s_1$ is undefined.
In this case, create a program so that the data output from $s_1$ is not used.
### Operation Error

No operation error occurs in the execution of the LIMITATION(E) function.

### Program Example

The program which outputs the values input to variables $g_1$, $g_2$, and $g_3$ according to the word (signed) data from $s$ in the same data type as that of $g_1$, $g_2$, and $g_3$.

(a) Function without EN/ENO (LIMITATION)

[Structured ladder/FBD]

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

(b) Function with EN/ENO (LIMITATION_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := LIMITATION_E (g_bool1, g_int1, g_int2, g_int3, g_int4);
```
5.5.4 Multiplexer

**MUX(_E)**

_E: With EN/ENO

---

**Function**

**Operation processing**

1. Selects the value to be output among the values input to variables \(s_1\) to \(s_8\) according to the value input to \(n\), and outputs the operation result from \(d\) in the same data type as that of variables \(s_1\) to \(s_8\).

   When the input value of \(n\) is 1, the value input to \(s_1\) is output from \(d\).

   When the input value of \(n\) is \(n\), the value input to \(s_2\) is output from \(d\).

   (Example) Word (signed) type data

   \[
   \begin{array}{c|c|c|c}
   n & s_1 & s_2 & d \\
   \hline
   1 & 1234 & & \\
   & 5678 & & \\
   \end{array}
   \]

2. If a value input to \(n\) is outside the range of number of pins of variable 's', an undefined value is output from \(d\).

   (No operation error occurs. In case of MUX_E, FALSE is output from ENO.)
(3) The value to be input to \( n \) is word (signed) type data within the range from 1 to 28 (within the range of the number of pins of variable 's').

(4) The value to be input to variable 's' is bit, word (signed), double word (signed), word (unsigned)/16-bit string, double word (unsigned)/32-bit string, single-precision real, double-precision real, string, time, structure, or array type data.

(5) Rounding error may occur when specifying single-precision real or double-precision real type data to \( \oplus \) through \( \ominus \) by programming tool.
For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

(6) The number of pins of variable 's' can be changed in the range from 2 to 28.

**Operation result**

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \ominus \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \ominus )</th>
<th>Operation result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Op.)</td>
<td>TRUE</td>
<td>Operation output value</td>
<td></td>
</tr>
<tr>
<td>FALSE (Op.)</td>
<td>FALSE</td>
<td>Undefined value</td>
<td></td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \ominus \) is undefined.
In this case, create a program so that the data output from \( \ominus \) is not used.
Operation Error

No operation error occurs in the execution of the MUX(_E) function.

Program Example

The program which selects the value to be output among the values input to variables \( s_1 \) and \( s_2 \) according to the value input to \( n \), and outputs the operation result from \( d \) in the same data type as that of \( s_1 \) or \( s_2 \).

(a) Function without EN/ENO (MUX)

[Structured ladder/FBD]

(ST)
\[
g_{\text{int}4} := \text{MUX}(g_{\text{int}1}, g_{\text{int}2}, g_{\text{int}3});
\]

(b) Function with EN/ENO (MUX_E)

[Structured ladder/FBD]

(ST)
\[
g_{\text{bool}3} := \text{MUX}_E(g_{\text{bool}1}, g_{\text{int}1}, g_{\text{int}2}, g_{\text{int}3}, g_{\text{int}4});
\]
5.6 Standard Comparison Functions

5.6.1 Comparison

GT_E, GE_E, EQ_E, LE_E, LT_E, NE_E

GT_E, GE_E, EQ_E, LE_E, LT_E, NE_E

 Function

 Operation processing

(1) Performs comparison operation between the values input to variables \(s_1\) to \(s_2\), and outputs the operation result from \(d\) in bit type as that of variables \(s_1\) to \(s_2\).
(a) **GT_E(>)** Performs comparison of \([s_1 > s_2] & [s_2 > s_3] & \cdots & [s_{(n-1)} > s_n]\).

- Outputs **TRUE** if all of comparisons satisfy \(s_1 > s_2\).
- Outputs **FALSE** if any of comparisons satisfies \(s_1 < s_2\).

(b) **GE_E(\geq )** Performs comparison of \([s_1 \geq s_2] & [s_2 \geq s_3] & \cdots & [s_{(n-1)} \geq s_n]\).

- Outputs **TRUE** if all of comparisons satisfy \(s_1 \geq s_2\).
- Outputs **FALSE** if any of comparisons satisfies \(s_1 < s_2\).

(c) **EQ_E(=)** Performs comparison of \([s_1 = s_2] & [s_2 = s_3] & \cdots & [s_{(n-1)} = s_n]\).

- Outputs **TRUE** if all of comparisons satisfy \(s_1 = s_2\).
- Outputs **FALSE** if any of comparisons satisfies \(s_1 \neq s_2\).

(d) **LE_E(\leq )** Performs comparison of \([s_1 \leq s_2] & [s_2 \leq s_3] & \cdots & [s_{(n-1)} \leq s_n]\).

- Outputs **TRUE** if all comparisons satisfy \(s_1 \leq s_2\).
- Outputs **FALSE** if any of comparisons satisfies \(s_1 > s_2\).

(e) **LT_E(<)** Performs comparison of \([s_1 < s_2] & [s_2 < s_3] & \cdots & [s_{(n-1)} < s_n]\).

- Outputs **TRUE** if all comparisons satisfy \(s_1 < s_2\).
- Outputs **FALSE** if any of comparisons satisfies \(s_1 \geq s_2\).

(f) **NE_E(\neq)** Performs comparison of \([s_1 \neq s_2]\).

- Outputs **TRUE** if \(s_1 \neq s_2\).
- Outputs **FALSE** if \(s_1 = s_2\).

(2) The values to be input to \(s_1\) is bit, word (signed), double word (signed), word (unsigned), 16-bit string, double word (unsigned), 32-bit string, single-precision real, double-precision real, string, time type data.

(3) Rounding error may occur when specifying single-precision real or double-precision real type data to \(s_1\) through \(s_2\) by programming tool.
For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

(4) The number of pins of \(s_1\) can be changed in the range from 2 to 28. (The number of pins of \(s_1\) for comparison operator NE(_E)) is fixed at \(s_1\) and \(s_2\).
### Operation result

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \circ \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*¹</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*¹: When FALSE is output from ENO, the data output from \( \circ \) is undefined. In this case, create a program so that the data output from \( \circ \) is not used.
5.6 Standard Comparison Functions

5.6.1 Comparison Functions

Operation Error

No operation error occurs in the execution of the GT_E, GE_E, EQ_E, LE_E, LT_E, and NE_E functions.

Program Example

The program which performs comparison operation between the values input to $\#$ and $\&$, and outputs the operation result from $\%$.

[Structured ladder/FBD]

[ST]

```
g_bool3 := GT_E (g_bool1, g_int1, g_int2, g_bool2);
```
5.7 Standard Character String Functions

5.7.1 Extract mid string

**MID(_E)**

_E: With EN/ENO

---

Input argument,

<table>
<thead>
<tr>
<th>EN: Executing condition (TRUE: Execution, FALSE: Stop)</th>
<th>:Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>s(_IN): Input</td>
<td>:String (255)</td>
</tr>
<tr>
<td>n1(_L): Number of characters to be extracted</td>
<td>:Word (signed)</td>
</tr>
<tr>
<td>n2(_P): Start position to be extracted</td>
<td>:Word (signed)</td>
</tr>
</tbody>
</table>

Output argument,

<table>
<thead>
<tr>
<th>ENO: Execution result (TRUE: Normal, FALSE: Error)</th>
<th>:Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>d: Output</td>
<td>:String (255)</td>
</tr>
</tbody>
</table>

---

**Operation processing**

1. Extracts the specified number of characters from the specified start position in the character string input to _E, and outputs the operation result from _E.
2. The number of characters to be extracted is specified by the value input to n1.
3. The start position of the characters to be extracted is specified by the value input to n2.
5.7 Standard Character String Functions

5.7.1 Extract mid string

(Example) Values input to n1 and n2 are 5

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>2nd word</td>
</tr>
<tr>
<td>3rd word</td>
<td>4th word</td>
</tr>
<tr>
<td>5th word</td>
<td>6th word</td>
</tr>
</tbody>
</table>

(2) The value to be input to \( \odot \) is string type data within the range from 0 to 255 bytes.
(3) The value to be input to n1 is word (signed) type data within the range from 0 to 255.
   (The input value must not exceed the number of characters of character string input to \( \odot \).)
(4) The value to be input to n2 is word (signed) type data within the range from 1 to 255.
   (The input value must not exceed the number of characters of character string input to \( \odot \).)

**Operation result**

(1) Function without EN/ENO
   An operation is executed and the operation value is output from \( \odot \).
(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \odot )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \odot \) is undefined.
   In this case, create a program so that the data output from \( \odot \) is not used.
5.7 Standard Character String Functions

5.7.1 Extract mid string

**Operation Error**

These functions consist of the following instructions.
MID(_E): MIDR

For details of an error which occurs when the function is executed, refer to MELSEC-Q/L Structured Programming Manual (Common Instructions).

**Program Example**

The program which extracts the specified number of characters from the specified start position in the character string input to @, and outputs the operation result from @.

(a) Function without EN/ENO (MID)

[Structured ladder/FBD]

1

<table>
<thead>
<tr>
<th>g_string1</th>
<th>_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_int1</td>
<td>_L</td>
</tr>
<tr>
<td>g_int2</td>
<td>_P</td>
</tr>
</tbody>
</table>

[ST]

g_string2 := MID(g_string1, g_int1, g_int2);

(b) Function with EN/ENO (MID_E)

[Structured ladder/FBD]

2

<table>
<thead>
<tr>
<th>g_bool1</th>
<th>MID_E</th>
<th>g_bool3</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_string1</td>
<td>_N</td>
<td>g_bool3</td>
</tr>
<tr>
<td>g_int1</td>
<td>_L</td>
<td></td>
</tr>
<tr>
<td>g_int2</td>
<td>_P</td>
<td></td>
</tr>
</tbody>
</table>

[ST]

g_bool3 := MID_E(g_bool1, g_string1, g_int1, g_int2, g_string2);
5.7.2 String concatenation

CONCAT(_E)

Function

Operation processing

(1) Concatenates the character string input to \( \text{①} \) to \( \text{③} \) following the one input to \( \text{③} \) , and outputs the operation result from \( \text{③} \) .

This function concatenates character string \( \text{①} \) to \( \text{③} \) with ignoring '00H', which indicates the end of character string \( \text{③} \) . If the concatenated character string has over 255 bytes, the character string up to 255 bytes is output.

\[
\begin{array}{c|c|c}
\text{①} & \text{②} & \text{③} \\
\hline
\text{High-order byte} & \text{Low-order byte} & \\
1\text{st word} & 42(H) & 41(A) \\
2\text{nd word} & 44(H) & 43(C) \\
3\text{rd word} & 00n & 45(E) \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{High-order byte} & \text{Low-order byte} & \\
1\text{st word} & 32(2) & 31(1) \\
2\text{nd word} & 34(4) & 33(3) \\
3\text{rd word} & 36(6) & 35(5) \\
4\text{th word} & 00n & \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{High-order byte} & \text{Low-order byte} & \\
1\text{st word} & 42(H) & 41(A) \\
2\text{nd word} & 44(H) & 43(C) \\
3\text{rd word} & 31(1) & 45(E) \\
4\text{th word} & 33(3) & 32(2) \\
5\text{th word} & 35(5) & 34(4) \\
6\text{th word} & 00n & 36(6) \\
\end{array}
\]

(2) The values to be input to \( \text{①} \) and \( \text{③} \) to \( \text{①} \) are string type data within the range from 0 to 255 bytes.

(3) The number of pins for \( \text{①} \) can be changed in the range from 2 to 28.
**Operation result**

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \text{②} \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \text{②} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \text{②} \) is undefined. In this case, create a program so that the data output from \( \text{②} \) is not used.
### Operation Error

No operation error occurs in the execution of the CONCAT(E) function.

### Program Example

The program which concatenates the character string input to `@` following the one input to `®`, and outputs the operation result from `®`.

(a) Function without EN/ENO (CONCAT)

[Structured ladder/FBD]

```
1
```

```
CONCAT
IN c_string1
IN c_string2
OUT c_string3
```

[ST]

```
g_string3 := CONCAT(g_string1, g_string2);
```

(b) Function with EN/ENO (CONCAT_E)

[Structured ladder/FBD]

```
2
```

```
g_bool1
```

```
CONCAT_E
EN g_bool1
IN g_string1
IN g_string2
ENO g_bool3
OUT g_string3
```

[ST]

```
g_bool3 := CONCAT_E(g_bool1, g_string1, g_string2, g_string3);
```
5.7.3 String insertion

**INSERT(_E)**

**Operation processing**

1. Inserts the character string input to `s1` to the specified position in the character string input to `s2`, and outputs the operation result from `d`.
   - Specify the start position of the character string to be inserted by the value input to `n`.
   - After the insertion of character string `s1` to character string `s2`, '00H' that indicates the end of character string `s1` is ignored. If the character string after insertion has over 255 bytes, the character string up to 255 bytes is output.
(Example) Value input to n is 4

Input value to \( s_1 \) 'ABCDE'

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>42h(B)</td>
</tr>
<tr>
<td>2nd word</td>
<td>44h(D)</td>
</tr>
<tr>
<td>3rd word</td>
<td>00h</td>
</tr>
</tbody>
</table>

Input value to \( s_2 \) '123456'

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>32h(2)</td>
</tr>
<tr>
<td>2nd word</td>
<td>34h(4)</td>
</tr>
<tr>
<td>3rd word</td>
<td>36h(6)</td>
</tr>
<tr>
<td>4th word</td>
<td>00h</td>
</tr>
</tbody>
</table>

Start position to be inserted n: 4th character

Output value 'ABC123456DE'

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>42h(B)</td>
</tr>
<tr>
<td>2nd word</td>
<td>31h(1)</td>
</tr>
<tr>
<td>3rd word</td>
<td>33h(3)</td>
</tr>
<tr>
<td>4th word</td>
<td>44h(D)</td>
</tr>
<tr>
<td>5th word</td>
<td>00h</td>
</tr>
</tbody>
</table>

(2) The values to be input to \( s_1 \) and \( s_2 \) are string type data within the range from 0 to 255 bytes.

(3) The value to be input to n is word (signed) type data within the range from 1 to 255.

(The input value must not exceed the number of characters of character string input to \( s_1 \).)

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( s_2 \).

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( s_2 \) is undefined. In this case, create a program so that the data output from \( s_2 \) is not used.
**Operation Error**

No operation error occurs in the execution of the INSERT(_E) function.

**Program Example**

The program which inserts the character string input to \( s_2 \) to the specified position in the character string input to \( s_1 \), and outputs the operation result from \( s_3 \).

(a) Function without EN/ENO (INSERT)

[Structured ladder/FBD]

\[
g_{\text{string}3} := \text{INSERT}(g_{\text{string}1}, g_{\text{string}2}, g_{\text{int}1});
\]

(b) Function with EN/ENO (INSERT_E)

[Structured ladder/FBD]

\[
g_{\text{bool}} := \text{INSERT_E}(g_{\text{bool}1}, g_{\text{string}1}, g_{\text{string}2}, g_{\text{int}1}, g_{\text{string}3});
\]
5.7.4 String deletion

DELETE(_E)

Function

**Operation processing**

1. Deletes the specified number of characters from the specified position in the character string input to \( s \), and outputs the remaining character string from \( d \).
   - The number of characters to be deleted is specified by the value input to \( n1 \).
   - The start position to be deleted in the character string is specified by the value input to \( n2 \).
5.7 Standard Character String Functions

5.7.4 String deletion

DELETE(_E)

(Example) Values input to n1 and n2 are 5

\[
\text{"ABCDEF12345"} \rightarrow \text{"ABCD45"}
\]

1st word 42H(B) : 41H(A)
2nd word 44H(D) : 43H(C)
3rd word 46H(F) : 45H(E)
4th word 32H(2) : 31H(1)
5th word 34H(4) : 33H(3)
6th word 00H : 35H(5)

Start position to be deleted n2: 5th character

Number of characters to be deleted n1: 5 characters

(2) The value to be input to ③ is string type data within the range from 0 to 255 bytes.

(3) The value to be input to n1 is word (signed) type data within the range from 0 to 255.

(The input value must not exceed the number of characters of character string input to ③.)

(4) The value to be input to n2 is word (signed) type data within the range from 1 to 255.

(The input value must not exceed the number of characters of character string input to ③.)

Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from ③.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined.

In this case, create a program so that the data output from ③ is not used.
**Operation Error**

No operation error occurs in the execution of the DELETE(_E) function.

**Program Example**

The program which deletes the specified number of characters from the specified position in the character string input to , and outputs the remaining character string from .

(a) Function without EN/ENO (DELETE)

[Structured ladder/FBD]

```
[ST]
g_string2 := DELETE(g_string1, g_int1, g_int2);
```

(b) Function with EN/ENO (DELETE_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := DELETE_E(g_bool1, g_string1, g_int1, g_int2, g_string2);
```
5.7.5 String replacement

REPLACE(_E)

Function

Operation processing

(1) Replaces the specified number of characters from the specified position in the character string input to  with the character string input to , and outputs the operation result from .

The number of characters to be replaced is specified by the value input to n1.
The start position to be replaced in the character string is specified by the value input to n2.
(Example) Values input to n1 and n2 are 5

<table>
<thead>
<tr>
<th>Input value to n1</th>
<th>Output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ABCDEFGH123'</td>
<td>'ABCD1234523'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>42h(B)</td>
<td>1st word</td>
<td>42h(B)</td>
</tr>
<tr>
<td>2nd word</td>
<td>44h(D)</td>
<td>2nd word</td>
<td>44h(D)</td>
</tr>
<tr>
<td>3rd word</td>
<td>46h(F)</td>
<td>3rd word</td>
<td>32h(2)</td>
</tr>
<tr>
<td>4th word</td>
<td>48h(H)</td>
<td>4th word</td>
<td>34h(4)</td>
</tr>
<tr>
<td>5th word</td>
<td>32h(2)</td>
<td>5th word</td>
<td>32h(2)</td>
</tr>
<tr>
<td>6th word</td>
<td>00h</td>
<td>6th word</td>
<td>00h</td>
</tr>
</tbody>
</table>

Start position to be replaced n2: 5th character

Input value to n2: '123456'

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>Number of characters to be replaced n1: 5 characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>32h(2)</td>
<td></td>
</tr>
<tr>
<td>2nd word</td>
<td>34h(4)</td>
<td></td>
</tr>
<tr>
<td>3rd word</td>
<td>36h(6)</td>
<td></td>
</tr>
<tr>
<td>4th word</td>
<td>00h</td>
<td></td>
</tr>
</tbody>
</table>

(2) The values to be input to n1 and n2 are string type data within the range from 0 to 255 bytes.

(3) The value to be input to n1 is word (signed) type data within the range from 0 to 255.
   (The input value must not exceed the number of characters of character string input to input variable n1.)

(4) The value to be input to n2 is word (signed) type data within the range from 1 to 255.
   (The input value must not exceed the number of characters of character string input to n2.)

Operation result

(1) Function without EN/ENO
   An operation is executed and the operation value is output from ENO.

(2) Function with EN/ENO
   The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from ③ is undefined. In this case, create a program so that the data output from ③ is not used.
5.7 Standard Character String Functions
5.7.5 String replacement

---

**Operation Error**

No operation error occurs in the execution of the REPLACE(_E) function.

---

**Program Example**

The program which replaces the specified number of characters from the specified position in the character string input to \( \mathbb{g} \) with the character string input to \( \mathbb{s} \), and outputs the operation result from \( \mathbb{d} \).

(a) Function without EN/ENO (REPLACE)

[Structured ladder/FBD]

```plaintext
1

\[ g_{\text{string3}} := \text{REPLACE}(g_{\text{string1}}, g_{\text{string2}}, g_{\text{int1}}, g_{\text{int2}}); \]
```

(b) Function with EN/ENO (REPLACE_E)

[Structured ladder/FBD]

```plaintext
2

\[ g_{\text{bool3}} := \text{REPLACE}_{E}(g_{\text{bool1}}, g_{\text{string1}}, g_{\text{string2}}, g_{\text{int1}}, g_{\text{int2}}, g_{\text{string3}}); \]
```
5.8 Functions of Time Data Type

5.8.1 Addition

ADD_TIME(E)

_E: With EN/ENO

Function

Operation processing

1. Performs addition (+) on time type data input to _E and _E, and outputs the operation result from _E in time type.

   (Example) When the input value to _E and _E are T#1d2h33m44s55ms (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and T#2ms (2 milliseconds).

2. The value to be input to _E, _E are time type data.
5.8 Functions of Time Data Type

5.8.1 Addition

(3) No operation error occurs even if an underflow/overflow occurs. Data is output from \( \circ \) as follows.

In case of ADD_TIME_E, TRUE is output from ENO.

(Example) Overflow

\[
\text{T#24d20h31m23s647m} + \text{T#2ms} \rightarrow \text{T#-24d20h31m23s647m}
\]

Since the highest-order bit is 1, the result value is negative.

(Example) Underflow

\[
\text{T#-24d20h31m23s648ms} + \text{T#-2ms} \rightarrow \text{T#24d20h31m23s646ms}
\]

Since the highest-order bit is 0, the result value is positive.

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( \circ \).

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \circ \) is undefined. In this case, create a program so that the data output from \( \circ \) is not used.

**Operation Error**

No operation error occurs in the execution of the ADD_TIME_E function.
Program Example

The program which performs addition \((\oplus + \ominus)\) on time type data input to \(\oplus\) and \(\ominus\), and outputs the operation result from \(\ominus\) in time type.

(a) Function without EN/ENO (ADD_TIME)

[Structured ladder/FBD]

```
  1
  
  g_time1   ADD_TIME
  g_time2   JN1
  g_time3   JN2
```

[ST]

\[
g_{time3} := \text{ADD\_TIME}(g_{time1}, g_{time2});
\]

(b) Function with EN/ENO (ADD\_TIME\_E)

[Structured ladder/FBD]

```
  2
  
  g_bool1   ADD_TIME_E
  g_time1   EN
  g_time2   IN1
  g_time3   IN2
  g_bool3   EXO
```

[ST]

\[
g_{bool3} := \text{ADD\_TIME\_E}(g_{bool1}, g_{time1}, g_{time2}, g_{time3});
\]
5.8.2 Subtraction

SUB_TIME(_E)

_E: With EN/ENO

---

**Operation processing**

1. Performs subtraction ($a - b$) on time type data input to $a$ and $b$, and outputs the operation result from $a$ in time type.

   (Example) When the input value to $a$ and $b$ are T#1d2h33m44s55ms (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and T#2ms (2 milliseconds).

   ![Diagram](image)

   Time type

2. The value to be input to $a$, $b$ are time type data.
(3) No operation error occurs even if an underflow/overflow occurs. Data is output from $\odot$ as follows.

In case of SUB_TIME_E, TRUE is output from ENO.

(Example) Overflow
\[
\begin{align*}
\text{T#24d20h31m23s647ms} & \rightarrow \text{T#-2ms} & \rightarrow \text{T#-24d20h31m23s647ms} \\
(7FFFFFFFH) & & (FFFFFFFEH) & & (8000001H)
\end{align*}
\]

Since the highest-order bit is 1, the result value is negative.

(Example) Underflow
\[
\begin{align*}
\text{T#-24d20h31m23s648ms} & \rightarrow \text{T#2ms} & \rightarrow \text{T#24d20h31m23s646ms} \\
(80000000H) & & (00000002H) & & (7FFFFFFEH)
\end{align*}
\]

Since the highest-order bit is 0, the result value is positive.

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from $\odot$.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN (Function execution)</th>
<th>ENO</th>
<th>$\odot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE$^1$</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from $\odot$ is undefined. In this case, create a program so that the data output from $\odot$ is not used.

**Operation Error**

No operation error occurs in the execution of the SUB_TIME(E) function.
Program Example

The program which performs subtraction ($\text{time}_1 - \text{time}_2$) on time type data input to $\text{time}_1$ and $\text{time}_2$, and outputs the operation result from $\text{time}_3$ in time type.

(a) Function without EN/ENO (SUB_TIME)

[Structured ladder/FBD]

[ST]

g_time3 := SUB_TIME(g_time1, g_time2);

(b) Function with EN/ENO (SUB_TIME_E)

[Structured ladder/FBD]

[ST]

g_bool3 := SUB_TIME_E(g_bool1, g_time1, g_time2, g_time3);
5.8.3 Multiplication

MUL_TIME(_E)

_E: With EN/ENO

Function

Operation processing

(1) Performs multiplication \((\mathbb{1} \times \mathbb{2})\) on time type data input to \(\mathbb{1}\) and the word (signed), double word (signed), single-precision real or double-precision real type data input to \(\mathbb{2}\), and outputs the operation result from \(\mathbb{3}\) in time type.

(Example) When the input value to \(\mathbb{1}\) and \(\mathbb{2}\) are \(T\#1d2h33m44s55ms\) (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and 2.

\[
\begin{align*}
\mathbb{1} & : T\#1d2h33m44s55ms \\
\mathbb{2} & : 2 \\
\mathbb{3} & : T\#2d5h7m28s110ms
\end{align*}
\]

Time type
Word (signed)
Time type

(2) The value to be input to \(\mathbb{1}\) is time type data.

(3) The value to be input to \(\mathbb{2}\) is word (signed), double word (signed), single-precision real or double-precision real type data.

(4) Rounding error may occur when specifying single-precision real or double-precision real type data to \(\mathbb{2}\) by programming tool.

For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).
(5) No operation error occurs even if an underflow/overflow occurs. Data is output from  as follows.

In case of MUL_TIME_E, TRUE is output from ENO.
(Although the operation result is 64-bit data, data is output in time type with the high-order 32 bits discarded.)
(Example) Overflow

\[
\begin{array}{c}
\text{T#24d20h31m23s647ms} \\
(7FFFFFFFH) \\
\times \\
(00000002H) \\
\rightarrow \\
\text{T#-2ms} \\
(FFFFF8H)
\end{array}
\]

Since the highest-order bit is 1, the result value is negative.

(Example) Underflow

\[
\begin{array}{c}
\text{T#-24d20h31m23s648ms} \\
(80000000H) \\
\times \\
(00000002H) \\
\rightarrow \\
\text{T#0ms} \\
(00000000H)
\end{array}
\]

Since the highest-order bit is 0, the result value is positive.

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from  .

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>⑤</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE*1</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from  is undefined. In this case, create a program so that the data output from  is not used.
5.8 Functions of Time Data Type

5.8.3 Multiplication

Operation Error

No operation error occurs in the execution of the MUL_TIME(E) function.

Program Example

The program which performs multiplication (\( \times \)) on time type data input to \( \) and the word (signed) type data input to \( \), and outputs the operation result from \( \) in time type.

(a) Function without EN/ENO (MUL_TIME)

[Structured ladder/FBD]

(ST)

\[ g\text{-time2} := \text{MUL\_TIME}(g\text{-time1}, g\text{-int1}); \]

(b) Function with EN/ENO (MUL_TIME_E)

[Structured ladder/FBD]

(ST)

\[ g\text{-bool3} := \text{MUL\_TIME\_E}(g\text{-bool1}, g\text{-time1}, g\text{-int1}, g\text{-time2}); \]
5.8.4 Division

DIV_TIME(_E)

DIV_TIME(_E)

_E: With EN/ENO

 Structured ladder/FBD

ST

EN: Executing condition (TRUE: Execution, FALSE: Stop)
ENO: Execution result (TRUE: Normal, FALSE: Error)
d: Output

Input argument,

s1(_IN1): Input
s2(_IN2): Input

Output argument,

d: Output

Operation processing

(1) Performs division (÷) on time type data input to  and the word (signed), double word (signed), single-precision real or double-precision real type data input to  , and outputs the quotient of the operation result from  in time type. Remainder is rounded down.

(Example) When the input value to  and  are T#1d2h33m44s55ms (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and 2.

\[
\begin{array}{c}
\text{T#1d2h33m44s55ms} \\
\text{Time type}
\end{array}
\div
\begin{array}{c}
2 \\
\text{Word (signed)}
\end{array}
\rightarrow
\begin{array}{c}
\text{T#13h16m63s27ms} \\
\text{T#1ms} \\
\text{Discarded}
\end{array}
\]

(2) The value to be input to  is time type data.

(3) The value to be input to  is word (signed), double word (signed), single-precision real or double-precision real type data.

(The value to be input to  must be other than 0.)
(4) Rounding error may occur when specifying single-precision real or double-precision real type data to \( \oplus \) by programming tool. For the precautions on setting the input value by the programming tool, refer to MELSEC-Q/L/F Structured Programming Manual (Fundamentals).

**Operation result**

(1) Function without EN/ENO
An operation is executed and the operation value is output from \( \oplus \).

(2) Function with EN/ENO
The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \oplus )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE(^{-1})</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

*1: When FALSE is output from ENO, the data output from \( \oplus \) is undefined. In this case, create a program so that the data output from \( \oplus \) is not used.

**Operation Error**

An operation error occurs in the following cases.

- The value input to \( \oplus \) is 0. (Division by zero) (Error code: 4100)

**Program Example**

The program which performs division (\( \oplus \)) on time type data input to \( \oplus \) and the word (signed) type data input to \( \oplus \), and outputs the quotient of the operation result from \( \oplus \) in time type.

(a) Function without EN/ENO (DIV_TIME)

[Structured ladder/FBD]

```
1
```

[ST]

```
g_time2 := DIV_TIME(g_time1, g_int1);
```

(b) Function with EN/ENO (DIV_TIME_E)

[Structured ladder/FBD]

```
2
```

[ST]

```
g_bool3 := DIV_TIME_E(g_bool1, g_time1, g_int1, g_time2);
```
5.9 Standard Bistable Function Blocks

5.9.1 Standard bistable function blocks (Set-dominant)

Function

Operation processing

Sets \( @ \) when \( \infty \) is turned ON, and resets \( @ \) when \( @ \infty \) is turned ON while \( \infty \) is OFF.

\( @ \) is not reset even when \( @ \infty \) is turned ON while \( \infty \) is ON.
Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from ③.

[Timing chart]

Sets ③ when ① = ON. Resets ③ when ① = OFF and ② = ON.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]

Sets ③ when EN = ON and ① = ON. Resets ③ when EN = ON, ① = OFF and ② = ON.
### Operation Error

No operation error occurs in the execution of the SR (_E) function.

### Program Example

The program which outputs bit type data input to $\bullet$ from $\bigcirc$ and holds the output, and resets the value of $\bigcirc$ only when bit type data input to $\bullet$ is 1 and the data input to $\bigcirc$ is 0.

(a) Function without EN/ENO (SR)

[Structured ladder/FBD]

```
   SR_Instance(_,S1:= g_bool1 ,RESET:= g_bool2 ,Q1:= g_bool3 );
```

(b) Function with EN/ENO (SR_E)

[Structured ladder/FBD]

```
   SR_E_Instance(EN:= X0 ,_S1:= g_bool1 ,_R:= g_bool2 ,Q1:= g_bool3 ,ENO:= Y20 );
```
5.9.2 Standard bistable function blocks (Reset-dominant)

**Function**

**Operation processing**

Sets  when  is turned ON, and resets  when  is turned ON.

 is not set even when  is turned ON while  is ON.
**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from ③.

[Timing chart]

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]
5.9 Standard Bistable Function Blocks

5.9.2 Standard bistable function blocks (Reset-dominant)

---

**Operation Error**

No operation error occurs in the execution of the RS (E) function.

---

**Program Example**

The program which outputs bit type data input to \( \textcircled{1} \) from \( \textcircled{2} \) and holds the output, and resets forcibly the value of \( \textcircled{3} \) when bit type data input to \( \textcircled{4} \) is 1.

(a) Function without EN/ENO (RS)

[Structured ladder/FBD]

```plaintext
RS_inst(_S:= g_bool1 ,_R1:= g_bool2 ,Q1:= g_bool3 );
```

(b) Function with EN/ENO (RS_E)

[Structured ladder/FBD]

```plaintext
RS_E_inst(EN:= X0 ,_S:= g_bool1 ,_R1:= g_bool2 ,Q1:= g_bool3 ,ENO:= Y20 );
```
5.10 Standard Edge Detection Function Blocks

5.10.1 Rising edge detector

\[ \text{R}_{-}\text{TRIG}(\_E) \]

_E: With EN/ENO

Input argument, \( \text{EN} \): Executing condition (TRUE: Execution, FALSE: Stop) :Bit
\( s(_\text{CLK}) \): Input
Output argument, \( \text{ENO} \): Execution result (TRUE: Normal, FALSE: Error or stop) :Bit
\( d(Q) \): Output

\[ \text{R}_{-}\text{TRIG}(\_E, \text{EN}, s(_\text{CLK}), \text{ENO}, d(Q)) \]

Operation processing

Turns ON ③ for one scan when ③ is turned ON.
Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from ③.

[Timing chart]

Turns ④ ON at the rising of ③.

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]

Turns ③ ON when EN = ON and the rising of ③.

When EN = OFF, ③ retains the previous scan output result.

Operation Error

No operation error occurs in the execution of the R_TRIG(_E) function.
Program Example

The program which turns ON \( \circ \) for one scan when bit type data input to \( \circ \) is turned from OFF to ON.

(a) Function without EN/ENO (R_TRIG)

[Structured ladder/FBD]

```
R_TRIG_Inst(_CLK:= g_bool1 ,Q:= g_bool2);
```

(b) Function with EN/ENO (R_TRIG_E)

[Structured ladder/FBD]

```
R_TRIG_E_Inst(EN:= X0 ,_CLK:= g_bool1 ,Q:= g_bool2 ,ENO:= Y20);
```
5.10.2 Falling edge detector

**F_TRIG(_E)**

_E: With EN/ENO

---

**Structured ladder/FBD**

- **EN**: Executing condition (TRUE: Execution, FALSE: Stop)
- **s(CLK)**: Input
- **ENO**: Execution result (TRUE: Normal, FALSE: Error or stop)
- **d(Q)**: Output

---

**Operation processing**

Turns ON for one scan when is turned OFF.

**Operation result**

1. Function without EN/ENO
   
   An operation is executed and the operation value is output from .
   
   [Timing chart]

   ![Timing chart image]

   Turns (ON) at the falling of .
   
   Turns (OFF) at the next scan.
(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>Operation output value</th>
<th>Previous output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
<td>Previous output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
<td></td>
</tr>
</tbody>
</table>

[Timing chart]

Turns ③ ON when EN = ON and the falling of ④.

When EN = OFF, ② retains the previous scan output result.

Operation Error

No operation error occurs in the execution of the F_TRIG (_E) function.

Program Example

The program which turns ON ② for one scan when bit type data input to ③ is turned from ON to OFF.

(a) Function without EN/ENO (F_TRIG)

[Structured ladder/FBD]

```
F_TRIG_Inst(_CLK:= g_bool1 ,Q:= g_bool2 )
```

[ST]

```
F_TRIG_Inst(_CLK:= g_bool1 ,Q:= g_bool2 );
```

(b) Function with EN/ENO (F_TRIG_E)

[Structured ladder/FBD]

```
F_TRIG_E_Inst(EN:= X0 ,_CLK:= g_bool1 ,Q:= g_bool2 ,ENO:= Y20 )
```

[ST]

```
F_TRIG_E_Inst(EN:= X0 ,_CLK:= g_bool1 ,Q:= g_bool2 ,ENO:= Y20 );
```
5.11 Standard Counter Function Blocks

5.11.1 Up counter

CTU(_E) _E: With EN/ENO

CTU(_E) indicates any of the following functions.

CTU CTU_E

Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop) :Bit
s1(CU): Count signal input :Bit
s2(RESET): Count reset :Bit
n(PV): Maximum count value :Word (signed)

Output argument, ENO: Execution result (TRUE: Normal, FALSE: Error or stop) :Bit
d1(Q): Count match output :Bit
d2(CV): Count value :Word (signed)

Operation processing

(1) Counts  when  is turned ON.
When the count value  reaches the value input to n,  turns ON.
When  is turned ON,  turns OFF and count value  is reset.

(2) Valid setting range for n is -32768 to 32767.
However, if 0 or less is set,  is turned on regardless of the count value of .
Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( D_1 \) and \( D_2 \).

[Timing chart]

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( D_1 ), ( D_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]

Operation Error

No operation error occurs in the execution of the CTU (E) function.
Program Example

The program which counts the number of times that bit type data input to \( \oplus \) is turned from OFF to ON, and outputs the count value from \( \ominus \).

(a) Function without EN/ENO (CTU)

[Structured ladder/FBD]

```
CTU_Inst(CU:= g_bool1 ,RESET:= g_bool2 ,PV:= g_int1 ,Q:= g_bool3 ,CV:= g_int2 );
```

(b) Function with EN/ENO (CTU_E)

[Structured ladder/FBD]

```
CTU_E_Inst(EN:= M10 ,CU:= g_bool1 ,RESET:= g_bool2 ,PV:= g_int1 ,Q:= g_bool3 ,CV:= g_int2 ,ENO:= M11 );
```
5.11.2 Down counter

CTD(_E)

CTD(_E)  _E: With EN/ENO

Function

Operation processing

(1) Counts down (-1) \( \oplus \) when \( \oplus \) is turned ON.
   \( n \) sets the initial value for subtraction.
   \( \ominus \) turns ON when count value \( \ominus \) reaches 0.
   When \( \ominus \) is turned ON, \( \ominus \) turns OFF and initial value for subtraction \( n \) is set for count value \( \ominus \).

(2) Valid setting range for \( n \) is -32768 to 32767.
   However, if 0 or less is set, \( \ominus \) is turned on regardless of the count value of \( \ominus \).
Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( \text{CTD}(\text{E}) \) and \( \text{ENO} \).

[Timing chart]

When \( n=3 \)

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( \text{ENO} ), ( \text{ENO} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]

When \( n=3 \)

\( \text{EN} \) and \( \text{ENO} \) initialize \( \text{ENO} \) when \( \text{EN} = \text{ON} \) and \( \text{ENO} = \text{ON} \). Counts \( \text{ENO} \) down when \( \text{EN} = \text{ON} \) and the falling of \( \text{ENO} \).

Operation Error

No operation error occurs in the execution of the CTD (E) function.
Program Example

The program which counts the number of times that bit type data input to the bit is turned from OFF to ON, and turns ON when the value of reaches 0.

(a) Function without EN/ENO (CTD)

```plaintext
[Structured ladder/FBD]

[ST]
CTD_Inst(CD: g_bool1, LOAD: g_bool2, PV: g_int1, Q: g_bool3, CV: g_int2);
```

(b) Function with EN/ENO (CTD_E)

```plaintext
[Structured ladder/FBD]

[ST]
CTD_E_Inst(EN: M10, CD: g_bool1, LOAD: g_bool2, PV: g_int1, Q: g_bool3, CV: g_int2, ENO: M11);
```
5.11.3 Up/Down counter

CTUD(_E)

_E: With EN/ENO

CTUD(_E)

Function

Operation processing

1. Counts up (+1) when ON is turned ON, and counts down (-1) when ON is turned ON. n sets the maximum value of counter.
   - turns ON when reaches 0.
   - turns ON when reaches the maximum value n.
   - Resets when turns ON.
   - The value of n is set to when is turned ON.

2. Valid setting range for n is -32768 to 32767.
   However, if 0 or less is set, and are turned on regardless of the count value of .

Structured ladder/FBD

CTU_D(_E) (EN, s1, s2, s3, s4, n, ENO, d1, d2, d3);

Input argument,
EN: Executing condition (TRUE: Execution, FALSE: Stop) :Bit
s1(CU): Count-up signal input :Bit
s2(CD): Count-down signal input :Bit
s3(RESET): Count-up reset :Bit
s4(LOAD): Count-down reset :Bit
n(PV): Maximum count value :Word (signed)

Output argument,
ENO: Execution result (TRUE: Normal, FALSE: Error or stop) :Bit
d1(QU): Count-up match output :Bit
d2(QD): Count-down match output :Bit
d3(CV): Current count value :Word (signed)
### Operation result

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( \text{\#1}, \text{\#2}, \) and \( \text{\#3}. \)

[Timing chart]

When \( n=3 \)

Counts \( \text{\#3} \) up when \( \text{\#1} \) is turned from OFF to ON.

Initializes \( \text{\#2} \) when \( \text{\#2} \) is turned from OFF to ON.

Counts \( \text{\#3} \) down when \( \text{\#2} \) is turned from OFF to ON.

Initializes \( \text{\#3} \) when \( \text{\#4} \) is turned from OFF to ON.

### CTUD(E)

5.11 Standard Counter Function Blocks
5.11.3 Up/Down counter
(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>(\oplus, \oplus, \oplus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]

When n=3

Counts \(d_3\) up when EN = ON and \(s_4\) is turned from OFF to ON.
Clears \(d_0\) when EN = ON and \(s_3\) is turned from OFF to ON.
Counts \(d_3\) down when EN = ON and \(s_2\) is turned from OFF to ON.
Initializes \(d_3\) when EN = ON and \(s_4\) is turned from OFF to ON.
### Operation Error

No operation error occurs in the execution of the CTUD (_E) function.

### Program Example

The program which counts the number of times that bit type data input to  is turned from OFF to ON, and turns ON when the value of reaches the value set at . Simultaneously, it counts the number of times that bit type data input to is turned from OFF to ON, and turns ON when the value of reaches 0.

(a) Function without EN/ENO (CTD)

```
[Structured ladder/FBD]

[ST]
CTUD_Init(CU:= g_bool1 , CD := g_bool2 , RESET:= g_bool3 , LOAD:= g_bool4 , PV:= g_int1 , QU:= g_bool5 , QD:= g_bool6 , CV:= g_int2 );
```

(b) Function with EN/ENO (CTD_E)

```
[Structured ladder/FBD]

[ST]
CTUD_Init(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 );
```
5.11.4 Counter function blocks

**COUNTER_FB_M**

**Operation processing**

1. Counts the detected rising edge (from OFF to ON) of $\delta$. It is not counted when $\delta$ stays ON. The count starts from the value input to $\gamma$ and when the count value reaches the value input to $\epsilon$, $\epsilon$ turns ON. The current value is stored in $\delta$.

2. Valid setting range for $\delta$ is 0 to 32767.

3. Valid setting range for $\delta$ is -32768 to 32767. However, if negative value is specified, the initial value is 0.
(4) When resetting the current value of the counter, reset \( \Box \).

(Example) When instance name is COUNTER_FB_M_1.

[Structured ladder/FBD]

\[
\text{COUNTER_FB_M_1(Coil:=Var_M0, Preset:=10, ValueIn:=0, ValueOut:=Var_D10, Status:=Var_M10);}
\]

RST(M15, COUNTER_FB_M_1.Coil);

Operation Error

No operation error occurs in the execution of the counter function blocks.

Program Example

The program which counts the number of times that bit type data input to \( \Box \) is turned from OFF to ON, and outputs the count value from \( \Box \).

[Structured ladder/FBD]

\[
\text{COUNTER_FB_M_Inst(Coil:= Var_M0 , Preset:= 10 , ValueIn:= 0 , ValueOut:= Var_D10 , Status:= Var_M10 );}
\]

[Timing chart]

Var_M0 OFF

Value of Var_D10

Var_M10 OFF
5.12 Standard Timer Function Blocks

5.12.1 Pulse timer

TP(_E), TP_HIGH(_E)

--- Function ---

**Operation processing**

Turns ON  for the duration set to n after  is turned ON. The duration (elapsed time) during which  stays ON is set to  .

When the elapsed time reaches the preset time,  turns OFF.

The elapsed time is not reset even when  turns OFF.

After  turns OFF, it is reset when  is OFF.
(1) **TP(_E)**

Uses a low-speed timer to count the elapsed time.
Output time can be set between 1ms and 1000ms. The unit is set in Timer limit setting on the PLC system of PLC parameter.
Valid setting range for n is T#0ms to T#3276700ms.

(2) **TP_HIGH(_E)**

Uses a high-speed timer to count the elapsed time.
Output time can be set within the following range. The unit is set in Timer limit setting on the PLC system of PLC parameter.

<table>
<thead>
<tr>
<th>CPU module</th>
<th>Setting range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic model QCPU, High Performance model QCPU, Process CPU, Redundant CPU</td>
<td>0.1ms to 100ms</td>
</tr>
<tr>
<td>Universal model QCPU, LCPU</td>
<td>0.01ms to 100ms</td>
</tr>
</tbody>
</table>

Valid setting range for n is T#0ms to T#327670ms.

**Operation result**

(1) **Function without EN/ENO**

An operation is executed and the operation value is output from \(d1\) and \(d2\).

<table>
<thead>
<tr>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Timing chart" /></td>
</tr>
</tbody>
</table>

When \(n = T#5s\) (5 seconds)

(2) **Function with EN/ENO**

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>d1, d2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>
5.12 Standard Timer Function Blocks

5.12.1 Pulse timer

**APPLICATION**

**FUNCTIONS**

TP(_E), TP_HIGH(_E)

---

**Timing chart**

When \( n = T\#5s \) (5 seconds)

---

**Operation Error**

No operation error occurs in the execution of the TP (_E) function.

**Program Example**

The program which turns ON bit type data of \( \) for 10 seconds after bit type data input to \( \) is turned ON.

(a) Function without EN/ENO (TP)

[Structured ladder/FBD]

```plaintext
18
TP_Inst(IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1 );
```

(b) Function with EN/ENO (TP_E)

[Structured ladder/FBD]

```plaintext
17
TP_E_Inst(EN:= M0 ,IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1,
ENO:= M10 );
```
5.12.2 On delay timer

**TON(_E), TON_HIGH(_E)**

*e: With EN/ENO*

**Function**

**Operation processing**

Turns ON ① when ① is turned ON after the elapse of the time set to n. Elapsed delay time until ① is turned ON is set to ②.

When ② is turned OFF, ③ turns OFF and the elapsed delay time is reset.

(1) **TON(_E)**

Uses a low-speed timer to count the elapsed time.

Output time can be set between 1ms and 1000ms. The unit is set in Timer limit setting on the PLC system of PLC parameter.

Valid setting range for n is T#0ms to T#3276700ms.
(2) **TON_HIGH(_E)**

Uses a high-speed timer to count the elapsed time.
Output time can be set within the following range. The unit is set in Timer limit setting on the
PLC system of PLC parameter.

<table>
<thead>
<tr>
<th>CPU module</th>
<th>Setting range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic model QCPU, High Performance model QCPU, Process CPU, Redundant CPU</td>
<td>0.1ms to 100ms</td>
</tr>
<tr>
<td>Universal model QCPU, LCPU</td>
<td>0.01ms to 100ms</td>
</tr>
</tbody>
</table>

Valid setting range for n is T#0ms to T#327670ms.

**Operation result**

(1) Function without EN/ENO

An operation is executed and the operation value is output from \( d1 \) and \( d2 \).

<table>
<thead>
<tr>
<th>Timing chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>When ( n = T#5s ) (5 seconds)</td>
</tr>
</tbody>
</table>

(2) Function with EN/ENO

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>( d1 ), ( d2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timing chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>When ( n = T#5s ) (5 seconds)</td>
</tr>
</tbody>
</table>
5.12 Standard Timer Function Blocks

5.12.2 On delay timer

**Operation Error**

No operation error occurs in the execution of the TON (_E) function.

**Program Example**

The program which turns ON bit type data of \( \oplus \) 10 seconds after bit type data input to \( \oplus \) is turned ON.

(a) Function without EN/ENO (TON)

[Structured ladder/FBD]

```
TON_Inst(IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1);
```

[ST]

```
TON_Inst(IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1);
```

(b) Function with EN/ENO (TON_E)

[Structured ladder/FBD]

```
TON_E_Inst(EN:= M0 ,IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1,
ENO:= M10);
```

[ST]

```
TON_E_Inst(EN:= M0 ,IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1,
ENO:= M10);
```
5.12.3 Off delay timer

TOF(_E), TOF_HIGH(_E)

TOF(_E), TOF_HIGH(_E)  
_E: With EN/ENO

Structure ladder/FBD

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN ENO s d1 d2</td>
<td></td>
</tr>
</tbody>
</table>

Input argument, EN: Executing condition (TRUE: Execution, FALSE: Stop)  
s(IN): Input  
n(PT): Delay time setting value
Output argument, ENO: Execution result (TRUE: Normal, FALSE: Error or stop)  
  d1(Q): Output  
  d2(ET): Elapsed time

Function

Operation processing

Turns ON  when  is turned ON.

Turns OFF  when  is turned from ON to OFF after the elapse of the time set to n. Elapsed time until  is turned OFF is set to .

When  is turned ON again,  turns ON and the elapsed time is reset.

(1) TOF(_E)

Uses a low-speed timer to count the elapsed time.

Output time can be set between 1ms and 1000ms. The unit is set in Timer limit setting on the PLC system of PLC parameter.

Valid setting range for n is T#0ms to T#3276700ms.
(2) **TOF_HIGH(_E)**

Uses a high-speed timer to count the elapsed time. Output time can be set within the following range. The unit is set in Timer limit setting on the PLC system of PLC parameter.

<table>
<thead>
<tr>
<th>CPU module</th>
<th>Setting range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic model QCPU, High Performance model QCPU, Process CPU, Redundant CPU</td>
<td>0.1ms to 100ms</td>
</tr>
<tr>
<td>Universal model QCPU, LPCPU</td>
<td>0.01ms to 100ms</td>
</tr>
</tbody>
</table>

Valid setting range for n is T#0ms to T#327670ms.

**Operation result**

(1) **Function without EN/ENO**

An operation is executed and the operation value is output from ③ and ④.

**[Timing chart]**

When \( n = T\#5s \) (5 seconds)

- ③ OFF
- ① OFF
- ② OFF
- ④ ON
- ⑤ ON
- ⑥ ON

Starts measuring ② when ③ is OFF.

Turns ③ OFF when ② reaches the time set to \( n \).

Sets ② when ③ = ON.

(2) **Function with EN/ENO**

The following table shows the executing conditions and operation results.

<table>
<thead>
<tr>
<th>EN</th>
<th>ENO</th>
<th>③, ④</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

**[Timing chart]**

When \( n = T\#5s \) (5 seconds)

- ③ OFF
- ① OFF
- ② OFF
- ④ ON
- ⑤ ON
- ⑥ ON

Starts measuring ② when EN = ON and ③ = OFF.

Resets ② when EN = ON and ③ = ON.
5.12 Standard Timer Function Blocks

5.12.3 Off delay timer

**Operation Error**

No operation error occurs in the execution of the TOF(_E) function.

**Program Example**

The program which turns ON bit type data of $\mathbb{1}$ when bit type data input to $\mathbb{2}$ is turned ON, and turns $\mathbb{3}$ OFF 10 seconds after $\mathbb{4}$ is turned OFF.

(a) Function without EN/ENO (TOF)

[Structured ladder/FBD]

[ST]

```
TOF_Inst(IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1 );
```

(b) Function with EN/ENO (TOF_E)

[Structured ladder/FBD]

[ST]

```
TOF_E_Inst(EN:= M0 ,IN:= g_bool1 ,PT:= T#10s ,Q:= g_bool2 ,ET:= g_time1 ,
ENO:= M10 );
```
5.12.4 Timer function blocks

**Function**

**Operation processing**

1. **TIMER_10_FB_M**
   
   (a) Starts measuring the current value when the executing condition of \( \neq 0 \) turns ON.
   
   Starts measuring from the value input to \( \neq 0 \) × 10ms, and when the measuring value reaches to the value input to \( \neq 0 \) × 10ms, \( \neq 0 \) turns ON.
   
   The current value is output from \( \neq 0 \).

   (b) When the executing condition of \( \neq 0 \) turns OFF, the current value is set to the value input to \( \neq 0 \), and \( \neq 0 \) turns OFF.

   (c) When the unit of measurement (time period) for the high-speed timer is changed from default value of PLC parameter, warning C9047 occurs in compilation.

   (d) Valid setting range for \( \neq 0 \) is 0 to 32767.

   (e) Valid setting range for \( \neq 0 \) is -32768 to 32767. However, if negative value is specified, the initial value is 0.
(2) TIMER_100_FB_M

(a) Starts measuring the current value when the executing condition of \( s_1 \) turns ON.
   Starts measuring from the value input to \( s_2 \times 100\text{ms} \), and when the measuring value
   reaches to the value input to \( s_2 \times 100\text{ms} \), \( s_3 \) turns ON.
   The current value is output from \( s_2 \).

(b) When the executing condition of \( s_1 \) turns OFF, the current value is set to the value input
    to \( s_2 \), and \( s_3 \) turns OFF.

(c) When the unit of measurement (time period) for the low-speed timer is changed from
    default value of PLC parameter, warning C9047 occurs in compilation.

(d) Valid setting range for \( s_2 \) is 0 to 32767.

(e) Valid setting range for \( s_3 \) is -32768 to 32767. However, if negative value is specified,
    the initial value is 0.

(3) TIMER_HIGH_FB_M

(a) The high-speed timer with the unit of measurement from 0.1 to 100ms. Starts
    measuring the current value when the executing condition of \( s_1 \) turns ON.
    Starts measuring from the value input to \( s_2 \times 0.1 \text{ to } 100\text{ms} \), and when the measuring
    value reaches to the value input to \( s_2 \times 0.1 \text{ to } 100\text{ms} \), \( s_3 \) turns ON.
    The current value is output from \( s_2 \).

(b) When the executing condition of \( s_1 \) turns OFF, the current value is set to the value input
    to \( s_2 \), and \( s_3 \) turns OFF.

(c) The default value of the unit of measurement (time period) for the high-speed timer is
    10ms.
    The unit of measurement can be changed within the following range.
    This setting is set in the PLC system setting of the PLC parameter.

<table>
<thead>
<tr>
<th>CPU module</th>
<th>Setting range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic model QCPU, High-Performance model QCPU, Process CPU, Redundant CPU</td>
<td>0.1ms to 100ms</td>
</tr>
<tr>
<td>Universal model QCPU, LCPU</td>
<td>0.01ms to 100ms</td>
</tr>
</tbody>
</table>

(d) Valid setting range for \( s_2 \) is 0 to 32767.

(e) Valid setting range for \( s_3 \) is -32768 to 32767. However, if negative value is specified,
    the initial value is 0.
(4) TIMER_LOW_FB_M

(a) The low-speed timer with the unit of measurement from 1 to 1000ms. Starts measuring the current value when the executing condition of \( s_1 \) turns ON. Starts measuring from the value input to \( s_2 \times 1 \) to 1000ms, and when the measuring value reaches to the value input to \( s_2 \times 1 \) to 1000ms, \( s_2 \) turns ON. The current value is output from \( s_3 \).

(b) When the executing condition of \( s_1 \) turns OFF, the current value is set to the value input to \( s_2 \), and \( s_2 \) turns OFF.

(c) The default value of the unit of measurement (time period) for the low-speed timer is 100ms. The unit of measurement is from 1 to 1000ms and it can be changed by unit of 1ms. This setting is set in the PLC system setting of the PLC parameter.

(d) Valid setting range for \( s_2 \) is 0 to 32767.

(e) Valid setting range for \( s_2 \) is -32768 to 32767. However, if negative value is specified, the initial value is 0.

(5) TIMER_CONT_FB_M, TIMER_CONTHFB_M

(a) The retentive timer that measures the time during variable is ON. Starts measuring the current value when the executing condition of \( s_1 \) turns ON. The low-speed retentive timer (TIMER_CONT_FB_M) and the high-speed retentive timer (TIMER_CONTHFB_M) are the two types of retentive timer. Starts measuring from the value input to \( s_2 \times 1 \) to 1000ms, and when the count value reaches to the value input to \( s_2 \times 1 \) to 1000ms, \( s_2 \) turns ON. The current value is output from \( s_3 \).

(b) Even when the executing condition of \( s_1 \) turns OFF, the ON/OFF statuses of measuring value \( s_2 \) and \( s_2 \) are retained. When the executing condition of \( s_1 \) turns ON again, restarts measuring from the values that are retained.

(c) The unit of measurement (time period) for retentive timer is same as the low-speed timer (TIMER_LOW_FB_M) and the high-speed timer (TIMER_HIGH_FB_M).

- Low-speed retentive timer : Low-speed timer
- High-speed retentive timer : High-speed timer

(d) Valid setting range for \( s_2 \) is 0 to 32767.

(e) Valid setting range for \( s_2 \) is -32768 to 32767. However, if negative value is specified, the initial value is 0.
(f) When resetting the current value of the retentive timer, reset \( \text{RST}(\text{Coil}) \).

(Example) When instance name is TIMER_CONT_FB_M_1.

[Structured ladder/FBD]

[ST]

\[
\text{TIMER\_CONT\_FB\_M\_1(}\text{Coil:=Var\_M0, Preset:=10, ValueIn:=0, ValueOut:=Var\_D10, Status:=Var\_M10)};
\]

\[
\text{RST(M15,TIMER\_CONT\_FB\_M\_1.Ccoil)};
\]

---

Operation Error

No operation error occurs in the execution of the timer function blocks.
### Program Example

1. **TIMER\_10\_FB\_M**

   The program which starts measuring from $\varnothing \times 10$ ms when the executing condition of $\varnothing$ turns ON, and when the measuring value reaches to the value input to $\varnothing \times 10$ ms, $\varnothing$ turns ON.

   [Structured ladder/FBD]

   ```
   TIMER_10_FB_M_Inst(Coil:= Var_M0 ,Preset:= 10 ,ValueIn:= 1 ,ValueOut:= Var_D10 ,Status:= Var_M10);
   ```

2. **TIMER\_HIGH\_FB\_M**

   The program which starts measuring from $\varnothing \times 10$ ms when the executing condition of $\varnothing$ turns ON, and when the measuring value reaches to the value input to $\varnothing \times 10$ ms, $\varnothing$ turns ON.

   [Structured ladder/FBD]

   ```
   TIMER_HIGH_FB_M_Inst(Coil:= Var_M0 ,Preset:= 10 ,ValueIn:= 1 ,ValueOut:= Var_D10 ,Status:= Var_M10);
   ```
(3) TIMER_LOW_FB_M

The program which starts measuring from $\times 10\text{ms}$ when the executing condition of $\times$ turns ON, and when the measuring value reaches to the value input to $\times 100\text{ms}$, $\times$ turns ON.

[Structured ladder/FBD]

[ST]

```
TIMER_LOW_FB_M_Inst(Coil:= Var_M0 ,Preset:= 10 ,ValueIn:= 1 ,ValueOut:= Var_D10 ,Status:= Var_M10);
```

[Timing chart]

(4) TIMER_CONT_FB_M

The program which measures from $\times 10\text{ms}$, and when the measuring value reaches to the value input to $\times 100\text{ms}$, $\times$ turns ON.

[Structured ladder/FBD]

[ST]

```
TIMER_CONT_FB_M_Inst(Coil:= Var_M0 ,Preset:= 200 ,ValueIn:= 0 ,ValueOut:= Var_D10 ,Status:= Var_M10);
```

[Timing chart]
OPERATOR

6.1 Arithmetic Operations .................................................. 6-2
6.2 Logical Operations ...................................................... 6-13
6.3 Comparison Operations ............................................... 6-16
6.1 Arithmetic Operations

6.1.1 Addition

ADD, +

**Operation processing**

For details of the operation processing, refer to Section 5.3.1.

**Operation Error**

No operation error occurs in the execution of the ADD and + operations.
Program Example

The program which performs addition (s1 + s2) on double word (signed) type data input to s1 and s2, and outputs the operation result from s3 in the same data type as that of s1 and s2.

[Structured ladder/FBD]

[ST]

\[ g_{\text{dint}3} := (g_{\text{dint}1}) + (g_{\text{dint}2}); \]
6.1.2 Multiplication

MUL, *

**Function**

**Operation processing**

For details of the operation processing, refer to Section 5.3.2.

**Operation Error**

No operation error occurs in the execution of the MUL and * operations.
### Program Example

The program which performs multiplication \((\text{ } \times \text{ })\) on double word (signed) type data input to \(\text{1}\) and \(\text{2}\), and outputs the operation result from \(\text{3}\) in the same data type as that of \(\text{1}\) and \(\text{2}\).

**[Structured ladder/FBD]**

![Structured ladder/FBD diagram]

**ST**

\[
g_\text{dint3} := (g_\text{dint1}) \times (g_\text{dint2});
\]
6.1.3 Subtraction

SUB, -

Function

Operation processing

For details of the operation processing, refer to Section 5.3.3.

Operation Error

No operation error occurs in the execution of the SUB and - operations.
The program which performs subtraction \((\text{l} - \text{r})\) on double word (signed) type data input to \(\text{l}\) and \(\text{r}\), and outputs the operation result from \(\text{r}\) in the same data type as that of \(\text{l}\) and \(\text{r}\).

[Structured ladder/FBD]

[ST]

\[
g_{\_dint3} := (g_{\_dint1}) - (g_{\_dint2});
\]
6.1.4 Division

DIV, /

Function

Operation processing
For details of the operation processing, refer to Section 5.3.4.

Operation Error

An operation error occurs in the following case.

- The value to be input to \( \) is 0. (Division by 0) (Error code: 4100)
Program Example

The program which performs division (\(\div\)) on double word (signed) type data input to \(\texttt{s1}\) and \(\texttt{s2}\), and outputs the quotient of the operation result from \(\texttt{s2}\) in the same data type as that of \(\texttt{s1}\) and \(\texttt{s2}\).

[Structured ladder/FBD]

[ST]

\[
g_{\text{dint3}} := (g_{\text{dint1}}) / (g_{\text{dint2}});
\]
6.1.5 Modules operation

**Function**

**Operation processing**

For details of the operation processing, refer to Section 5.3.5.

**Operation Error**

An operation error occurs in the following case.

- The value to be input to $\div$ is 0. (Division by 0)  (Error code: 4100)

**Program Example**

The program which performs division ($\div$) on double word (signed) type data input to $\div$ and $\div$, and outputs the remainder of the operation result from $\div$ in the same data type as that of $\div$ and $\div$.

[ST]

$$g_{\text{dint}3} := (g_{\text{dint}1}) \text{MOD} (g_{\text{dint}2});$$
6.1.6 Exponentiation

**

Function

Operation processing

\[ d = s_1^{s_2}; \]

For details of the operation processing, refer to Section 5.3.6.

Operation Error

These operators consist of the following common instructions.

- When \( s_1 \) is single-precision real number, \( s_2 \) is word (signed): LOG, FLT
- When \( s_1 \) is single-precision real number, \( s_2 \) is double word (signed): LOG, DFLT
- When \( s_1 \) is single-precision real number, \( s_2 \) is single-precision real number: LOG
- When \( s_1 \) is single-precision real number, \( s_2 \) is double-precision real number: LOGD, DFLTD
- When \( s_1 \) is double-precision real number, \( s_2 \) is word (signed): LOGD
- When \( s_1 \) is double-precision real number, \( s_2 \) is double word (signed): LOGD, FLTD
- When \( s_1 \) is double-precision real number, \( s_2 \) is single-precision real number: LOGD, DFLTD
- When \( s_1 \) is double-precision real number, \( s_2 \) is double-precision real number: LOGD

For details of an error which occurs when the operation is executed, refer to MELSEC-Q/L Structured Programming Manual (Common Instructions).
Program Example

The program which performs exponentiation and outputs the operation result from $@$ in the same data type as that of $\oplus$ and $\ominus$.

[ST]
\[ g_{\text{real}2} := (g_{\text{real}1})^{\ast} (g_{\int}1); \]
6.2 Logical Operations

6.2.1 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT

AND, &, OR, XOR, NOT

AND
$
$
OR
XOR
NOT

Structured ladder/FBD

ST

Structured ladder/FBD

ST

Input argument, s1 to s28: Input
(s1 only for NOT)
Output argument, d: Output

■ Function

Operation processing
For details of the operation processing, refer to Section 5.4.1.

Operation Error
No operation error occurs in the execution of the AND, &, OR, XOR, and NOT operations.
Program Example

(1) The program which performs Boolean AND on bit, word (unsigned)/16-bit string type data input to \( s_1 \) to \( s_2 \) bit by bit, and outputs the operation result from \( d \) in the same data type as that of \( s_1 \) to \( s_2 \).

\[ g_{\text{word3}} := (g_{\text{word1}}) \text{AND} (g_{\text{word2}}); \]
\[ \text{or} \quad g_{\text{word3}} := (g_{\text{word1}}) \& (g_{\text{word2}}); \]

(2) The program which performs Boolean OR on bit, word (unsigned)/16-bit string type data input to \( s_1 \) to \( s_2 \) bit by bit, and outputs the operation result from \( d \) in the same data type as that of \( s_1 \) to \( s_2 \).

\[ g_{\text{word3}} := (g_{\text{word1}}) \text{OR} (g_{\text{word2}}); \]

(3) The program which performs Boolean XOR on bit, word (unsigned)/16-bit string type data input to \( s_1 \) to \( s_2 \) bit by bit, and outputs the operation result from \( d \) in the same data type as that of \( s_1 \) to \( s_2 \).

\[ g_{\text{word3}} := (g_{\text{word1}}) \text{XOR} (g_{\text{word2}}); \]
(4) The program which performs Boolean NOT on bit, word (unsigned)/16-bit string type data input to bit by bit, and outputs the operation result from in the same data type as that of.

[Structured ladder/FBD]

[ST]

```
g_word2 := NOT (g_word1);
```
6.3 Comparison Operations

6.3.1 Comparison

GT, GE, EQ, LE, LT, NE, >, >=, =, <=, <, <>

Function

Operation processing

For details of the operation processing, refer to Section 5.6.1.

Operation Error

No operation error occurs in the execution of the GT, GE, EQ, LE, LT, NE, >, >=, =, <=, < and <> operations.
Program Example

The program which performs comparison operation between the values input to \( g_1 \) and \( g_2 \), and outputs the operation result from \( g \).

[Structured ladder/FBD]

```plaintext
[ST]
g_bool1 := (g_int1) > (g_int2);
```

\[ \text{g_bool1 := (g_int1) > (g_int2);} \]
<table>
<thead>
<tr>
<th>[L]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LE</td>
<td>6-16</td>
</tr>
<tr>
<td>LE(E)</td>
<td>5-174</td>
</tr>
<tr>
<td>LIMITATION(E)</td>
<td>5-168</td>
</tr>
<tr>
<td>Logical operations table</td>
<td>2-8</td>
</tr>
<tr>
<td>LREAL_TO_DINT(E)</td>
<td>5-44</td>
</tr>
<tr>
<td>LREAL_TO_INT(E)</td>
<td>5-44</td>
</tr>
<tr>
<td>LREAL_TO_REAL(E)</td>
<td>5-50</td>
</tr>
<tr>
<td>LT</td>
<td>6-16</td>
</tr>
<tr>
<td>LTI(E)</td>
<td>5-174</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[M]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXIMUM(E)</td>
<td>5-165</td>
</tr>
<tr>
<td>MID(E)</td>
<td>5-178</td>
</tr>
<tr>
<td>MINIMUM(E)</td>
<td>5-165</td>
</tr>
<tr>
<td>MOD</td>
<td>6-10</td>
</tr>
<tr>
<td>MODI(E)</td>
<td>5-147</td>
</tr>
<tr>
<td>MOVE(E)</td>
<td>5-153</td>
</tr>
<tr>
<td>MUL</td>
<td>6-4</td>
</tr>
<tr>
<td>MUL(E)</td>
<td>5-138</td>
</tr>
<tr>
<td>MUL_TIME(E)</td>
<td>5-199</td>
</tr>
<tr>
<td>MUX(E)</td>
<td>5-171</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[N]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NE</td>
<td>6-16</td>
</tr>
<tr>
<td>NE(E)</td>
<td>5-174</td>
</tr>
<tr>
<td>NOT</td>
<td>6-13</td>
</tr>
<tr>
<td>NOT(E)</td>
<td>5-157</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[O]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>6-13</td>
</tr>
<tr>
<td>OR(E)</td>
<td>5-157</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[R]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL_TO_DINT(E)</td>
<td>5-41</td>
</tr>
<tr>
<td>REAL_TO_INT(E)</td>
<td>5-41</td>
</tr>
<tr>
<td>REAL_TO_LREAL(E)</td>
<td>5-47</td>
</tr>
<tr>
<td>REAL_TO_STR(E)</td>
<td>5-53</td>
</tr>
<tr>
<td>REPLACE(E)</td>
<td>5-190</td>
</tr>
<tr>
<td>RS(E)</td>
<td>5-207</td>
</tr>
<tr>
<td>R_TRIG(E)</td>
<td>5-210</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[S]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL(E)</td>
<td>5-162</td>
</tr>
<tr>
<td>SET_BIT_OF_INT(E)</td>
<td>5-125</td>
</tr>
<tr>
<td>SR(E)</td>
<td>5-204</td>
</tr>
<tr>
<td>Standard arithmetic functions table</td>
<td>2-5</td>
</tr>
<tr>
<td>Standard bistable function blocks table</td>
<td>2-6</td>
</tr>
<tr>
<td>Standard bitwise Boolean functions table</td>
<td>2-5</td>
</tr>
<tr>
<td>Standard character string functions table</td>
<td>2-6</td>
</tr>
<tr>
<td>Standard comparison functions table</td>
<td>2-6</td>
</tr>
<tr>
<td>Standard counter function blocks table</td>
<td>2-7</td>
</tr>
</tbody>
</table>

Standard edge detection function blocks table | 2-7 |
Standard functions of one numeric variable table | 2-5 |
Standard selection functions table | 2-5 |
Standard timer function blocks table | 2-7 |
STR_TO_BCD(E) | 5-95 |
STR_TO_BOOL(E) | 5-78 |
STR_TO_DINT(E) | 5-81 |
STR_TO_DWORD(E) | 5-88 |
STR_TO_INT(E) | 5-81 |
STR_TO_REAL(E) | 5-84 |
STR_TO_TIME(E) | 5-92 |
STR_TO_WORD(E) | 5-88 |
SUB | 6-6 |
SUB(E) | 5-141 |
SUB_TIME(E) | 5-196 |

<table>
<thead>
<tr>
<th>[T]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_TO_BOOL(E)</td>
<td>5-107</td>
</tr>
<tr>
<td>TIME_TO_DINT(E)</td>
<td>5-109</td>
</tr>
<tr>
<td>TIME_TO_DWORD(E)</td>
<td>5-114</td>
</tr>
<tr>
<td>TIME_TO_INT(E)</td>
<td>5-109</td>
</tr>
<tr>
<td>TIME_TO_STR(E)</td>
<td>5-112</td>
</tr>
<tr>
<td>TIME_TO_WORD(E)</td>
<td>5-114</td>
</tr>
<tr>
<td>TOF(E)</td>
<td>5-233</td>
</tr>
<tr>
<td>TOF_HIGH(E)</td>
<td>5-233</td>
</tr>
<tr>
<td>TON(E)</td>
<td>5-230</td>
</tr>
<tr>
<td>TON_HIGH(E)</td>
<td>5-230</td>
</tr>
<tr>
<td>TP(E)</td>
<td>5-227</td>
</tr>
<tr>
<td>TP_HIGH(E)</td>
<td>5-227</td>
</tr>
<tr>
<td>Type conversion functions table</td>
<td>2-3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[W]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD_TO_BOOL(E)</td>
<td>5-57</td>
</tr>
<tr>
<td>WORD_TO_DINT(E)</td>
<td>5-60</td>
</tr>
<tr>
<td>WORD_TO_DWORD(E)</td>
<td>5-66</td>
</tr>
<tr>
<td>WORD_TO_INT(E)</td>
<td>5-60</td>
</tr>
<tr>
<td>WORD_TO_STR(E)</td>
<td>5-72</td>
</tr>
<tr>
<td>WORD_TO_TIME(E)</td>
<td>5-75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[X]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>6-13</td>
</tr>
<tr>
<td>XOR(E)</td>
<td>5-157</td>
</tr>
</tbody>
</table>
WARRANTY

Please confirm the following product warranty details before using this product.

1. Gratis Warranty Term and Gratis Warranty Range
   If any faults or defects (hereinafter "Failure") found to be the responsibility of Mitsubishi occurs during use of the product within the gratis warranty term, the product shall be repaired at no cost via the sales representative or Mitsubishi Service Company. However, if repairs are required onsite at domestic or overseas location, expenses to send an engineer will be solely at the customer's discretion. Mitsubishi shall not be held responsible for any re-commissioning, maintenance, or testing onsite 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.
   1. Failure occurring from inappropriate storage or handling, carelessness or negligence by the user. Failure caused by the user's hardware or software design.
   2. Failure caused by unapproved modifications, etc., to the product by the user.
   3. 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.
   4. Failure that could have been avoided if consumable parts (battery, backlight, fuse, etc.) designated in the instruction manual had been correctly serviced or replaced.
   5. 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.
   6. Failure caused by reasons unpredictable by scientific technology standards at time of shipment from Mitsubishi.
   7. 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 to damages caused by any cause found not to be the responsibility of Mitsubishi, loss in opportunity, lost profits incurred to the user 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.
MELSEC-Q/L Structured Programming Manual

Application Functions

MODEL: Q-KP-OK-E
MODEL CODE: 13JW08
SH(NA)-080784ENG-K(1306)KWIX

MITSUBISHI ELECTRIC CORPORATION
HEAD OFFICE: TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHYODA-KU, TOKYO 100-8018, JAPAN
NAGOYA WORKS: 1-14, YADA-MINAMISHO-ME, HIGASHI-ku, NAGOYA, JAPAN

When exported from Japan, this manual does not require application to the Ministry of Economy, Trade and Industry for service transaction permission.

Specifications subject to change without notice.