SAFETY PRECAUTIONS
(Read these precautions before using this product.)

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.

CONDITIONS OF USE FOR THE PRODUCT

(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.
INTRODUCTION
Thank you for purchasing the Mitsubishi Electric 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 ................................................................. 1
CONDITIONS OF USE FOR THE PRODUCT .................................. 1
INTRODUCTION ........................................................................... 2
MANUALS ................................................................................... 8
TERMS ....................................................................................... 9
GENERIC TERMS AND ABBREVIATIONS .................................... 9

CHAPTER 1 OVERVIEW ............................................................... 10
1.1 Purpose of This Manual ....................................................... 10

CHAPTER 2 FUNCTION TABLES .................................................. 12
2.1 How to Read Function Tables .............................................. 12
2.2 Function Tables .................................................................. 13
   Type conversion functions .................................................... 13
   Standard functions of one numeric variable ......................... 16
   Standard arithmetic functions ............................................. 16
   Standard bitwise Boolean functions .................................... 16
   Standard selection functions .............................................. 16
   Standard comparison functions ......................................... 17
   Standard character string functions .................................... 17
   Functions of time data types .............................................. 17
   Standard bistable function blocks ....................................... 17
   Standard edge detection function blocks .............................. 18
   Standard counter function blocks ...................................... 18
   Standard timer function blocks .......................................... 18
2.3 Operator Tables ................................................................. 19
   Arithmetic operations .......................................................... 19
   Logical operations .............................................................. 19
   Comparison operations ....................................................... 19

CHAPTER 3 CONFIGURATION OF FUNCTIONS ......................... 20
3.1 Configuration of Functions .................................................. 20
3.2 Input Pins Variable Function ............................................... 21

CHAPTER 4 HOW TO READ FUNCTIONS .................................. 22

CHAPTER 5 APPLICATION FUNCTIONS .................................... 24
5.1 Type Conversion Functions .................................................. 24
   Converting bit type to word (signed), double word (signed) type ........................................ 24
   Converting bit type to string type ....................................... 26
   Converting bit type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type .............................................................................................................. 28
   Converting bit type to time type ........................................ 30
   Converting word (signed) type to double word (signed) type ............................................. 32
   Converting double word (signed) type to word (signed) type ........................................... 34
   Converting word (signed), double word (signed) type to bit type .................................... 36
   Converting word (signed), double word (signed) type to single-precision real type .......... 38
   Converting word (signed), double word (signed) type to double-precision real type ........ 40
   Converting word (signed), double word (signed) type to string type ................................. 42
5.2 Standard Functions of One Numeric Variable

5.3 Standard Arithmetic Functions

5.4 Standard Bitwise Boolean Functions

5.5 Standard Selection Functions
Maximum/Minimum selection .................................................. 146
Upper/Lower limit control ...................................................... 149
Multiplexer ........................................................................... 151
5.6 Standard Comparison Functions ......................................... 153
Comparison ........................................................................... 153
5.7 Standard Character String Functions .................................... 155
Extract mid string ................................................................ 155
String concatenation ............................................................... 157
String insertion .................................................................... 159
String deletion ...................................................................... 161
String replacement ................................................................. 163
5.8 Functions of Time Data Type ................................................ 166
Addition ................................................................................ 166
Subtraction .......................................................................... 168
Multiplication ........................................................................ 170
Division ................................................................................ 172
5.9 Standard Bistable Function Blocks ........................................ 174
Standard bistable function blocks (Set-dominant) ....................... 174
Standard bistable function blocks (Reset-dominant) .................... 176
5.10 Standard Edge Detection Function Blocks .............................. 178
Rising edge detector ............................................................... 178
Falling edge detector ............................................................. 180
5.11 Standard Counter Function Blocks ....................................... 182
Up counter ............................................................................ 182
Down counter ...................................................................... 184
Up/Down counter .................................................................. 186
Counter function blocks ......................................................... 190
5.12 Standard Timer Function Blocks .......................................... 192
Pulse timer ............................................................................ 192
On delay timer ...................................................................... 195
Off delay timer ..................................................................... 198
Timer function blocks ............................................................. 201

CHAPTER 6 OPERATOR .................................................................. 205
6.1 Arithmetic Operations ......................................................... 205
Addition ................................................................................ 205
Multiplication ....................................................................... 207
Subtraction .......................................................................... 208
Division ................................................................................ 209
Remainder ............................................................................ 211
Exponentiation ..................................................................... 212
6.2 Logical Operations ............................................................. 213
Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT .... 213
6.3 Comparison Operations ....................................................... 215
Comparison .......................................................................... 215

INDEX .......................................................................................... 216
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTRUCTION INDEX</td>
<td>218</td>
</tr>
<tr>
<td>REVISIONS</td>
<td>220</td>
</tr>
<tr>
<td>WARRANTY</td>
<td>221</td>
</tr>
<tr>
<td>TRADEMARKS</td>
<td>222</td>
</tr>
</tbody>
</table>
# MANUALS

## RELEVANT MANUALS

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

### Structured programming

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

### Operation of GX Works2

<table>
<thead>
<tr>
<th>Manual name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GX Works2 Version 1 Operating Manual (Common)</td>
<td>System configuration, parameter settings, and online operations of GX Works2, which are common to Simple projects and Structured projects</td>
</tr>
<tr>
<td>&lt;SH-080779ENG&gt;</td>
<td></td>
</tr>
<tr>
<td>GX Works2 Version 1 Operating Manual (Structured Project) &lt;SH-080781ENG&gt;</td>
<td>Operations, such as programming and monitoring in Structured projects, of GX Works2</td>
</tr>
<tr>
<td>GX Works2 Beginner’s Manual (Structured Project) &lt;SH-080788ENG&gt;</td>
<td>Basic operations, such as programming, editing, and monitoring in Structured projects, of GX Works2. This manual is intended for first-time users of GX Works2.</td>
</tr>
</tbody>
</table>
## TERMS

Unless otherwise specified, this manual uses the following terms.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application function</td>
<td>A term for the functions, such as functions and function blocks, defined in IEC 61131-3. It is executed by combinations of multiple common instructions in a programmable controller.</td>
</tr>
<tr>
<td>GX Works2</td>
<td>A programming tool used for setting up programmable controllers, programming, debugging, and maintenance</td>
</tr>
</tbody>
</table>

## GENERIC TERMS AND ABBREVIATIONS

Unless otherwise specified, this manual uses the following generic terms and abbreviations.

<table>
<thead>
<tr>
<th>Generic term/abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic model QCPU</td>
<td>Q00CPU, Q00CPU, Q01CPU</td>
</tr>
<tr>
<td>Common instruction</td>
<td>A sequence instruction, a basic instruction, an application instruction, a data link instruction, a multiple CPU dedicated instruction, a multiple CPU high-speed transmission dedicated instruction, and a redundant system instruction</td>
</tr>
<tr>
<td>High Performance model QCPU</td>
<td>Q02CPU, Q02HCPU, Q06HCPU, Q12HCPU, Q25HCPU</td>
</tr>
<tr>
<td>IEC 61131-3</td>
<td>The IEC 61131-3 international standard</td>
</tr>
<tr>
<td>LCPU</td>
<td>L02SCPU, L02SCPU-P, L02CPU, L02CPU-P, L06CPU, L06CPU-P, L26CPU, L26CPU-P, L26CPU-BT, L26CPU-PBT</td>
</tr>
<tr>
<td>Personal computer</td>
<td>A personal computer where Windows® operates</td>
</tr>
<tr>
<td>Process CPU</td>
<td>Q02PHCPU, Q06PHCPU, Q12PHCPU, Q25PHCPU</td>
</tr>
<tr>
<td>QCPU (Q mode)</td>
<td>A Basic model QCPU, a High Performance model QCPU, a Process CPU, a Redundant CPU, and a Universal model QCPU</td>
</tr>
<tr>
<td>Redundant CPU</td>
<td>Q12PRHCPU, Q25PRHCPU</td>
</tr>
<tr>
<td>Special instruction</td>
<td>A module dedicated instruction, a PID control instruction, a socket communication function instruction, a built-in I/O function instruction, and a data logging function instruction</td>
</tr>
<tr>
<td>Universal model QCPU</td>
<td>Q00UCPU, Q00UCPU, Q01UCPU, Q02UCPU, Q03UDCPU, Q03UDVCPU, Q03UDECPU, Q04UDHCPU, Q04UDVCPU, Q04UDPVCPU, Q04UDEHCPU, Q04UDPVCPU, Q06UDHCPU, Q06UDVCPU, Q06UDPVCPU, Q06UDEHCPU, Q06UDPVCPU, Q10UDHCPU, Q10UDEHCPU, Q11UDHCPU, Q11UDVCPU, Q11UDEHCPU, Q20UDEHCPU, Q20UDHCPU, Q20UDVCPU, Q20UDPVCPU, Q25UDHCPU, Q25UDVCPU, Q25UDPVCPU, Q25UDEHCPU, Q25UDEHCPU, Q50UDEHCPU, Q100UDEHCPU</td>
</tr>
</tbody>
</table>
1 OVERVIEW

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 following.

Page 8 RELEVANT MANUALS

Operation of GX Works2

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Overview</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation</td>
<td>Learning the operating environment and installation method</td>
<td>—</td>
</tr>
<tr>
<td>Operation of GX Works2</td>
<td>Learning all functions of GX Works2</td>
<td>GX Works2 Version 1 Operating Manual (Common)</td>
</tr>
<tr>
<td></td>
<td>Learning the project types and available languages in GX Works2</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the basic operations and operating procedures when creating a simple project for the first time</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the basic operations and operating procedures when creating a structured project for the first time</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the operations of available functions regardless of project type</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the functions and operation methods for programming</td>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
</tr>
<tr>
<td></td>
<td>Learning data setting methods for intelligent function module</td>
<td>—</td>
</tr>
</tbody>
</table>

Operations in each programming language

For details of instructions used in each programming language, refer to the following.

Page 11 Details of instructions in each programming language

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Overview</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Project</td>
<td>Ladder</td>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
</tr>
<tr>
<td></td>
<td>SFC</td>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
</tr>
<tr>
<td></td>
<td>ST</td>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
</tr>
<tr>
<td>Structured Project/ FBD</td>
<td>Ladder</td>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
</tr>
<tr>
<td></td>
<td>SFC</td>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
</tr>
<tr>
<td></td>
<td>Structured ladder/FBD</td>
<td>GX Works2 Version 1 Operating Manual (Structured Project)</td>
</tr>
</tbody>
</table>

*1 MELSAP3 and FX series SFC only
### Details of instructions in each programming language

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Overview</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All languages</strong></td>
<td>Learning details of programmable controller CPU error codes, special relays, and special registers</td>
<td>—</td>
</tr>
<tr>
<td><strong>Using ladder language</strong></td>
<td>Learning the types and details of common instructions</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for intelligent function modules</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for network modules</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for the PID control function</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of the process control instructions</td>
<td>—</td>
</tr>
<tr>
<td><strong>Using SFC language</strong></td>
<td>Learning details of specifications, functions, and instructions of SFC (MELSAP3)</td>
<td>—</td>
</tr>
<tr>
<td><strong>Using structured ladder/FBD/ST language</strong></td>
<td>Learning the fundamentals for creating a structured program</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of common instructions</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for intelligent function modules</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for network modules</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of instructions for the PID control function</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of application functions</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Learning the types and details of the process control instructions</td>
<td>—</td>
</tr>
</tbody>
</table>
2 FUNCTION TABLES

2.1 How to Read Function Tables

Function tables in Page 13 Function Tables are shown in the following format.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>Source</td>
<td>Stores data before operation.</td>
</tr>
<tr>
<td>(d)</td>
<td>Destination</td>
<td>Indicates the destination of data after operation.</td>
</tr>
<tr>
<td>(Number of pins variable)</td>
<td>Allows the number of (s) (source) to be changed in the range from 2 to 28.</td>
<td></td>
</tr>
</tbody>
</table>

- Changing the number of pins

Description

① Indicates the functions used in a program. 'Function name(_E)' is used as a function with EN/ENO. Without "_E", it is used as a function without EN/ENO.

② Indicates the arguments of the function.

③ Indicates the processing details of each function.

④ Indicates the references on which the functions are explained.
### Type conversion functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_INT(_E)</td>
<td>(s), (d)</td>
<td>Converts bit type data into word (signed) or double word (signed) type data.</td>
<td>Page 24 Converting bit type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>BOOL_TO_DINT(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 24 Converting bit type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>BOOL_TO_STR(_E)</td>
<td>(s), (d)</td>
<td>Converts bit type data into string type data.</td>
<td>Page 26 Converting bit type to string type</td>
</tr>
<tr>
<td>BOOL_TO_WORD(_E)</td>
<td>(s), (d)</td>
<td>Converts bit type data into word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data.</td>
<td>Page 28 Converting bit type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>BOOL_TO_DWORD(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 28 Converting bit type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>BOOL_TO_TIME(_E)</td>
<td>(s), (d)</td>
<td>Converts bit type data into time type data.</td>
<td>Page 30 Converting bit type to time type</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) type data into double word (signed) type data.</td>
<td>Page 32 Converting word (signed) type to double word (signed) type</td>
</tr>
<tr>
<td>DINT_TO_INT(_E)</td>
<td>(s), (d)</td>
<td>Converts double word (signed) type data into word (signed) type data.</td>
<td>Page 34 Converting double word (signed) type to word (signed) type</td>
</tr>
<tr>
<td>INT_TO_BOOL(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into bit type data.</td>
<td>Page 36 Converting word (signed), double word (signed) type to bit type</td>
</tr>
<tr>
<td>DINT_TO_Bool(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 36 Converting word (signed), double word (signed) type to bit type</td>
</tr>
<tr>
<td>INT_TO_REAL(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into single-precision real type data.</td>
<td>Page 38 Converting word (signed), double word (signed) type to single-precision real type</td>
</tr>
<tr>
<td>DINT_TO_REAL(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 38 Converting word (signed), double word (signed) type to single-precision real type</td>
</tr>
<tr>
<td>INT_TO_LREAL(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into double-precision real type data.</td>
<td>Page 40 Converting word (signed), double word (signed) type to double-precision real type</td>
</tr>
<tr>
<td>DINT_TO_LREAL(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 40 Converting word (signed), double word (signed) type to double-precision real type</td>
</tr>
<tr>
<td>INT_TO_STR(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into string type data.</td>
<td>Page 42 Converting word (signed), double word (signed) type to string type</td>
</tr>
<tr>
<td>DINT_TO_STR(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 42 Converting word (signed), double word (signed) type to string type</td>
</tr>
<tr>
<td>INT_TO_WORD(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into word (unsigned)/16-bit string type data.</td>
<td>Page 45 Converting word (signed), double word (signed) type to word (unsigned)/16-bit string type</td>
</tr>
<tr>
<td>DINT_TO_WORD(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 45 Converting word (signed), double word (signed) type to word (unsigned)/16-bit string type</td>
</tr>
<tr>
<td>INT_TO_DWORD(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into double word (unsigned)/32-bit string type data.</td>
<td>Page 47 Converting word (signed), double word (signed) type to double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>DINT_TO_DWORD(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 47 Converting word (signed), double word (signed) type to double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>INT_TO_BCD(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into BCD type data.</td>
<td>Page 49 Converting word (signed), double word (signed) type to BCD type</td>
</tr>
<tr>
<td>DINT_TO_BCD(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 49 Converting word (signed), double word (signed) type to BCD type</td>
</tr>
<tr>
<td>INT_TO_TIME(_E)</td>
<td>(s), (d)</td>
<td>Converts word (signed) or double word (signed) type data into time type data.</td>
<td>Page 52 Converting word (signed), double word (signed) type to time type</td>
</tr>
<tr>
<td>DINT_TO_TIME(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 52 Converting word (signed), double word (signed) type to time type</td>
</tr>
<tr>
<td>REAL_TO_INT(_E)</td>
<td>(s), (d)</td>
<td>Converts single-precision real type data into word (signed) or double word (signed) type data.</td>
<td>Page 54 Converting single-precision real type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>REAL_TO_DINT(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 54 Converting single-precision real type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>LREAL_TO_INT(_E)</td>
<td>(s), (d)</td>
<td>Converts double-precision real type data into word (signed) or double word (signed) type data.</td>
<td>Page 56 Converting double-precision real type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>LREAL_TO_DINT(_E)</td>
<td>(s), (d)</td>
<td></td>
<td>Page 56 Converting double-precision real type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>REAL_TO_LREAL(_E)</td>
<td>(s), (d)</td>
<td>Converts single-precision real type data into double-precision real type data.</td>
<td>Page 58 Converting single-precision real type to double-precision real type</td>
</tr>
<tr>
<td>Function name</td>
<td>Argument</td>
<td>Processing details</td>
<td>Reference</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>LREAL_TO_REAL(E)</td>
<td>(s), (d)</td>
<td>Converts double-precision real type data into single-precision real type data.</td>
<td>Page 60 Converting double-precision real type to single-precision real type</td>
</tr>
<tr>
<td>REAL_TO_STR(E)</td>
<td>(s), (d)</td>
<td>Converts single-precision real type data into string type (exponential form) data.</td>
<td>Page 62 Converting single-precision real type to string type</td>
</tr>
<tr>
<td>WORD_TO_BOOL(E)</td>
<td>(s), (d)</td>
<td>Converts word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data into bit type data.</td>
<td>Page 65 Converting word (unsigned)/16-bit string, double word (unsigned)/32-bit string type to bit type</td>
</tr>
<tr>
<td>DWORD_TO_BOOL(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORD_TO_INT(E)</td>
<td>(s), (d)</td>
<td>Converts word (unsigned)/16-bit string type data into word (signed) or double word (signed) type data.</td>
<td>Page 67 Converting word (unsigned)/16-bit string type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>DWORD_TO_INT(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORD_TO_DINT(E)</td>
<td>(s), (d)</td>
<td>Converts double word (unsigned)/32-bit string type data into word (signed) or double word (signed) type data.</td>
<td>Page 69 Converting double word (unsigned)/32-bit string type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>DWORD_TO_DINT(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORD_TO_DWORD(E)</td>
<td>(s), (d)</td>
<td>Converts word (unsigned)/16-bit string type data into double word (unsigned)/32-bit string type data.</td>
<td>Page 71 Converting word (unsigned)/16-bit string type to double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>DWORD_TO_DWORD(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORD_TO_STR(E)</td>
<td>(s), (d)</td>
<td>Converts word (unsigned)/16-bit string, double word (unsigned)/32-bit string type data into string type data.</td>
<td>Page 75 Converting word (unsigned)/16-bit string, double word (unsigned)/32-bit string type to string type</td>
</tr>
<tr>
<td>DWORD_TO_STR(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORD_TO_TIME(E)</td>
<td>(s), (d)</td>
<td>Converts word (unsigned)/16-bit string, double word (unsigned)/32-bit string type data into time type data.</td>
<td>Page 77 Converting word (unsigned)/16-bit string, double word (unsigned)/32-bit string type to time type</td>
</tr>
<tr>
<td>DWORD_TO_TIME(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STR_TO_BOOL(E)</td>
<td>(s), (d)</td>
<td>Converts string type data into bit type data.</td>
<td>Page 79 Converting string type to bit type</td>
</tr>
<tr>
<td>STR_TO_INT(E)</td>
<td>(s), (d)</td>
<td>Converts string type data into word (signed) or double word (signed) type data.</td>
<td>Page 81 Converting string type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>STR_TO_DINT(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STR_TO_REAL(E)</td>
<td>(s), (d)</td>
<td>Converts string type data into single-precision real type data.</td>
<td>Page 84 Converting string type to single-precision real type</td>
</tr>
<tr>
<td>STR_TO_WORD(E)</td>
<td>(s), (d)</td>
<td>Converts string type data into word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data.</td>
<td>Page 88 Converting string type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>STR_TO_DWORD(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STR_TO_TIME(E)</td>
<td>(s), (d)</td>
<td>Converts string type data into time type data.</td>
<td>Page 91 Converting string type to time type</td>
</tr>
<tr>
<td>STR_TO_BCD(E)</td>
<td>(s), (d)</td>
<td>Converts string type data into BCD type data.</td>
<td>Page 93 Converting string type to BCD type</td>
</tr>
<tr>
<td>BCD_TO_INT(E)</td>
<td>(s), (d)</td>
<td>Converts BCD type data into word (signed) or double word (signed) type data.</td>
<td>Page 96 Converting BCD type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>BCD_TO_DINT(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCD_TO_STR(E)</td>
<td>(s), (d)</td>
<td>Converts BCD type data into string type data.</td>
<td>Page 99 Converting BCD type to string type</td>
</tr>
<tr>
<td>TIME_TO_BOOL(E)</td>
<td>(s), (d)</td>
<td>Converts time type data into bit type data.</td>
<td>Page 101 Converting time type to bit type</td>
</tr>
<tr>
<td>TIME_TO_INT(E)</td>
<td>(s), (d)</td>
<td>Converts time type data into word (signed) or double word (signed) type data.</td>
<td>Page 103 Converting time type to word (signed), double word (signed) type</td>
</tr>
<tr>
<td>TIME_TO_DINT(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME_TO_STR(E)</td>
<td>(s), (d)</td>
<td>Converts time type data into string type data.</td>
<td>Page 105 Converting time type to string type</td>
</tr>
<tr>
<td>TIME_TO_WORD(E)</td>
<td>(s), (d)</td>
<td>Converts time type data into word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data.</td>
<td>Page 107 Converting time type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>TIME_TO_DWORD(E)</td>
<td>(s), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function name</td>
<td>Argument</td>
<td>Processing details</td>
<td>Reference</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>BITARR_TO_INT(_E)</td>
<td>(s), n, (d)</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>Page 109 Converting bit array to word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, double word (unsigned)/32-bit string type</td>
</tr>
<tr>
<td>BITARR_TO_DINT(_E)</td>
<td>(s), n, (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT_TO_BITARR(_E)</td>
<td>(s), n, (d)</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>Page 111 Converting word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, double word (unsigned)/32-bit string type to bit array</td>
</tr>
<tr>
<td>DINT_TO_BITARR(_E)</td>
<td>(s), n, (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPY_BITARR(_E)</td>
<td>(s), n, (d)</td>
<td>Copies specified number of bits from bit array.</td>
<td>Page 113 Bit array copy</td>
</tr>
<tr>
<td>GET_BIT_OF_INT(_E)</td>
<td>(s), n, (d)</td>
<td>Reads a value of specified bit of word (signed) type data.</td>
<td>Page 115 Specified bit read of word (signed) type data</td>
</tr>
<tr>
<td>SET_BIT_OF_INT(_E)</td>
<td>(s), n, (d)</td>
<td>Writes a value to the specified bit of word (signed) type data.</td>
<td>Page 117 Specified bit write of word (signed) type data</td>
</tr>
<tr>
<td>CPY_BIT_OF_INT(_E)</td>
<td>(s), n1, n2, (d)</td>
<td>Copies a specified bit of word (signed) type data to the specified bit of another word (signed) type data.</td>
<td>Page 119 Specified bit copy of word (signed) type data</td>
</tr>
<tr>
<td>GET_BOOL_ADDR</td>
<td>(s), (d)</td>
<td>Converts the type of data into bit type.</td>
<td>Page 121 Nonessential type conversion</td>
</tr>
<tr>
<td>GET_INT_ADDR</td>
<td>(s), (d)</td>
<td>Converts the type of data into word (signed) type.</td>
<td></td>
</tr>
<tr>
<td>GET_WORD_ADDR</td>
<td>(s), (d)</td>
<td>Converts the type of data to word (unsigned)/16-bit string type.</td>
<td></td>
</tr>
</tbody>
</table>
Standard functions of one numeric variable

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(_E)</td>
<td>(s), (d)</td>
<td>Outputs the absolute value of an input value.</td>
<td>Page 123 Absolute value</td>
</tr>
</tbody>
</table>

Standard arithmetic functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_E</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the sum ((s1)+(s2)+ ... +(s28)) of input values.</td>
<td>Page 126 Addition</td>
</tr>
<tr>
<td>MUL_E</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the product ((s1)×(s2)× ... ×(s28)) of input values.</td>
<td>Page 128 Multiplication</td>
</tr>
<tr>
<td>SUB_E</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the difference ((s1)-(s2)) between input values.</td>
<td>Page 130 Subtraction</td>
</tr>
<tr>
<td>DIV_E</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the quotient ((s1) ÷ (s2)) of input values.</td>
<td>Page 132 Division</td>
</tr>
<tr>
<td>MOD(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the remainder after division of input values ((s1) ÷ (s2)).</td>
<td>Page 134 Remainder</td>
</tr>
<tr>
<td>EXPT(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the exponentiation of an input value.</td>
<td>Page 136 Exponentiation</td>
</tr>
<tr>
<td>MOVE(_E)</td>
<td>(s), (d)</td>
<td>Moves the input value to (d).</td>
<td>Page 138 Move operation</td>
</tr>
</tbody>
</table>

Standard bitwise Boolean functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND_E</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the Boolean AND of input values.</td>
<td>Page 140 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT</td>
</tr>
<tr>
<td>OR_E</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the Boolean OR of input values.</td>
<td></td>
</tr>
<tr>
<td>XOR_E</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the Boolean exclusive OR of input values.</td>
<td></td>
</tr>
<tr>
<td>NOT(_E)</td>
<td>(s), (d)</td>
<td>Outputs the Boolean NOT of input values.</td>
<td></td>
</tr>
</tbody>
</table>

Standard selection functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL(_E)</td>
<td>(s1), (s2), (s3), (d)</td>
<td>Outputs the value selected from the input values.</td>
<td>Page 144 Selection</td>
</tr>
<tr>
<td>MAXIMUM(_E)</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the maximum value of the input values.</td>
<td>Page 146 Maximum/Minimum selection</td>
</tr>
<tr>
<td>MINIMUM(_E)</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the minimum value of the input values.</td>
<td></td>
</tr>
<tr>
<td>LIMITATION(_E)</td>
<td>(s1), (s2), (s3), (d) (Number of pins variable)</td>
<td>Outputs the input value controlled by the upper and lower limit control.</td>
<td>Page 149 Upper/Lower limit control</td>
</tr>
<tr>
<td>MUX(_E)</td>
<td>(s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs one of the multiple input values.</td>
<td>Page 151 Multiplexer</td>
</tr>
</tbody>
</table>
### Standard comparison functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT_E</td>
<td>(s1), (s2), ...(s28), (d)</td>
<td>Outputs the comparison value of an input value.</td>
<td>Page 153 Comparison</td>
</tr>
<tr>
<td>GE_E</td>
<td>(s1), (s2), ...(s28), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQ_E</td>
<td>(s1), (s2), ...(s28), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LE_E</td>
<td>(s1), (s2), ...(s28), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT_E</td>
<td>(s1), (s2), ...(s28), (d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE_E</td>
<td>(s1), (s2), (d)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Standard character string functions

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID(_E)</td>
<td>(s), n1, n2, (d)</td>
<td>Outputs the specified number of characters, extracted from the specified start position in the input character string.</td>
<td>Page 155 Extract mid string</td>
</tr>
<tr>
<td>CONCAT(_E)</td>
<td>(s1), (s2), ...(s28), (d)</td>
<td>Concatenates the character strings and outputs the operation result.</td>
<td>Page 157 String concatenation</td>
</tr>
<tr>
<td>INSERT(_E)</td>
<td>(s1), (s2), n, (d)</td>
<td>Inserts a character string between other character strings and outputs the operation result.</td>
<td>Page 159 String insertion</td>
</tr>
<tr>
<td>DELETE(_E)</td>
<td>(s), n1, n2, (d)</td>
<td>Deletes the specified range in a character string and outputs the operation result.</td>
<td>Page 161 String deletion</td>
</tr>
<tr>
<td>REPLACE(_E)</td>
<td>(s1), (s2), n1, n2, (d)</td>
<td>Replaces the specified range in a character string with the specified character string and outputs the operation result.</td>
<td>Page 163 String replacement</td>
</tr>
</tbody>
</table>

### Functions of time data types

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_TIME(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the sum ((s1)+(s2)) of the input values (time type).</td>
<td>Page 166 Addition</td>
</tr>
<tr>
<td>SUB_TIME(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the difference ((s1)-(s2)) of input values (time type).</td>
<td>Page 168 Subtraction</td>
</tr>
<tr>
<td>MUL_TIME(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the product ((s1)\times(s2)) of input values (time type).</td>
<td>Page 170 Multiplication</td>
</tr>
<tr>
<td>DIV_TIME(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Outputs the quotient ((s1) \div (s2)) of input values (time type).</td>
<td>Page 172 Division</td>
</tr>
</tbody>
</table>

### Standard bistable function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Discriminates two input values and outputs 1 (TRUE) or 0 (FALSE). (Set-dominant)</td>
<td>Page 174 Standard bistable function blocks (Set-dominant)</td>
</tr>
<tr>
<td>RS(_E)</td>
<td>(s1), (s2), (d)</td>
<td>Discriminates two input values and outputs 1 (TRUE) or 0 (FALSE). (Reset-dominant)</td>
<td>Page 176 Standard bistable function blocks (Reset-dominant)</td>
</tr>
</tbody>
</table>
## Standard edge detection function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TRIG(_E)</td>
<td>(s), (d)</td>
<td>Detects the rising edge of a signal and outputs pulse signals.</td>
<td>Page 178 Rising edge detector</td>
</tr>
<tr>
<td>F_TRIG(_E)</td>
<td>(s), (d)</td>
<td>Detects the falling edge of a signal and outputs pulse signals.</td>
<td>Page 180 Falling edge detector</td>
</tr>
</tbody>
</table>

## Standard counter function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTU(_E)</td>
<td>(s1), (s2), n, (d1), (d2)</td>
<td>Counts the number of times that the signal turns ON.</td>
<td>Page 182 Up counter</td>
</tr>
<tr>
<td>CTD(_E)</td>
<td>(s1), (s2), n, (d1), (d2)</td>
<td>Counts down the number of times that the signal turns ON.</td>
<td>Page 184 Down counter</td>
</tr>
<tr>
<td>CTUD(_E)</td>
<td>(s1), (s2), (s3), (s4), n, (d1), (d2), (d3)</td>
<td>Counts/counts down the number of times that the signal turns ON.</td>
<td>Page 186 Up/Down counter</td>
</tr>
<tr>
<td>COUNTER_FB_M</td>
<td>(s1), (s2), (s3), (d1), (d2)</td>
<td>Counts the number of times that the signal turns ON from (s3) to (s2).</td>
<td>Page 190 Counter function blocks</td>
</tr>
</tbody>
</table>

## Standard timer function blocks

<table>
<thead>
<tr>
<th>Function name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP(_E)</td>
<td>(s), n, (d1), (d2)</td>
<td>Holds the signal ON for the specified time.</td>
<td>Page 192 Pulse timer</td>
</tr>
<tr>
<td>TP_HIGH(_E)</td>
<td>(s), n, (d1), (d2)</td>
<td>Holds the signal ON for the specified time.</td>
<td>Page 192 Pulse timer</td>
</tr>
<tr>
<td>TON(_E)</td>
<td>(s), n, (d1), (d2)</td>
<td>Turns ON the signal after the specified time.</td>
<td>Page 195 On delay timer</td>
</tr>
<tr>
<td>TON_HIGH(_E)</td>
<td>(s), n, (d1), (d2)</td>
<td>Turns ON the signal after the specified time.</td>
<td>Page 195 On delay timer</td>
</tr>
<tr>
<td>TOF(_E)</td>
<td>(s), n, (d1), (d2)</td>
<td>Turns OFF the signal after the specified time.</td>
<td>Page 198 Off delay timer</td>
</tr>
<tr>
<td>TOF_HIGH(_E)</td>
<td>(s), n, (d1), (d2)</td>
<td>Turns OFF the signal after the specified time.</td>
<td>Page 198 Off delay timer</td>
</tr>
<tr>
<td>TIMER_10_FB_M</td>
<td>(s1), (s2), (s3), (d1), (d2)</td>
<td>Turns ON the signal after the specified time counted from input value (s3) to (s2).</td>
<td>Page 201 Timer function blocks</td>
</tr>
<tr>
<td>TIMER_100_FB_M</td>
<td>(s1), (s2), (s3), (d1), (d2)</td>
<td>Turns ON the signal after the specified time counted from input value (s3) to (s2).</td>
<td>Page 201 Timer function blocks</td>
</tr>
<tr>
<td>TIMER_HIGH_FB_M</td>
<td>(s1), (s2), (s3), (d1), (d2)</td>
<td>Turns ON the signal after the specified time counted from input value (s3) to (s2).</td>
<td>Page 201 Timer function blocks</td>
</tr>
<tr>
<td>TIMER_LOW_FB_M</td>
<td>(s1), (s2), (s3), (d1), (d2)</td>
<td>Turns ON the signal after the specified time counted from input value (s3) to (s2).</td>
<td>Page 201 Timer function blocks</td>
</tr>
<tr>
<td>TIMER_CONT_FB_M</td>
<td>(s1), (s2), (s3), (d1), (d2)</td>
<td>Turns ON the signal after the specified time counted from input value (s3) to (s2).</td>
<td>Page 201 Timer function blocks</td>
</tr>
<tr>
<td>TIMER_CONT_HFB_M</td>
<td>(s1), (s2), (s3), (d1), (d2)</td>
<td>Turns ON the signal after the specified time counted from input value (s3) to (s2).</td>
<td>Page 201 Timer function blocks</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

### Arithmetic operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>+ (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the sum ((s1)+(s2)+...+(s28)) of input values.</td>
<td>Page 205 Addition</td>
</tr>
<tr>
<td>MUL</td>
<td>* (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the product ((s1) × (s2) × ... × (s28)) of input values.</td>
<td>Page 207 Multiplication</td>
</tr>
<tr>
<td>SUB</td>
<td>- (s1), (s2), (d)</td>
<td>Outputs the difference ((s1)-(s2)) between input values.</td>
<td>Page 208 Subtraction</td>
</tr>
<tr>
<td>DIV</td>
<td>/ (s1), (s2), (d)</td>
<td>Outputs the quotient ((s1) ÷ (s2)) of input values.</td>
<td>Page 209 Division</td>
</tr>
<tr>
<td>MOD</td>
<td>MOD (s1), (s2), (d)</td>
<td>Outputs the remainder after division of input values ((s1) ÷ (s2)).</td>
<td>Page 211 Remainder</td>
</tr>
<tr>
<td>**</td>
<td>** (s1), (s2), (d)</td>
<td>Outputs the exponentiation of an input value.</td>
<td>Page 212 Exponentiation</td>
</tr>
</tbody>
</table>

### Logical operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp; AND (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the Boolean AND of input values.</td>
<td>Page 213 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT</td>
</tr>
<tr>
<td>OR</td>
<td>OR (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the Boolean OR of input values.</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>XOR (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the Boolean exclusive OR of input values.</td>
<td></td>
</tr>
<tr>
<td>(Not supported)</td>
<td>NOT (s), (d)</td>
<td>Outputs the Boolean NOT of input values.</td>
<td></td>
</tr>
</tbody>
</table>

### Comparison operations

<table>
<thead>
<tr>
<th>Operator name</th>
<th>Argument</th>
<th>Processing details</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>&gt; (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td>Outputs the comparison value of an input value.</td>
<td>Page 215 Comparison</td>
</tr>
<tr>
<td>GE</td>
<td>&gt;= (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQ</td>
<td>= (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LE</td>
<td>&lt;= (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT</td>
<td>&lt; (s1), (s2), ...(s28), (d) (Number of pins variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>&lt;&gt; (s1), (s2), (d)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3 CONFIGURATION OF FUNCTIONS

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.

Source (s)
A source is data used in an operation.
The following source types are available depending on the device specified in a function:

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>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.</td>
</tr>
<tr>
<td>Bit device and word device</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

Destination (d)
Data after the operation are stored to a destination.
Set a device in which data are to be stored to a destination.
Coils cannot be directly connected to destinations that store bit devices.

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

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

For details of the configuration of functions for labels and structures, refer to the following.

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 details, refer to the following.
GX Works2 Version 1 Operating Manual (Structured Project)
Chapter 5 and after provides detailed explanation on each function in the layout as shown below.

1. **Indicates an outline of a function.**
2. **Indicates a function to be explained.**

### Operation processing

Converts word (signed) type data input to (y) to double word (signed) type data, and outputs the operation result from (z).

![Diagram of operation processing](image)

### Operation error

- No operation error occurs.

### Program example

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

```plaintext
1._function:
   IN (y)
   INT_TO_DINT_E
   OUT (z)

2._function:
   IN (y)
   INT_TO_DINT_E
   OUT (z)
```

*Indicates an outline of a function.*

*Indicates a function to be explained.*
Indicates the CPU modules that can use the function.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Basic model</th>
<th>High Performance model</th>
<th>Process CPU</th>
<th>Redundant CPU</th>
<th>Universal model</th>
<th>LCPU</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Basic</td>
<td>High performance</td>
<td>Process</td>
<td>Redundant</td>
<td>Universal</td>
<td>LCPU</td>
<td>The basic icon indicates that the CPU module can use the corresponding function.</td>
</tr>
<tr>
<td>✗</td>
<td>✗ Basic</td>
<td>✗ High performance</td>
<td>✗ Process</td>
<td>✗ Redundant</td>
<td>✗ Universal</td>
<td>✗ LCPU</td>
<td>The icon with ✗ indicates that the CPU module cannot use the corresponding function.</td>
</tr>
</tbody>
</table>

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

Indicates the names of input and output arguments, and the data type of each argument. For details of each data type, refer to the following.

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

Indicates the processing performed by the function.

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

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

5.1 Type Conversion Functions

Converting bit type to word (signed), double word (signed) type

BOOL_TO_INT(_E), BOOL_TO_DINT(_E)

The following function(s) can go in the dotted squares.
BOOL_TO_INT, BOOL_TO_INT_E, BOOL_TO_DINT, BOOL_TO_DINT_E

- Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_BOOL)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
</tr>
</tbody>
</table>

- Operation processing

  - BOOL_TO_INT, BOOL_TO_INT_E
    Converts bit type data input to (s) into word (signed) type data, and outputs the operation result from (d).
    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.

    ![Diagram](image)

  - BOOL_TO_DINT, BOOL_TO_DINT_E
    Converts bit type data input to (s) into double word (signed) type data, and outputs the operation result from (d).
    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.

    ![Diagram](image)
# Operation result

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

- **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 (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 (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.

## Program example

### BOOL_TO_INT(_E)

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

- **Function without EN/ENO (BOOL_TO_INT)**

  [Structured ladder/FBD]

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

- **Function with EN/ENO (BOOL_TO_INT_E)**

  [Structured ladder/FBD]

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

### BOOL_TO_DINT(_E)

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

- **Function without EN/ENO (BOOL_TO_DINT)**

  [Structured ladder/FBD]

  ```plaintext
  g_dint1:=BOOL_TO_DINT(g_bool1);
  ```
Converting bit type to string type

### BOOL_TO_STR(E)

The following function(s) can go in the dotted squares.

**BOOL_TO_STR, BOOL_TO_STR_E**

#### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_BOOL)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String</td>
</tr>
</tbody>
</table>

#### Processing details

**Operation processing**

Converts bit type data input to (s) into string type data, and outputs the operation result from (d).

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.

#### Operation result

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

- 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 (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 (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.

Program example

**BOOL_TO_STR(E)**
The program which converts bit type data input to (s) into string type data, and outputs the operation result from (d).

• Function without EN/ENO (BOOL_TO_STR)

[Structured ladder/FBD]

```
g_string1 := BOOL_TO_STR(g_bool1);
```
Converting bit type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type

**BOOL_TO_WORD(_E), BOOL_TO_DWORD(_E)**

Structured ladder/FBD

```
BOOL_TO_WORD_E
  EN  ENO
  s   d
```

ST

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

The following function(s) can go in the dotted squares.

BOOL_TO_WORD, BOOL_TO_WORD_E, BOOL_TO_DWORD, BOOL_TO_DWORD_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_BOOL)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (unsigned)/16-bit string, double word (unsigned)/32-bit string</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- **BOOL_TO_WORD, BOOL_TO_WORD_E**
  - Converts bit type data input to (s) into word (unsigned)/16-bit string type data, and outputs the operation result from (d).
  - 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.

  ![Diagram](false_true_word_conversion)

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

  ![Diagram](false_true_dword_conversion)
■ Operation result

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

Program example

■ BOOL_TO_WORD(_E)
The program which converts bit type data input to (s) into word (unsigned)/16-bit string type data, and outputs the operation result from (d).

- Function without EN/ENO (BOOL_TO_WORD)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

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

- Function with EN/ENO (BOOL_TO_WORD_E)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

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

■ BOOL_TO_DWORD(_E)
The program which converts bit type data input to (s) into double word (unsigned)/32-bit string type data, and outputs the operation result from (d).

- Function without EN/ENO (BOOL_TO_DWORD)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

  ```
  g_dword1 := BOOL_TO_DWORD(g_bool1);
  ```
Converting bit type to time type

**BOOL_TO_TIME(_E)**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TO_TIME (_E)</td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s(_BOOL)</td>
<td>d</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.

**BOOL_TO_TIME, BOOL_TO_TIME_E**

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_BOOL)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Time</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

Converts bit type data input to (s) into time type data, and outputs the operation result from (d).

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.

<table>
<thead>
<tr>
<th>Bit type</th>
<th>Time type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>0</td>
</tr>
<tr>
<td>TRUE</td>
<td>T#1ms</td>
</tr>
</tbody>
</table>

#### Operation result

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

- **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 (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 (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.

Program example

■ BOOL_TO_TIME(_E)
The program which converts bit type data input to (s) into time type data, and outputs the operation result from (d).

• Function without EN/ENO (BOOL_TO_TIME)

[Structured ladder/FBD]

```plaintext
1

g_time1 := BOOL_TO_TIME(g_bool1);
```

[ST]

```plaintext
2

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

The program which converts bit type data input to (s) into time type data, and outputs the operation result from (d).

• Function without EN/ENO (BOOL_TO_TIME)
Converting word (signed) type to double word (signed) type

**INT_TO_DINT(_E)**

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT)</td>
<td>Input</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Double word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

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

**Operation result**

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

- 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 (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 (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.

Program example

**INT_TO_DINT(E)**

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

• Function without EN/ENO (INT_TO_DINT)

  [Structured ladder/FBD]

  1
  
  g_int1 := INT_TO_DINT(g_int1);

  [ST]
  g_bool3 := INT_TO_DINT_E(g_bool1, g_int1, g_dint1);

• Function with EN/ENO (INT_TO_DINT_E)

  [Structured ladder/FBD]

  2
  
  g_bool1 := INT_TO_DINT_E(g_bool1, g_int1, g_dint1);
Converting double word (signed) type to word (signed) type

**DINT_TO_INT(_E)**

The following function(s) can go in the dotted squares.

DINT_TO_INT, DINT_TO_INT_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_DINT)</td>
<td>Input</td>
<td>Double word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed)</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

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

![Diagram](#)

#### Operation result

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

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) 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 (s) are converted into word (signed) type data. High-order 16-bit data are discarded.
Operation error

• No operation error occurs.

Program example

**DINT_TO_INT(_E)**
The program which converts double word (signed) type data input to (s) into word (signed) type data, and outputs the operation result from (d).

• Function without EN/ENO (DINT_TO_INT)

[Structured ladder/FBD]

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

• Function with EN/ENO (DINT_TO_INT_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := DINT_TO_INT_E(g_bool1, g_dint1, g_int1);
```
Converting word (signed), double word (signed) type to bit type

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

The following function(s) can go in the dotted squares.
INT_TO_BOOL, INT_TO_BOOL_E, DINT_TO_BOOL, DINT_TO_BOOL_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT, _DINT)</td>
<td>Input (Word (signed), double word (signed))</td>
<td>Word (signed), double word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- **INT_TO_BOOL, INT_TO_BOOL_E**
  Converts word (signed) type data input to (s) into bit type data, and outputs the operation result from (d).
  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.

  ![Diagram of INT_TO_BOOL_E](image)

- **DINT_TO_BOOL, DINT_TO_BOOL_E**
  Converts double word (signed) type data input to (s) into bit type data, and outputs the operation result from (d).
  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.

  ![Diagram of DINT_TO_BOOL_E](image)
## Operation result

- Function without EN/ENO

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

- 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 (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 (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.

## Program example

### INT_TO_BOOL(_E)

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

- Function without EN/ENO (INT_TO_BOOL)

[Structured ladder/FBD]

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

- Function with EN/ENO (INT_TO_BOOL_E)

[Structured ladder/FBD]

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

### DINT_TO_BOOL(_E)

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

- Function without EN/ENO (DINT_TO_BOOL)

[Structured ladder/FBD]

```
[ST]
g_bool1 := DINT_TO_BOOL(g_dint1);
```
Converting word (signed), double word (signed) type to single-precision real type

INT_TO_REAL(_E), DINT_TO_REAL(_E)

The following function(s) can go in the dotted squares.
INT_TO_REAL, INT_TO_REAL_E, DINT_TO_REAL, DINT_TO_REAL_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT, _DINT)</td>
<td>Input</td>
<td>Word (signed), double word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Single-precision real number</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

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

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

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.
5 APPLICATION FUNCTIONS
5.1 Type Conversion Functions

Operation result

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

Program example

- **INT_TO_REAL(_E)**
  The program which converts word (signed) type data input to (s) into single-precision real type data, and outputs the operation result from (d).
  - Function without EN/ENO (INT_TO_REAL)
    [Structured ladder/FBD]
    
    ![Structured ladder/FBD](image)
    
    g_real1 := INT_TO_REAL(g_int1);
  - Function with EN/ENO (INT_TO_REAL_E)
    [Structured ladder/FBD]
    
    ![Structured ladder/FBD](image)
    
    g_bool3 := INT_TO_REAL_E(g_bool1, g_int1, g_real1);

- **DINT_TO_REAL(_E)**
  The program which converts double word (signed) type data input to (s) into single-precision real type data, and outputs the operation result from (d).
  - Function without EN/ENO (DINT_TO_REAL)
    [Structured ladder/FBD]
    
    ![Structured ladder/FBD](image)
    
    g_real1 := DINT_TO_REAL(g_dint1);
  - Function with EN/ENO (DINT_TO_REAL_E)
    [Structured ladder/FBD]
    
    ![Structured ladder/FBD](image)
    
    g_bool3 := DINT_TO_REAL_E(g_bool1, g_dint1, g_real1);
Converting word (signed), double word (signed) type to double-precision real type

**INT_TO_LREAL(_E), DINT_TO_LREAL(_E)**

The following function(s) can go in the dotted squares.
INT_TO_LREAL, INT_TO_LREAL_E, DINT_TO_LREAL, DINT_TO_LREAL_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT, _DINT)</td>
<td>Input</td>
<td>Word (signed), double word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Double-precision real</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- **INT_TO_LREAL, INT_TO_LREAL_E**
  
  Converts word (signed) type data input to (s) into double-precision real type data, and outputs the operation result from (d).

  ![Diagram](word-to-double precision)

- **DINT_TO_LREAL, DINT_TO_LREAL_E**
  
  Converts double word (signed) type data input to (s) into double-precision real type data, and outputs the operation result from (d).

  ![Diagram](double word-to-double precision)
5 APPLICATION FUNCTIONS

5.1 Type Conversion Functions

Operation result

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

Program example

**INT_TO_LREAL(_E)**
The program which converts word (signed) type data input to (s) into double-precision real type data, and outputs the operation result from (d).

- Function without EN/ENO (INT_TO_LREAL)
  [Structured ladder/FBD]
  
  ```plaintext
  g_lreal1 := INT_TO_LREAL(g_int1);
  ```

- Function with EN/ENO
  [Structured ladder/FBD]
  
  ```plaintext
  g_bool3 := INT_TO_LREAL_E(g_bool1, g_int1, g_lreal2);
  ```

**DINT_TO_LREAL(_E)**
The program which converts double word (signed) type data input to (s) into double-precision real type data, and outputs the operation result from (d).

- Function without EN/ENO (DINT_TO_LREAL)
  [Structured ladder/FBD]
  
  ```plaintext
  g_lreal1 := DINT_TO_LREAL(g_dint1);
  ```
Converting word (signed), double word (signed) type to string type

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

The following function(s) can go in the dotted squares.
INT_TO_STR, INT_TO_STR_E, DINT_TO_STR, DINT_TO_STR_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT, _DINT)</td>
<td>Input</td>
<td>Word (signed), double word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String(6)/(11)</td>
</tr>
</tbody>
</table>

**Processing details**

### Operation processing

- **INT_TO_STR, INT_TO_STR_E**

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

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

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

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

**Ex**

**Inputting -123**

When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string.
• DINT_TO_STR, DINT_TO_STR_E
Converting double 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 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 hundred-thousands 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>
</tbody>
</table>

Doubled word (signed) type

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

'20H (space)' is stored in 'Sign data' when the input value is positive; '2DH (-)' is stored when negative.
If the number of significant figures is less, '20H (space)' is stored to high-order digits.

Ex.
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>2DH (-)</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>
</tbody>
</table>

Double word (signed) type

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

**Operation result**
• Function without EN/ENO
An operation is executed and the operation value is output from (d).
• 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 (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 (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.

Program example

**INT_TO_STR(_E)**
The program which converts word (signed) type data input to (s) into string type data, and outputs the operation result from (d).

• Function without EN/ENO (INT_TO_STR)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram for INT_TO_STR](image1)

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

• Function with EN/ENO (INT_TO_STR_E)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram for INT_TO_STR_E](image2)

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

**DINT_TO_STR(_E)**
The program which converts double word (signed) type data input to (s) into string type data, and outputs the operation result from (d).

• Function without EN/ENO (DINT_TO_STR)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram for DINT_TO_STR](image3)

  ```
  g_string1 := DINT_TO_STR(g_dint1);
  ```
Converting word (signed), double word (signed) type to word (unsigned)/16-bit string type

**INT_TO_WORD(_E), DINT_TO_WORD(_E)**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s(_INT, _DINT)</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.
INT_TO_WORD, INT_TO_WORD_E, DINT_TO_WORD, DINT_TO_WORD_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT, _DINT)</td>
<td>Input</td>
<td>Word (signed), double word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (unsigned)/16-bit string</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- **INT_TO_WORD, INT_TO_WORD_E**
Converting word (signed) type data input to (s) into word (unsigned)/16-bit string type data, and outputs the operation result from (d).

  22136 → 5678H
  
  - Word (signed) type
  - Word (unsigned)/16-bit string type

- **DINT_TO_WORD, DINT_TO_WORD_E**
Converting double word (signed) type data input to (s) into word (unsigned)/16-bit string type data, and outputs the operation result from (d).

  12345678 → 614EH
  
  - Double word (signed) type
  - Word (unsigned)/16-bit string type
  
12345678

  0101100010111000111100010011101011001000

614EH

  0110001010011100111001111

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

- **Function without EN/ENO**
  An operation is executed and the operation value is output from (d).
- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) 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 (s) are converted into word (unsigned)/16-bit string type data. High-order word (unsigned)/16-bit string type data are discarded.

### Operation error

- No operation error occurs.

### Program example

#### INT_TO_WORD(_E)

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

- **Function without EN/ENO (INT_TO_WORD)**
  [Structured ladder/FBD]

```plaintext
1
  g_int1 := INT_TO_WORD(g_int1);
```

- **Function with EN/ENO (INT_TO_WORD_E)**
  [Structured ladder/FBD]

```plaintext
2
  g_bool3 := INT_TO_WORD_E(g_bool1, g_int1, g_word1);
```

#### DINT_TO_WORD(_E)

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

- **Function without EN/ENO (DINT_TO_WORD)**
  [Structured ladder/FBD]

```plaintext
3
  g_dint1 := DINT_TO_WORD(g_dint1);
```
Converting word (signed), double word (signed) type to double word (unsigned)/32-bit string type

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

<table>
<thead>
<tr>
<th>Basic</th>
<th>High performance</th>
<th>Redundant</th>
<th>Universal</th>
<th>LCPU</th>
</tr>
</thead>
</table>

**Structured ladder/FBD**

```
INT_TO_DWORD_E:

EN: Executing condition (TRUE: Execution, FALSE: Stop)
  s(_INT, _DINT): Input Word (signed), double word (signed)

ENO: Execution result (TRUE: Normal, FALSE: Error)
  d: Output Double word (unsigned)/32-bit string
```

The following function(s) can go in the dotted squares.
INT_TO_DWORD, INT_TO_DWORD_E, DINT_TO_DWORD, DINT_TO_DWORD_E

■Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT, _DINT)</td>
<td>Input Word (signed), double word (signed)</td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output Double word (unsigned)/32-bit string</td>
<td></td>
</tr>
</tbody>
</table>

■Processing details

■Operation processing

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

```
{-325} \rightarrow {0000FEBBh}
```

Word (signed) type

```
1111111010111011
```

Double word (unsigned)/32-bit string type

```
0000FEBBh
```

Always filled with 0s.

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

```
{12345678} \rightarrow {BC614EH}
```

Double word (signed) type

```
1111111010111011
```

Double word (unsigned)/32-bit string type
**Operation result**

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

**Program example**

- **INT_TO_DWORD(_E)**
  The program which converts word (signed) type data input to (s) into double word (unsigned)/32-bit string type data, and outputs the operation result from (d).
  - **Function without EN/ENO (INT_TO_WORD)**
    [Structured ladder/FBD]
    ```plaintext
    g_dword1 := INT_TO_DWORD(g_int1);
    ```
  - **Function with EN/ENO (INT_TO_DWORD_E)**
    [Structured ladder/FBD]
    ```plaintext
    g_bool3 := INT_TO_DWORD_E(g_bool1, g_int1, g_dword1);
    ```

- **DINT_TO_DWORD(_E)**
  The program which converts double word (signed) type data input to (s) into double word (unsigned)/32-bit string type data, and outputs the operation result from (d).
  - **Function without EN/ENO (DINT_TO_DWORD)**
    [Structured ladder/FBD]
    ```plaintext
    g_dword1 := DINT_TO_DWORD(g_dint1);
    ```
Converting word (signed), double word (signed) type to BCD type

**INT_TO_BCD(_E), DINT_TO_BCD(_E)**

The following function(s) can go in the dotted squares.
INT_TO_BCD, INT_TO_BCD_E, DINT_TO_BCD, DINT_TO_BCD_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_INT, _DINT)</td>
<td>Input</td>
<td>Word (signed), double word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (unsigned)/16-bit string, double word (unsigned)/32-bit string</td>
</tr>
</tbody>
</table>

### Processing details

**Operation processing**

- **INT_TO_BCD, INT_TO_BCD_E**

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

The value to be input to (s) is word (signed) type data within the range from 0 to 9999.
• DINT_TO_BCD, DINT_TO_BCD_E

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

The value to be input to (s) is double word (signed) type data within the range from 0 to 99999999. Word (unsigned)/16-bit string type, double word (unsigned)/32-bit string type data can be specified for (s). Bit type cannot be specified.

The output from (d) 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

#### Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

#### 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.
Operation error

• An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/Q00/Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>When the value input exceeds 9999 in the execution of the INT_TO_BCD(_E) When the value input exceeds 99999999 in the execution of the DINT_TO_BCD(_E)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Program example

**INT_TO_BCD(_E)**
The program which converts word (signed) type data input to (s) into BCD type data, and outputs the operation result from (d).

• Function without EN/ENO (INT_TO_BCD)

[Structured ladder/FBD]

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

• Function with EN/ENO (INT_TO_BCD_E)

[Structured ladder/FBD]

```
[ST]  g Bool3 := INT_TO_BCD_E(g_ BOOL1, g_int1, g_word1);
```

**DINT_TO_BCD(_E)**
The program which converts double word (signed) type data input to (s) into BCD type data, and outputs the operation result from (d).

• Function without EN/ENO (DINT_TO_BCD)

[Structured ladder/FBD]

```
[ST]  g_dword1 := DINT_TO_BCD(g_dint1);
```
Converting word (signed), double word (signed) type to time type

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

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

The following function(s) can go in the dotted squares.
INT_TO_TIME, INT_TO_TIME_E, DINT_TO_TIME, DINT_TO_TIME_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(_INT, _DINT)</td>
<td>Input</td>
<td>Word (signed), double word (signed)</td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d Output</td>
<td>Time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

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

![Diagram of operation processing]

**Operation result**

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

- 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 (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 (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.

Program example

**INT_TO_TIME(_E)**

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

- Function without EN/ENO (INT_TO_TIME)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram](image)

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

- Function with EN/ENO (INT_TO_TIME_E)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram](image)

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

**DINT_TO_TIME(_E)**

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

- Function without EN/ENO (DINT_TO_TIME)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram](image)

  ```
  g_time1 := DINT_TO_TIME(g_dint1);
  ```
Converting single-precision real type to word (signed), double word (signed) type

REAL_TO_INT(_E), REAL_TO_DINT(_E)

The following function(s) can go in the dotted squares.
REAL_TO_INT, REAL_TO_INT_E, REAL_TO_DINT, REAL_TO_DINT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_REAL)</td>
<td>Input</td>
<td>Single-precision real number</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- **REAL_TO_INT, REAL_TO_INT_E**
  Converting single-precision real type data input to (s) into word (signed) type data, and outputs the operation result from (d).

  ![Conversion Example](image)

  The value to be input to (s) is single-precision real type data, within the range from -32768 to 32767.
  The converted data is the value rounded single-precision real type data to the first digit after the decimal point.

- **REAL_TO_DINT, REAL_TO_DINT_E**
  Converting single-precision real type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

  ![Conversion Example](image)

  The value to be input to (s) is single-precision real type data within the range from -2147483648 to 2147483647.
  However, a rounding error may occur when setting the input value by programming tool. For precautions when setting an input value using a programming tool, refer to the following.

  - [MELSEC-Q/L/F Structured Programming Manual (Fundamentals)]
  - The converted data is the value rounded single-precision real type data to the first digit after the decimal point.
### Operation result
- Function without EN/ENO
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

### Operation error
An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>When the REAL_TO_INT(_E) instruction is used, the input value is outside</td>
</tr>
<tr>
<td></td>
<td>the range of -32768 to 32767. When the REAL_TO_DINT(_E) instruction is used,</td>
</tr>
<tr>
<td></td>
<td>the input value is outside the range of -2147483648 to 2147483647.</td>
</tr>
</tbody>
</table>

### Program example
- REAL_TO_INT(_E)
The program which converts single-precision real type data input to (s) into word (signed) type data, and outputs the operation result from (d).

- Function without EN/ENO (REAL_TO_INT)
[Structured ladder/FBD]

```
[Structured ladder/FBD]
g_int1 := REAL_TO_INT(g_real1);
```

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

```
[Structured ladder/FBD]
g_bool3 := REAL_TO_INT_E(g_bool1, g_real1, g_int1);
```

- REAL_TO_DINT(_E)
The program which converts single-precision real type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

- Function without EN/ENO (REAL_TO_DINT)
[Structured ladder/FBD]

```
[Structured ladder/FBD]
g_dint1 := REAL_TO_DINT(g_real1);
```
Converting double-precision real type to word (signed), double word (signed) type

LREAL_TO_INT(_E), LREAL_TO_DINT(_E)

The following function(s) can go in the dotted squares.
LREAL_TO_INT, LREAL_TO_INT_E, LREAL_TO_DINT, LREAL_TO_DINT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_LREAL)</td>
<td>Input</td>
<td>Double-precision real</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

• LREAL_TO_INT, LREAL_TO_INT_E

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

![Diagram](image)

The value to be input to (s) is double-precision real type data, within the range from -32768 to 32767.
The converted data is the value rounded double-precision real type data to the first digit after the decimal point.

• LREAL_TO_DINT, LREAL_TO_DINT_E

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

![Diagram](image)

The value to be input to (s) 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 precautions when setting an input value using a programming tool, refer to the following.

MELSEC-Q/L/F Structured Programming Manual (Fundamentals)
The converted data is the value rounded double-precision real type data to the first digit after the decimal point.
**Operation result**

- Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

**Operation error**

An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/Q00 Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4140</td>
<td>The input value is -0 or outside the following range.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0, $2^{1022} &lt;</td>
<td>(s)</td>
<td>&lt; $2^{1024}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the LREAL_TO_INT(E) instruction is used, the input value is outside</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the range of -32768.0 to 32767.0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the LREAL_TO_DINT(E) instruction is used, the input value is</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>outside the range of -2147483648.0 to 2147483647.0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Program example**

**LREAL_TO_INT(E)**

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

- Function without EN/ENO (LREAL_TO_INT)

[Structured ladder/FBD]

```plaintext
LREAL_TO_INT(g_lreal1)
```

- Function with EN/ENO (LREAL_TO_INT_E)

[Structured ladder/FBD]

```plaintext
LREAL_TO_INT_E(g_lreal1, g_int1)
```

**LREAL_TO_DINT(E)**

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

- Function without EN/ENO (LREAL_TO_DINT)

[Structured ladder/FBD]

```plaintext
LREAL_TO_DINT(g_lreal1)
```

- Function with EN/ENO (LREAL_TO_DINT_E)

[Structured ladder/FBD]

```plaintext
LREAL_TO_DINT_E(g_lreal1, g_dint1)
```
Converting single-precision real type to double-precision real type

**REAL_TO_LREAL(_E)**

- **Argument**
  - Input argument:
    - EN: Executing condition (TRUE: Execution, FALSE: Stop) Bit
    - s(_REAL): Input Single-precision real number
  - Output argument:
    - ENO: Execution result (TRUE: Normal, FALSE: Error) Bit
    - d: Output Double-precision real

- **Processing details**

- **Operation processing**
  - Converts single-precision real type data input to (s) into double-precision real type data, and outputs the operation result from (d).

  ![Diagram showing conversion from single-precision real type to double-precision real type](image)

  - Rounding error may occur when specifying the input value to (s) by programming tool. For precautions when setting an input value using a programming tool, refer to the following.

  - [MELSEC-Q/L/F Structured Programming Manual (Fundamentals)](https://example.com/manual)
Operation result

- Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- 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

- An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4140</td>
<td>The input value is -0 or outside the following range. 0, 2 (10^{26} \leq \mid (s) \leq 2^{128} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4141</td>
<td>The operation result exceeds the following range. (An overflow occurs.) ( 2^{1024} \leq \mid \text{operation result} \mid )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Program example

**REAL_TO_LREAL(_E)**

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

- Function without EN/ENO (REAL_TO_LREAL)

[Structured ladder/FBD]

```
1
-- REAL_TO_LREAL REAL

-- REAL_TO_LREAL REAL
```

[ST]

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

- Function with EN/ENO (REAL_TO_LREAL_E)

[Structured ladder/FBD]

```
2
-- REAL_TO_LREAL REAL

-- REAL_TO_LREAL REAL
```

[ST]

```
g_bool3 := REAL_TO_LREAL_E(g_bool1, g_real1, g_lreal1);
```
Converting double-precision real type to single-precision real type

LREAL_TO_REAL(_E)

The following function(s) can go in the dotted squares.
LREAL_TO_REAL, LREAL_TO_REAL_E

 Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_LREAL)</td>
<td>Input</td>
<td>Double-precision real</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Single-precision real number</td>
</tr>
</tbody>
</table>

 Processing details

 Operation processing
• Converts double-precision real type data input to (s) into single-precision real type data, and outputs the operation result from (d).

• Rounding error may occur when setting the input value to (s) by programming tool. For precautions when setting an input value using a programming tool, refer to the following.

☐ MELSEC-Q/L/F Structured Programming Manual (Fundamentals)
### Operation result

- **Function without EN/ENO**

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- **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 (Operation execution)</td>
<td>TRUE (No operation error)</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

<sup>1</sup> 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

- An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00/J/ Q00/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
</table>
| 4140 | The input value is <=0 or outside the following range. 
\(0, 2^{1022} \leq |(s)| < 2^{1024}\) | — | — | — | — | ○ | ○ |
| 4141 | The operation result exceeds the following range. (An overflow occurs.) 
\(2^{128} \leq |\text{operation result}|\) | — | — | — | — | ○ | ○ |

### Program example

**LREAL_TO_REAL(_E)**

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

- **Function without EN/ENO (LREAL_TO_REAL)**

  [Structured ladder/FBD]

  ```plaintext
  g_real1 := LREAL_TO_REAL(g_lreal1);
  ```

  [ST]
  
  ```plaintext
  g_bool3 := LREAL_TO_REAL_E(g_bool1, g_lreal1, g_real1);
  ```

- **Function with EN/ENO (LREAL_TO_REAL_E)**

  [Structured ladder/FBD]

  ```plaintext
  ```
  [ST]
  
  ```plaintext
  ```
Converting single-precision real type to string type

**REAL_TO_STR(_E)**

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

The following function(s) can go in the dotted squares.
REAL_TO_STR, REAL_TO_STR_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_LREAL)</td>
<td>Input single-precision real number</td>
<td>Single-precision real number</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output string</td>
<td>String (13)</td>
</tr>
</tbody>
</table>

### Processing details

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

When SM701 (signal for switching the number of output character) is OFF, "00" is stored.
• The character string data after conversion is output from output variable (d) in the following manner.
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)
‘20H’ (space), ‘2EH’ (.) and ‘45H’ (E) are automatically stored in the 2nd, 4th and 10th bytes, respectively.

-20H (space), -2EH (.) and -45H (E) are automatically stored in the 2nd, 4th and 10th bytes, respectively. 

'20H' (space) is stored in 'Sign data' (integer part) when the input value is positive; ‘2DH’ (-) is stored when negative.
Fractional part is rounded to 5 decimal places.

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

'2BH' (+) is stored in the 'Sign data' (exponent part) if the exponent is positive; '2DH' (-) is stored when negative.
'30H' (0) is stored to tens place in the exponent part if exponent part has only one digit.

• When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string (7th word).
• Rounding error may occur when specifying the input value to (S) by programming tool. For precautions when setting an input value using a programming tool, refer to the following.

MELSEC-Q/L/F Structured Programming Manual (Fundamentals)
### Operation result

- **Function without EN/ENO**

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

### Operation error

- An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00/J/Q00/Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The input value is outside the range of -3.40282<em>38 to -1.17549</em>38, 0 or 1.17549<em>38 to 3.40282</em>38</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
</tr>
</tbody>
</table>

### Program example

#### REAL_TO_STR(_E)

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

- **Function without EN/ENO (REAL_TO_STR)**

  [Structured ladder/FBD]

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

- **Function with EN/ENO (REAL_TO_STR_E)**

  [Structured ladder/FBD]

  ```plaintext
  g_bool3 := REAL_TO_STR_E(g_bool1, g_real1, g_string1);
  ```
Converting word (unsigned)/16-bit string, double word (unsigned)/32-bit string type to bit type

WORD_TO_BOOL(_E), DWORD_TO_BOOL(_E)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_WORD, _DWORD)</td>
<td>Input</td>
<td>Word (unsigned)/16-bit string, Double word (unsigned)/32-bit string</td>
</tr>
<tr>
<td>Output</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.
WORD_TO_BOOL, WORD_TO_BOOL_E, DWORD_TO_BOOL, DWORD_TO_BOOL_E

**Argument**

**Processing details**

**Operation processing**
- **WORD_TO_BOOL, WORD_TO_BOOL_E**
Converting word (unsigned)/16-bit string type data input to (s) 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.

- **DWORD_TO_BOOL, DWORD_TO_BOOL_E**
Converting double word (unsigned)/32-bit string type data input to (s) 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.
### Operation result

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

- 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 (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 (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.

### Program example

**WORD_TO_BOOL(_E)**
The program which converts word (unsigned)/16-bit string type data input to (s) into bit type data, and outputs the operation result from (d).

- Function without EN/ENO (WORD_TO_BOOL)
  
  ![Structured ladder/FBD](image1)

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

- Function with EN/ENO (WORD_TO_BOOL_E)
  
  ![Structured ladder/FBD](image2)

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

**DWORD_TO_BOOL(_E)**
The program which converts double word (unsigned)/32-bit string type data input to (s) into bit type data, and outputs the operation result from (d).

- Function without EN/ENO (DWORD_TO_BOOL)
  
  ![Structured ladder/FBD](image3)

  ```
  g_bool1 := DWORD_TO_BOOL(g_dword1);
  ```
Converting word (unsigned)/16-bit string type to word (signed), double word (signed) type

**WORD_TO_INT(_E), WORD_TO_DINT(_E)**

The following function(s) can go in the dotted squares.

**WORD_TO_INT, WORD_TO_INT_E, WORD_TO_DINT, WORD_TO_DINT_E**

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(_WORD)</td>
<td>Input</td>
<td>Word (unsigned)/16-bit string</td>
<td>Word (unsigned)/16-bit string</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
<td></td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- **WORD_TO_INT, WORD_TO_INT_E**
  Converts word (unsigned)/16-bit string type data input to (s) into word (signed) type data, and outputs the operation result from (d).

  ![Data conversion](image)

- **WORD_TO_DINT, WORD_TO_DINT_E**
  Converts word (unsigned)/16-bit string type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

  ![Data conversion](image)

  Always filled with 0s.
**Operation result**

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

- 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 (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 (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.

**Program example**

**WORD_TO_INT(\_E)**

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

- Function without EN/ENO (WORD_TO_INT)
  
  [Structured ladder/FBD]

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

- Function with EN/ENO (WORD_TO_INT_E)
  
  [Structured ladder/FBD]

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

**WORD_TO_DINT(\_E)**

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

- Function without EN/ENO (WORD_TO_DINT)
  
  [Structured ladder/FBD]

  ```
  g_dint1 := WORD_TO_DINT(g_word1);
  ```
Converting double word (unsigned)/32-bit string type to word (signed), double word (signed) type

DWORD_TO_INT(_E), DWORD_TO_DINT(_E)

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD_TO_INT(_E)</td>
<td>EN ENO</td>
</tr>
<tr>
<td>s(_DWORD)</td>
<td>d</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.
DWORD_TO_INT, DWORD_TO_INT_E, DWORD_TO_DINT, DWORD_TO_DINT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(_DWORD)</td>
<td>Input</td>
<td>Double word (unsigned)/32-bit string</td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
<td></td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- 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).

  ![BC614EH to 24910](image)

  - BC614EH: Double word (unsigned)/32-bit string type
  - 24910: Word (signed) type
  - High-order 16-bit data is discarded.

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

  ![BC614EH to 12345678](image)

  - BC614EH: Double word (unsigned)/32-bit string type
  - 12345678: Double word (signed) type
Operation result

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

- 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

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 (s) are converted into word (signed) type data. High-order 16-bit data are discarded.

Operation error

- No operation error occurs.

Program example

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

- Function without EN/ENO (DWORD_TO_INT)
[Structured ladder/FBD]

```
[ST]
g_int1 := DWORD_TO_INT(g_dword1);
```

- Function with EN/ENO (DWORD_TO_INT_E)
[Structured ladder/FBD]

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

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

- Function without EN/ENO (DWORD_TO_DINT)
[Structured ladder/FBD]

```
[ST]
g_dint1 := DWORD_TO_DINT(g_dword1);
```
Converting word (unsigned)/16-bit string type to double word (unsigned)/32-bit string type

**WORD_TO_DWORD(_E)**

The following function(s) can go in the dotted squares.

**WORD_TO_DWORD, WORD_TO_DWORD_E**

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_WORD)</td>
<td>Input</td>
<td>Word (unsigned)/16-bit string</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Double word (unsigned)/32-bit string</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

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

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

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

**Operation result**

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

- 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 (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 (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.

Program example

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

• Function without EN/ENO (WORD_TO_DWORD)
  [Structured ladder/FBD]

```
ST
```

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

• Function with EN/ENO (WORD_TO_DWORD_E)
  [Structured ladder/FBD]

```
ST
```

```
g_bool3 := WORD_TO_DWORD_E(g_bool1, g_word1, g_dword1);
```
Converting double word (unsigned)/32-bit string type to word (unsigned)/16-bit string type

**DWORD_TO_WORD(_E)**

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

The following function(s) can go in the dotted squares.

DWORD_TO_WORD, DWORD_TO_WORD_E

**Argument**

<table>
<thead>
<tr>
<th>Input argument/Output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_WORD)</td>
<td>Input</td>
<td>Double word (unsigned)/32-bit string</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (unsigned)/16-bit string</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

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

![Conversion Diagram]

High-order 16-bit data is discarded.

**Operation result**

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

- 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.
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 (s) are converted into word (unsigned)/16-bit string type data. High-order 16-bit data are discarded.

**Operation error**

- No operation error occurs.

**Program example**

**DWORD_TO_WORD(_E)**

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

- Function without EN/ENO (DWORD_TO_WORD)

  
  [Structured ladder/FBD]

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

- Function with EN/ENO (DWORD_TO_WORD_E)

  
  [Structured ladder/FBD]

  ```plaintext
  g_bool3 := DWORD_TO_WORD_E(g_bool1, g_dword1, g_word1);
  ```
Converting word (unsigned)/16-bit string, double word (unsigned)/32-bit string type to string type

WORD_TO_STR(_E), DWORD_TO_STR(_E)

The following function(s) can go in the dotted squares.
WWORD_TO_STR, ORD_TO_STR_E, DWORD_TO_STR, DWORD_TO_STR_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(_WORD, _DWORD)</td>
<td>Input</td>
<td>Word (unsigned)/16-bit string type data input to (s) into string type data, and outputs the operation result from (d).</td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>String(4)/(8)</td>
<td>String(4)/(8)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- WORD_TO_STR, WORD_TO_STR_E
  Converts word (unsigned)/16-bit string type data input to (s) into string type data, and outputs the operation result from (d).

  ![Diagram of operation processing](image)

  Word (unsigned)/16-bit string type  
  String type

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

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

  ![Diagram of operation processing](image)

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

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

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

### Program example

#### WORD_TO_STR(_E)

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

- **Function without EN/ENO (WORD_TO_STR)**
  [Structured ladder/FBD]

  ```plaintext
  g_string1 := WORD_TO_STR(g_word1);
  ```

- **Function with EN/ENO (WORD_TO_STR_E)**
  [Structured ladder/FBD]

  ```plaintext
  g_bool3 := WORD_TO_STR_E(g_bool1, g_word1, g_string1);
  ```

#### DWORD_TO_STR(_E)

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

- **Function without EN/ENO (DWORD_TO_STR)**
  [Structured ladder/FBD]

  ```plaintext
  g_string1 := DWORD_TO_STR(g_dword1);
  ```
Converting word (unsigned)/16-bit string, double word (unsigned)/32-bit string type to time type

WORD_TO_TIME(_E), DWORD_TO_TIME(_E)

The following function(s) can go in the dotted squares.
WORD_TO_TIME, WORD_TO_TIME_E, DWORD_TO_TIME, DWORD_TO_TIME_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_WORD, _DWORD)</td>
<td>Input Word (unsigned)/16-bit string, Double word (unsigned)/32-bit string</td>
<td>Word (unsigned)/16-bit string, Double word (unsigned)/32-bit string</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output Time</td>
<td>Time</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- **WORD_TO_TIME, WORD_TO_TIME_E**
  Converts word (unsigned)/16-bit string type data input to (s) into time type data, and outputs the operation result from (d).

- **DWORD_TO_TIME, DWORD_TO_TIME_E**
  Converts double word (unsigned)/32-bit string type data input to (s) into time type data, and outputs the operation result from (d).
## Operation result

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

- **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 (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 (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.**

### Program example

#### **WORD_TO_TIME(_E)**

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

- **Function without EN/ENO (WORD_TO_TIME)**

  [Structured ladder/FBD]

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

- **Function with EN/ENO (WORD_TO_TIME_E)**

  [Structured ladder/FBD]

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

#### **DWORD_TO_TIME(_E)**

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

- **Function without EN/ENO (DWORD_TO_TIME)**

  [Structured ladder/FBD]

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

- **Function with EN/ENO (DWORD_TO_TIME_E)**

  [Structured ladder/FBD]

  ```plaintext
  g_time1 := DWORD_TO_TIME_E(g_dword1)
  ```
Converting string type to bit type

**STR_TO_BOOL(E)**

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_STRING)</td>
<td>Input</td>
<td>String (1)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**
Converts string type data input to (s) into bit type data, and outputs the operation result from (d).
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></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
```

**Operation result**
- Function without EN/ENO
An operation is executed and the operation value is output from (d).
- 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.
5.1 Type Conversion Functions

Operation error

- No operation error occurs.

Program example

- **STR_TO_BOOL(_E)**
  The program which converts string type data input to (s) into bit type data, and outputs the operation result from (d).
  - Function without EN/ENO (STR_TO_BOOL)
    [Structured ladder/FBD]
    ```plaintext
    g_bool1 := STR_TO_BOOL(g_string1);
    ```
  - Function with EN/ENO (STR_TO_BOOL_E)
    [Structured ladder/FBD]
    ```plaintext
    g_bool3 := STR_TO_BOOL_E(g_bool1, g_string1, g_bool2);
    ```
Converting string type to word (signed), double word (signed) type

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

The following function(s) can go in the dotted squares.

STR_TO_INT, STR_TO_INT_E, STR_TO_DINT, STR_TO_DINT_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_, STRING)</td>
<td>Input</td>
<td>String(6)/(11)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- **STR_TO_INT, STR_TO_INT_E**

Converting string type data input to (s) into word (signed) type data, and outputs the operation result from (d).

String 1st word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of ten-thousands place</td>
<td>Sign data</td>
</tr>
</tbody>
</table>

2nd word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of hundreds place</td>
<td>ASCII code of thousands place</td>
</tr>
</tbody>
</table>

3rd word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of units place</td>
<td>ASCII code of tens place</td>
</tr>
</tbody>
</table>

4th word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H (Indicates the end of the character string.)</td>
<td></td>
</tr>
</tbody>
</table>

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

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

String type data: '-32768' to '32767'

- **STR_TO_DINT, STR_TO_DINT_E**

Converting string type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

String 1st word

<table>
<thead>
<tr>
<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>
</tr>
</tbody>
</table>

2nd word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of ten-millions place</td>
<td>ASCII code of hundred-millions place</td>
</tr>
</tbody>
</table>

3rd word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of hundred-thousands place</td>
<td>ASCII code of millions place</td>
</tr>
</tbody>
</table>

4th word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of thousands place</td>
<td>ASCII code of ten-thousands place</td>
</tr>
</tbody>
</table>

5th word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII code of tens place</td>
<td>ASCII code of hundreds place</td>
</tr>
</tbody>
</table>

6th word

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>00H</td>
<td>ASCII code of units place</td>
</tr>
</tbody>
</table>

Indicates the end of the character string.
The value to be input to (s) 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**

- **Function without EN/ENO**
The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

**Operation error**

- An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/ Q00/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The input value is other than '30H' to '39H', '20H', '2DH', and '00H' of ASCII code.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the STR_TO_INT(_E) instruction is used, the input value is outside the range of -32768 to 32767.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the STR_TO_DINT(_E) instruction is used, the input value is outside the range of -2147483648 to 2147483647.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Program example**

**STR_TO_INT(_E)**
The program which converts string type data input to (s) into word (signed) type data, and outputs the operation result from (d).

- **Function without EN/ENO (STR_TO_INT)**
  [Structured ladder/FBD]
  ```
  g_int1 := STR_TO_INT(g_string1);
  ```

- **Function with EN/ENO (STR_TO_INT_E)**
  [Structured ladder/FBD]
  ```
  g_bool3 := STR_TO_INT_E(g_bool1, g_string1, g_int1);
  ```
**STR_TO_DINT(E)**
The program which converts string type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

- Function without EN/ENO (STR_TO_DINT)

[Structured ladder/FBD]

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

```
Converting string type to single-precision real type

**STR_TO_REAL(_E)**

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

The following function(s) can go in the dotted squares.
STR_TO_REAL, STR_TO_REAL_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_STRING)</td>
<td>Input</td>
<td>String (24)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Single-precision real number</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing
- Converts string type (decimal form/exponential form) data input to (s) into single-precision real type data, and outputs the operation result from (d).

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

High-order byte  Low-order byte

Single-precision real type
Both string type data in decimal form and exponential form can be converted to single-precision real type data.

**Decimal form**

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

-1.35034

**Exponential form**

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

-1.35034E-10

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.

**Decimal form**

-1.35034

**Exponential form**

-1.35034E-10

-1.35034E-10

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.

-1.35034E-10

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

**Ex**

Decimal form

<table>
<thead>
<tr>
<th>Ex</th>
<th>Decimal form</th>
<th>Neglected</th>
<th>Single-precision real type</th>
</tr>
</thead>
</table>
| ![Decimal form example](image)

**Ex**

Exponential form

<table>
<thead>
<tr>
<th>Ex</th>
<th>Exponential form</th>
<th>Neglected</th>
<th>Single-precision real type</th>
</tr>
</thead>
</table>
| ![Exponential form example](image)

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

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

• String type data can contain up to 24 characters. '20H' (space) and '30H' (0) in the character string are counted as one character.

• The value to be input to (s) is string type data within the following range. ASCII code: '30H' to '39H', '45H', '2BH', '2DH', '2EH', '20H', and '00H'

### Operation result

- Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- 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 (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

An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00/J</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>Any characters other than '30H' to '39H' exist in the integer or fractional part. Two or more &quot;2EH&quot;(E) exist. Any characters other than &quot;45H&quot;(E), &quot;2BH&quot;(+), or &quot;2DH&quot;(-) exist in the exponent part, or more than one exponent parts exist. The data after conversion is outside the range of -3.40282(^{38}) to -1.17549(^{38}), 0 or 1.17549(^{38}) to 3.40282(^{38}). The number of characters is 0 or exceeding 24.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
Program example

**STR_TO_REAL(E)**
The program which converts string type data input to (s) into single-precision real type data, and outputs the operation result from (d).

- **Function without EN/ENO (STR_TO_REAL)**
  
  [Structured ladder/FBD]

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

- **Function with EN/ENO (STR_TO_REAL_E)**
  
  [Structured ladder/FBD]

  ```
  g_bool3 := STR_TO_REAL_E(g_bool1, g_string1, g_real1);
  ```
Converting string type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type

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

The following function(s) can go in the dotted squares.
STR_TO_WORD, STR_TO_WORD_E, STR_TO_DWORD, STR_TO_DWORD_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(_STRING)</td>
<td>Input</td>
<td>String(4)/(8)</td>
<td>String(4)/(8)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>Word (unsigned)/16-bit string, Double word (unsigned)/32-bit string</td>
<td></td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- **STR_TO_WORD, STR_TO_WORD_E**
  Converts string type data input to (s) into word (unsigned)/16-bit string type data, and outputs the operation result from (d).

  ![Example](image1)

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

  ![Example](image2)
### Operation result

- Function without EN/ENO

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

- 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 (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE&lt;sup&gt;*&lt;/sup&gt;</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

<sup>*</sup> 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

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.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/ Q00/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LPCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The ASCII code for each number specified for (s) is outside the range of 3OH to 39H, 41H to 46H.</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>O</td>
</tr>
<tr>
<td>4101</td>
<td>The device specified for (s) exceeds the corresponding device range.</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>O</td>
</tr>
</tbody>
</table>

### Program example

#### STR_TO_WORD(E)

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

- **Function without EN/ENO**: STR_TO_WORD

  [Structured ladder/FBD]

  
  ```
  g_string1 := "0012"
  g_word1 := STR_TO_WORD(g_string1);
  ```

- **Function with EN/ENO**: STR_TO_WORD(E)

  [Structured ladder/FBD]

  ```
  g_string1 := "0012";
  g_bool3 := STR_TO_WORD_E(g_bool1, g_string1, g_word1);
  ```
### STR_TO_DWORD(_E)

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

- Function without EN/ENO (STR_TO_DWORD)

  [Structured ladder/FBD]

  ```
  g_string1 := "00000012";
  g_dword1 := STR_TO_DWORD(g_string1);
  ```
Converting string type to time type

**STR_TO_TIME(_E)**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s(_STRING)</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.

STR_TO_TIME, STR_TO_TIME_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_STRING)</td>
<td>Input</td>
<td>String (11)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Time</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

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

```
"000000000" 0ms
"01234567" T#20m34s567ms
```

**Operation result**

- **Function without EN/ENO**

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

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) 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.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00/J</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The ASCII code for each number specified for (s) is outside the range of 30H to 39H, 20H, and 00H. When ASCII data specified for (s) is other than -2147483648 to 4147483647.</td>
<td>—</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

**Program example**

**STR_TO_TIME(_E)**

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

- **Function without EN/ENO (STR_TO_TIME)**

[Structured ladder/FBD]

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

- **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);
```
Converting string type to BCD type

STR_TO_BCD(_E)

The following function(s) can go in the dotted squares.
STR_TO_BCD, STR_TO_BCD_E

■Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_STRING)</td>
<td>Input</td>
<td>String (S)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

■Processing details

■Operation processing

- When word (unsigned)/16-bit string type is specified for output argument.
Convets string type 4-character-string data input to (s) into BCD type data, and outputs the operation result from (d).

When '20H' (space) exists in the character string, the conversion is executed ignoring '20H'. '20H' (space) and '30H' (0) in the character string are counted as one character.
The value to be input to (s) is string type data within the following range.
ASCII code: '30H' to '39H', '20H', and '00H'
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.
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>
• When double word (unsigned)/32-bit string type is specified for output argument.

Converts string type 8-character-string data input to (s) into BCD type data, and outputs the operation result from (d).

When '20H' (space) exists in the character string, the conversion is executed ignoring '20H'. '20H' (space) and '30H' (0) in the character string are counted as one character. The value to be input to (s) is string type data within the following range.

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

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

Output from (d) 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

- **Function without EN/ENO**

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

### Operation error

- An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00/J Q00/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The input character string is outside the range of ASCII code '30H' to '39H', '20H', and '00H'.</td>
<td>—</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
Program example

**STR_TO_BCD(_E) (when word (unsigned)/16-bit string is specified for (d))**

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

- Function without EN/ENO (STR_TO_BCD)

[Structured ladder/FBD]

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

- 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);
```

**STR_TO_BCD(_E) (when double word (unsigned)/32-bit string is specified for (d))**

The program which converts string type data input to (s) into BCD type data in double word (unsigned)/32-bit string type data, and outputs the operation result from (d).

- Function without EN/ENO (STR_TO_BCD)

[Structured ladder/FBD]

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

- 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);
```
Converting BCD type to word (signed), double word (signed) type

**BCD_TO_INT(_E), BCD_TO_DINT(_E)**

The following function(s) can go in the dotted squares.

BCD_TO_INT, BCD_TO_INT_E, BCD_TO_DINT, BCD_TO_DINT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_BCD)</td>
<td>Input</td>
<td>Word (unsigned)/16-bit string, Double word (unsigned)/32-bit string</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- BCD_TO_INT, BCD_TO_INT_E

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

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

The value to be input to (s) is word (unsigned)/16-bit string type data within the range from 0H to 9999H (0 to 9 for each digit).
• BCD_TO_DINT, BCD_TO_DINT_E
Converts BCD type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

  - When word (unsigned)/16-bit string is specified for (s)
    - **Word (unsigned)/16-bit string type**
    - **Double word (signed) type**

  9999H
  - Thousands place
  - Hundreds place
  - Tens place
  - Units place
  - **DINT conversion**

  9999
  - 0000010111101110000111111

Always filled with 0s.

  - When double word (unsigned)/32-bit string is specified for (s)
    - **Double word (unsigned)/32-bit string type**
    - **Double word (signed) type**

  99999999H
  - Ten millions place
  - Millions place
  - Hundred thousands place
  - Ten thousands place
  - Thousands place
  - Hundreds place
  - Tens place
  - Units place
  - **DINT conversion**

  99999999
  - 0000010111101110000111111

Always filled with 0s.

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

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

### Operation result

- **Function without EN/ENO**
  - An operation is executed and the operation value is output from (d).
- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.
**Operation error**

These functions consist of the following common instructions.

**BCD_TO_INT( _E )**: BIN

**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.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J</th>
<th>Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>Values other than 0 to 9 are specified for each digit of (s).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

**Program example**

**BCD_TO_INT( _E )**

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

- Function without EN/ENO (BCD_TO_INT)
  - [Structured ladder/FBD]
  
  ```plaintext
  g_int1 := BCD_TO_INT(g_word1);
  ```

- Function with EN/ENO (BCD_TO_INT_E)
  - [Structured ladder/FBD]
  
  ```plaintext
  g_bool3 := BCD_TO_INT_E(g_bool1, g_word1, g_int1);
  ```

**BCD_TO_DINT( _E )**

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

- Function without EN/ENO (BCD_TO_DINT)
  - [Structured ladder/FBD]
  
  ```plaintext
  g_dint1 := BCD_TO_DINT(g_dword1);
  ```
Converting BCD type to string type

**BCD_TO_STR(E)**

The following function(s) can go in the dotted squares.

BCD_TO_STR, BCD_TO_STR_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_BCD)</td>
<td>Input</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String (8)</td>
</tr>
</tbody>
</table>

#### Processing details

**Operation processing**

- Converts BCD type data input to (s) into string type data, and outputs the operation result from (d).

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

![Diagram showing conversion from word (unsigned) to string type]

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

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

![Diagram showing conversion from double word (unsigned) to string type]

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

- Word (unsigned)/16-bit string type, double word (unsigned)/32-bit string type data can be specified for (d). Bit type cannot be specified.
- When SM701 (signal for switching the number of output characters) is OFF, "00H" is stored to the end of the character string.

**Operation result**

- Function without EN/ENO
  An operation is executed and the operation value is output from (d).
- 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>
</tbody>
</table>
| FALSE  | FALSE*1 | Undefined value *

*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

These functions consist of the following common instructions.
When word (unsigned)/16-bit string type is specified for (s): BCDDA
When double word (unsigned)/32-bit string type is specified for (d): 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.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/Q00/</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>When word (unsigned)/16-bit string type is specified for (s), (s) is outside the range of 0 to 9999. When double word (unsigned)/32-bit string type is specified for (s), (s) is outside the range of 0 to 99999999.</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4101</td>
<td>The device specified for (d) exceeds the corresponding device range.</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Program example

**BCD_TO_STR(_E) (when word (unsigned)/16-bit string is specified for (s))**

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

• Function without EN/ENO (BCD_TO_STR)

[Structured ladder/FBD]

```
g_string1 := BCD_TO_STR(g_word1);
```

• Function with EN/ENO (BCD_TO_STR_E)

[Structured ladder/FBD]

```
g_bool3 := BCD_TO_STR_E(g_bool1, g_word1, g_string1);
```

**BCD_TO_STR(_E) (when double word (unsigned)/32-bit string is specified for (s))**

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

• Function without EN/ENO (BCD_TO_STR)

[Structured ladder/FBD]

```
g_string1 := BCD_TO_STR(g_dword1);
```

• Function with EN/ENO (BCD_TO_STR_E)

[Structured ladder/FBD]

```
g_bool3 := BCD_TO_STR_E(g_bool1, g_dword1, g_string1);
```
**Converting time type to bit type**

**TIME_TO_BOOL(_E)**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO=TIME_TO_BOOL_E(EN, s, d);</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.
TIME_TO_BOOL, TIME_TO_BOOL_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(_TIME)</td>
<td>Input</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>Bit</td>
<td></td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

Converts time type data input to (s) into bit type data, and outputs the operation result from (d).

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.

#### Operation result

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

Program example

**TIME_TO_BOOL(E)**

The program which converts time type data input to (s) into bit type data, and outputs the operation result from (d).

• Function without EN/ENO (TIME_TO_BOOL)

[Structured ladder/FBD]

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

• Function with EN/ENO (TIME_TO_BOOL_E)

[Structured ladder/FBD]

```plaintext
[ST]
g_bool3 := TIME_TO_BOOL_E(g_bool1, g_time1, g_bool2);
```
Converting time type to word (signed), double word (signed) type

**TIME_TO_INT(E), TIME_TO_DINT(E)**

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

The following function(s) can go in the dotted squares.
TIME_TO_INT, TIME_TO_INT_E, TIME_TO_DINT, TIME_TO_DINT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_TIME)</td>
<td>Input</td>
<td>Time</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed), double word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**
- **TIME_TO_INT, TIME_TO_INT_E**
Converting time type data input to (s) into word (signed) type data, and outputs the operation result from (d).

\[
\begin{align*}
t#1s234ms & \rightarrow 1234 \\
\text{Time type} & \rightarrow \text{Word (signed) type}
\end{align*}
\]

When converting to word (signed) type data, high-order 16-bit (1 word) data of time type is discarded.
- **TIME_TO_DINT, TIME_TO_DINT_E**
Converting time type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

\[
\begin{align*}
t20m34s567ms & \rightarrow 1234567 \\
\text{Time type} & \rightarrow \text{Double word (signed) type}
\end{align*}
\]

**Operation result**
- **Function without EN/ENO**
An operation is executed and the operation value is output from (d).
- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.
5 APPLICATION FUNCTIONS
5.1 Type Conversion Functions

Operation error

• No operation error occurs.

Program example

**TIME_TO_INT(_E)**
The program which converts time type data input to (s) into word (signed) type data, and outputs the operation result from (d).

• Function without EN/ENO (TIME_TO_INT)

[Structured ladder/FBD]

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

• Function with EN/ENO (TIME_TO_INT_E)

[Structured ladder/FBD]

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

**TIME_TO_DINT(_E)**
The program which converts time type data input to (s) into double word (signed) type data, and outputs the operation result from (d).

• Function without EN/ENO (TIME_TO_DINT)

[Structured ladder/FBD]

```
[ST]
g_dint1 := TIME_TO_DINT(g_time1);
```
Converting time type to string type

TIME_TO_STR(_E)

The following function(s) can go in the dotted squares.
TIME_TO_STR, TIME_TO_STR_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_BCD)</td>
<td>Input</td>
<td>Time</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String (11)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**
- Converts time type data input to (s) into string type data, and outputs the operation result from (d).

![Structured ladder/FBD diagram]

**Operation result**
- Function without EN/ENO
  An operation is executed and the operation value is output from (d).
- 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) 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 SD0.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/ Q00/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4101</td>
<td>The device specified for (d) exceeds the corresponding device range.</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Program example

**TIME_TO_STR(_E)**

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

- **Function without EN/ENO (TIME_TO_STR)**
  
  [Structured ladder/FBD]

  [ST]
  
  g_string := TIME_TO_STR(g_time);

- **Function with EN/ENO (TIME_TO_STR_E)**
  
  [Structured ladder/FBD]

  [ST]
  
  g_bool := TIME_TO_STR_E(g_bool, g_time, g_string);
Converting time type to word (unsigned)/16-bit string, double word (unsigned)/32-bit string type

### TIME_TO_WORD(_E), TIME_TO_DWORD(_E)

The following function(s) can go in the dotted squares.
TIME_TO_WORD, TIME_TO_WORD_E, TIME_TO_DWORD, TIME_TO_DWORD_E

#### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_TIME)</td>
<td>Input</td>
<td>Time</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (unsigned)/16-bit string, double word (unsigned)/32-bit string</td>
</tr>
</tbody>
</table>

#### Processing details

**Operation processing**
- TIME_TO_WORD, TIME_TO_WORD_E
Converting time type data input to (s) into word (unsigned)/16-bit string type data, and outputs the operation result from (d).

- When converting to word (unsigned)/16-bit string type data, high-order 16-bit (1 word) data is discarded.
- TIME_TO_DWORD, TIME_TO_DWORD_E
Converting time type data input to (s) into double word (unsigned)/32-bit string type data, and outputs the operation result from (d).
5 APPLICATION FUNCTIONS
5.1 Type Conversion Functions

■ Operation result
- Function without EN/ENO
An operation is executed and the operation value is output from (d).
- 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 (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 (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.

■ Program example

■ TIME_TO_WORD( _E)
The program which converts time type data input to (s) into word (unsigned)/16-bit string type data, and outputs the operation result from (d).
- Function without EN/ENO (TIME_TO_WORD)
  [Structured ladder/FBD]
  ```
  [ST] g_word1 := TIME_TO_WORD(g_time1);
  ```
- Function with EN/ENO (TIME_TO_WORD_E)
  [Structured ladder/FBD]
  ```
  [ST] g_bool3 := TIME_TO_WORD_E(g_bool1, g_time1, g_word1);
  ```

■ TIME_TO_DWORD( _E)
The program which converts time type data input to (s) into double word (unsigned)/32-bit string type data, and outputs the operation result from (d).
- Function without EN/ENO (TIME_TO_DWORD)
  [Structured ladder/FBD]
  ```
  [ST] g_dword1 := TIME_TO_DWORD(g_time1);
  ```
Converting bit array to word (signed) type, word (unsigned)/16-bit string type, double word (signed) type, double word (unsigned)/32-bit string type

**BITARR_TO_INT(_E), BITARR_TO_DINT(_E)**

The following function(s) can go in the dotted squares.
BITARR_TO_INT, BITARR_TO_INT_E, BITARR_TO_DINT, BITARR_TO_DINT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>_s(_BitArr)</td>
<td>Input (Variables are applicable to element specification.)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>Input (Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified)</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output ANY16, ANY32</td>
<td></td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

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

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

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

Program example

**BITARR_TO_INT(E)**

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

- Function without EN/ENO (BITARR_TO_INT)

  [Structured ladder/FBD]

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

- Function with EN/ENO (BITARR_TO_INT_E)

  [Structured ladder/FBD]

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

**INT_TO_BITARR(_E), DINT_TO_BITARR(_E)**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN ENO</td>
<td>ST</td>
</tr>
<tr>
<td>s d</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>ENO := INT_TO_BITARR_E[EN, s, n, d];</td>
<td></td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares. 
INT_TO_BITARR, INT_TO_BITARR_E, DINT_TO_BITARR, DINT_TO_BITARR_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>Input</td>
<td>ANY16, ANY32</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>Input (Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified)</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d(_BitArr)</td>
<td>Output (Variables are applicable to element specification.)</td>
<td>Bit</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- **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 (s) to (d).

  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.

- **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 (s) to (d).

  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.

#### Operation result

- **Function without EN/ENO**

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

- **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>(d)</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 (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.

Program example

**INT_TO_BITARR(_E)**

The program which outputs low-order 4 bits of word (signed) type data input to (s) to (d).

• Function without EN/ENO (INT_TO_BITARR)

[Structured ladder/FBD]

```
  g_bool4[0] := INT_TO_BITARR(g_int1, 4);
```

• Function with EN/ENO (INT_TO_BITARR_E)

[Structured ladder/FBD]

```
g_bool2 := INT_TO_BITARR_E(g_bool1, g_int1, 4, g_bool4[0]);
```
Bit array copy

CPY_BITARR(E)

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
</tr>
<tr>
<td>s (_BitArrIn)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ST</td>
</tr>
<tr>
<td>ENO:= CPY_BITARR_E (EN, s, n, d),</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.
CPY_BITARR, CPY_BITARR_E

- **Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s (_BitArrIn)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>Input (Only a constant 4, 8, 12, 16, 20, 24, 28 or 32 can be specified)</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d (_BitArrOut)</td>
<td>Output (Variables are applicable to element specification.)</td>
<td>Bit</td>
</tr>
</tbody>
</table>

- **Processing details**

- **Operation processing**
Outputs n bits of bit array input to (s) to (d).

- **Operation result**
  - Function without EN/ENO
  An operation is executed and the operation value is output from (d).
  - 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.
5 APPLICATION FUNCTIONS

5.1 Type Conversion Functions

Operation error

• No operation error occurs.

Program example

**CPY_BITARR(E)**

The program which outputs 12 bits from num1 element of bit string input to (s) to num2 and the following bits of (d).

• Function without EN/ENO (CPY_BITARR)

[Structured ladder/FBD]

![Structured ladder/FBD diagram]

```
[ST]
g_bool5[num2] := CPY_BITARR(g_bool4[num1], 12);
```

• Function with EN/ENO (CPY_BITARR_E)

[Structured ladder/FBD]

![Structured ladder/FBD diagram]

```
[ST]
g_bool2 := CPY_BITARR_E(g_bool1, g_bool4[num1], 12, g_bool5[num2]);
```
Specified bit read of word (signed) type data

**GET_BIT_OF_INT(E)**

**Structured ladder/FBD**

```
EN ENS
s d
n
ENO = GET_BIT_OF_INT_E(EN, s, n, d);
```

The following function(s) can go in the dotted squares.
GET_BIT_OF_INT, GET_BIT_OF_INT_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>Input</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>Input (Only a constant between 0 and 15 can be specified)</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing
Reads a value of nth bit of (s), and outputs the operation result from (d).

#### Operation result
- Function without EN/ENO
  An operation is executed and the operation value is output from (d).
- 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</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.

Program example

**GET_BIT_OF_INT(·E)**
The program which reads a value of 5th bit of data input to (s), and outputs the operation result from (d).

- Function without EN/ENO (GET_BIT_OF_INT)
  
  ![Structured ladder/FBD](image1)

  [ST]
  ```
  g_bool3 := GET_BIT_OF_INT(g_int1, 5);
  ```

- Function with EN/ENO (GET_BIT_OF_INT_E)
  
  ![Structured ladder/FBD](image2)

  [ST]
  ```
  g_bool2 := GET_BIT_OF_INT_E(g_bool1, g_int1, 5, g_bool3);
  ```
Specified bit write of word (signed) type data

**SET_BIT_OF_INT(_E)**

```plaintext
EN  ENO
  s  d
```

The following function(s) can go in the dotted squares.
SET_BIT_OF_INT, SET_BIT_OF_INT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>Input (Only a constant between 0 and 15 can be specified)</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

Writes a value specified for (s) to the nth bit of (d).

**Operation result**

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

Program example

SET_BIT_OF_INT(E)
The program which writes a value specified for (s) to the 3rd bit of (d).

• Function without EN/ENO (SET_BIT_OF_INT)

[Structured ladder/FBD]

[ST]
g_int3 := SET_BIT_OF_INT(g_bool1, 3);

• 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);
Specified bit copy of word (signed) type data

**CPY_BIT_OF_INT(_E)**

*Basic* | High performance | Process Redundant | Universal | LCPU

**Structured ladder/FBD**

The following function(s) can go in the dotted squares.

CPY_BIT_OF_INT, CPY_BIT_OF_INT_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>Input</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>n1</td>
<td>Input (Only a constant between 0 and 15 can be specified)</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>n2</td>
<td>Input (Only a constant between 0 and 15 can be specified)</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal execution, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

Copies a value of (n1)th bit of input (s) to the (n2)th bit of output (d).

**Operation result**

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

- 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 (Operation execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Operation stop)</td>
<td>FALSE&lt;sup&gt;*&lt;/sup&gt;</td>
<td>Undefined value</td>
</tr>
</tbody>
</table>

<sup>*</sup> 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.

Program example

**CPY_BIT_OF_INT(_E)**

The program which writes a value of 5th bit of (s) to the 3rd bit of (d).

- Function without EN/ENO (CPY_BIT_OF_INT)

  [Structured ladder/FBD]

  ```plaintext
  g_int3 := CPY_BIT_OF_INT(g_int1, 5, 3);
  ```

- Function with EN/ENO (CPY_BIT_OF_INT_E)

  [Structured ladder/FBD]

  ```plaintext
  g_bool3 := CPY_BIT_OF_INT_E(g_bool2, g_int1, 5, 3, g_int3);
  ```
Nonessential type conversion

**GET_BOOL_ADDR, GET_INT_ADDR, GET_WORD_ADDR**

- **Structured ladder/FBD**
  - GET_BOOL_ADDR (s);d:=
  - ST

The following function(s) can go in the dotted squares.
GET_BOOL_ADDR, GET_INT_ADDR, GET_WORD_ADDR

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s</td>
<td>Input</td>
<td>ANY</td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>Bit, word (signed), Word (unsigned)/16-bit string</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing
- Outputs data type of (s) as data type of (d).

<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 real number, Array of time type</td>
<td>Word (unsigned)/16-bit string</td>
</tr>
</tbody>
</table>

- Rounding error may occur when specifying the input value to (s) by programming tool. For precautions when setting an input value using a programming tool, refer to the following.

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

#### Operation result

An operation is executed and the operation value is output from (d).
Operation error

- No operation error occurs.

Program example

**GET_INT_ADDR**
The program which directly handles 32-bit input variable Var_D10 as 16-bit input data without the type conversion.

[Structured ladder/FBD]

```
Var_D10 := GET_INT_ADDR(Var_D10);
```

[ST]

```
Var_D10 := GET_INT_ADDR(Var_D10);
```
5.2 Standard Functions of One Numeric Variable

Absolute value

ABS(E)

**Input/output argument**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(IN)</td>
<td>Input ANY_NUM</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>ENO</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output ANY_NUM</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

**Operation processing**

- Outputs the absolute value of word (signed), double word (signed), single-precision real or double-precision real type data input to (s) from (d) in the same data type as that of (s).

Assuming that the input value is A and the operation output value is B, the relationship is expressed by the following equality.

$$B = |A|$$

- The value to be input to (s) is word (signed), double word (signed), single-precision real or double-precision real type data.
- When the data type of (s) is word (signed) type and the input value is -32768, -32768 is output from (d).
- When the data type of (s) is double word (signed) type and the input value is -2147483648, -2147483648 is output from (d).
- (No operation error occurs. In case of ABS_E, TRUE is output from ENO.)
- Rounding error may occur when specifying single-precision real or double-precision real type data to (s) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

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

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

Program example

**ABS(E) (when word (signed) is specified for (s))**

The program which outputs the absolute value of word (signed) type data input to (s) from (d) in the same data type as that of (s).

- Function without EN/ENO (ABS)
  [Structured ladder/FBD]

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

- Function with EN/ENO (ABS_E)
  [Structured ladder/FBD]

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

**ABS(E) (when double word (signed) is specified for (s))**

The program which outputs the absolute value of double word (signed) type data input to (s) from (d) in the same data type as that of (s).

- Function without EN/ENO (ABS)
  [Structured ladder/FBD]

  ```
  g_dint2 := ABS(g_dint1);
  ```

- Function with EN/ENO (ABS_E)
  [Structured ladder/FBD]

  ```
  g_bool2 := ABS_E(g_bool1, g_dint1, g_dint2);
  ```
**ABS**(E) (when single-precision real type data is specified for (s))

The program which outputs the absolute value of single-precision real type data input to (s) from (d) in the same data type as that of (s).

- Function without EN/ENO (ABS)
  
  ```
  [Structured ladder/FBD]
  ```

  ```
  [ST]
  g_real2 := ABS(g_real1);
  ```

- Function with EN/ENO (ABS_E)
  
  ```
  [Structured ladder/FBD]
  ```

  ```
  [ST]
  g_bool2 := ABS_E(g_bool1, g_real1, g_real2);
  ```

**ABS**(E) (when double-precision real type data is specified for (s))

The program which outputs the absolute value of double-precision real type data input to (s) from (d) in the same data type as that of (s).

- Function without EN/ENO (ABS)
  
  ```
  [Structured ladder/FBD]
  ```

  ```
  [ST]
  g_lreal2 := ABS(g_lreal1);
  ```

- Function with EN/ENO (ABS_E)
  
  ```
  [Structured ladder/FBD]
  ```

  ```
  [ST]
  g_bool2 := ABS_E(g_bool1, g_lreal1, g_lreal2);
  ```
5.3 Standard Arithmetic Functions

Addition

**ADD_E**

<table>
<thead>
<tr>
<th>Basic</th>
<th>High performance</th>
<th>Process</th>
<th>Redundant</th>
<th>Universal</th>
<th>LCPU</th>
</tr>
</thead>
</table>

**Structured ladder/FBD**

```
EN | ENO
s1 | d
s2
s28
```

The following function(s) can go in the dotted squares.

ADD_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1 to s28(_IN)</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- Performs addition ((s1)+(s2)+...+(s28)) on word (signed), double word (signed), single-precision real or double-precision real type data input to (s1) to (s28), and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

**Ex**

Word (signed) type data

- The values to be input to (s1) to (s28) are word (signed), double word (signed), single-precision real or double-precision real type data.
- The number of pins of (s) can be changed in the range from 2 to 28.
- If an underflow/overflow occurs in the operation result, data is output from (d) as follows.

Word (signed) type data

No operation error occurs even if an underflow/overflow occurs.

In case of ADD_E, TRUE is output from ENO.

\[
32767 + 2 = -32767 \\
(7FFFH) (002H) (8001H) \\
-32766 + (-2) = 32766 \\
(8000H) (FFFEH) (7FFE) \\
\]

Since the highest-order bit is 1, the result value is negative.

Since the highest-order bit is 0, the result value is positive.
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.

\[
2147483647 + 2 = -2147483647
\]

Since the highest-order bit is 1, the result value is negative.

\[
(7FFFFFFFH) \text{ (} 0002H \text{) } (80000001H)
\]

\[
-2147483648 + (-2) = 2147483646
\]

Since the highest-order bit is 0, the result value is positive.

\[
(80000000H) \text{ (} FFFEH \text{) } (7FFFFFEH)
\]

• Rounding error may occur when specifying single-precision real or double-precision real type data to (s1) through (s28) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

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

### Operation result

- Function without EN/ENO

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

- 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 (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 (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.

### Program example

**ADD_E**

The program which performs addition ((s1) + (s2)) on double word (signed) type data input to (s1) and (s2), and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

[Structured ladder/FBD]

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

### MUL_E

#### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1 to s28(_IN)</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

#### Processing details

**Operation processing**

- Performs multiplication \((s1 \times s2) \times \ldots \times (s28)\) on word (signed), double word (signed), single-precision real or double-precision real type data input to \((s1)\) to \((s28)\), and outputs the operation result from \((d)\) in the same data type as that of \((s1)\) to \((s28)\).

**Ex**

Word (signed) type data

\[
\begin{array}{c}
100 \\
15
\end{array} \times \begin{array}{c}
1500
\end{array}
\]

- The values to be input to \((s1)\) to \((s28)\) are word (signed), double word (signed), single-precision real or double-precision real type data.
- The number of pins of \((s)\) can be changed in the range from 2 to 28.
- If an underflow/overflow occurs in the operation result, data is output from \((d)\) as follows.

**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 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.

**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.
Rounding error may occur when specifying single-precision real or double-precision real type data to (s1) through (s28) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

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

### Operation result

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

- 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

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.

### Program example

- **MUL_E**
  
  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 (d) in the same data type as that of (s1) and (s2).

  [Structured ladder/FBD]

  [ST]

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

**SUB_E**

#### Arguments

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_IN1)</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td></td>
<td>s2(_IN2)</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

#### Processing details

**Operation processing**

- Performs subtraction ((s1) - (s2)) on word (signed), double word (signed), single-precision real or double-precision real type data input to (s1) and (s2), and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

**Ex**

Word (signed) type data

\[
\begin{align*}
12345 & - 6789 = 5556 \\
-(2) & = -32767 & \text{Since the highest-order bit is 1, the result value is negative.} \\
(7FFFH) (FFFEH) (8001H) & -32766 & \text{Since the highest-order bit is 0, the result value is positive.} \\
2 & = 32766 & \text{Since the highest-order bit is 0, the result value is positive.} \\
(8000H) (0002H) (7FFEH) & \end{align*}
\]

Double word (signed) type data

\[
\begin{align*}
2147483647 & - (2) = -2147483647 & \text{Since the highest-order bit is 1, the result value is negative.} \\
(7FFFFFFF) (FFFFEH) (80000001H) & -2147483646 & \text{Since the highest-order bit is 0, the result value is positive.} \\
2 & = 2147483646 & \text{Since the highest-order bit is 0, the result value is positive.} \\
(80000000H) (0002H) (7FFFFFFEH) & \end{align*}
\]
• Rounding error may occur when specifying single-precision real or double-precision real type data to (s1), (s2) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

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

### Operation result

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

### Program example

#### SUB_E

The program which performs subtraction ((s1) - (s2)) on double word (signed) type data input to (s1) and (s2), and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

[Structured ladder/FBD]

[ST]

g_bool3 := SUB_E(g Bool1, g Dint1, g Dint2, g Dint3);
Division

DIV_E

The following function(s) can go in the dotted squares.

DIV_E

■Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_IN1)</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td></td>
<td>s2(_IN2)</td>
<td></td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

■Operation processing

- Performs division ((s1 ÷ s2)) on word (signed), double word (signed), single-precision real or double-precision real type data input to (s1) and (s2), and outputs the quotient of the operation result from (d) in the same data type as that of (s1) and (s2).

Ex.

Word (signed) type data

\[ \begin{array}{ccc}
5 & \div & 2 \\
\text{(Quotient)} & = & 2 \\
\text{(Remainder)} & = & 1 \\
\end{array} \]

- The values to be input to (s1) and (s2) are word (signed), double word (signed), single-precision real or double-precision real type data. (The value to be input to (s2) must be other than 0.)
- Rounding error may occur when specifying single-precision real or double-precision real type data to (s1), (s2) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

- MELSEC-Q/L/F Structured Programming Manual (Fundamentals)
### Operation result

- **Function without EN/ENO**

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

### Operation error

- An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/Q00/Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The value to be input to (s2) is 0. (Division by 0)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Program example

#### DIV_E

The program which performs division \( (s1 \div s2) \) on double word (signed) type data input to (s1) and (s2), and outputs the quotient of the operation result from (d) in the same data type as that of (s1) and (s2).

[Structured ladder/FBD]

```
[ST]
g_bool3 := DIV_E(g_bool1, g_dint1, g_dint2, g_dint3);
```
Remainder

MOD(E)

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1_IN1</td>
<td>Input</td>
<td>ANY_INT</td>
</tr>
<tr>
<td></td>
<td>s2_IN2</td>
<td></td>
<td>ANY_INT</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY_INT</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- Performs division \((s1) \div (s2)\) on word (signed) or double word (signed) type data input to \((s1)\) and \((s2)\), and outputs the remainder of the operation result from \((d)\) in the same data type as that of \((s1)\) and \((s2)\).

**Ex**

Word (signed) type data

\[
\begin{array}{c}
5 \\
\hline
2
\end{array}
\]

Word (signed) type \quad Word (signed) type \quad No output \quad Word (signed) type

- The values to be input to \((s1)\) and \((s2)\) are word (signed) or double word (signed) type data. (Note that the value to be input to \((s2)\) must be other than 0.)

**Operation result**

- Function without EN/ENO

The following table shows the operation results.

<table>
<thead>
<tr>
<th>Operation result</th>
<th>(d)</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>

- 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 (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 \((d)\) is undefined. In this case, create a program so that the data output from \((d)\) is not used.
Operation error

- An operation error occurs in the following case.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/00/01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The value to be input to (s2) is 0. (Division by 0)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Program example

**MOD(_E)**
The program which performs division \( (s1) \div (s2) \) on double word (signed) type data input to \( (s1) \) and \( (s2) \), and outputs the remainder of the operation result from \( (d) \) in the same data type as that of \( (s1) \) and \( (s2) \).

- Function without EN/ENO (MOD)
  - [Structured ladder/FBD]
  ```
  g_dint3 := (g_dint1) MOD (g_dint2);
  ```
  - [ST]
  ```
  g_dint3 := (g_dint1) MOD (g_dint2);
  ```

- Function with EN/ENO (MOD_E)
  - [Structured ladder/FBD]
  ```
  g_bool3 := MOD_E(g_bool1, g_dint1, g_dint2, g_dint3);
  ```
  - [ST]
  ```
  g_bool3 := MOD_E(g_bool1, g_dint1, g_dint2, g_dint3);
  ```
Exponentiation

EXPT(_E)

The following function(s) can go in the dotted squares.
EXPT, EXPT _E

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_IN1)</td>
<td>Input</td>
<td>ANY_REAL</td>
</tr>
<tr>
<td></td>
<td>s2(_IN2)</td>
<td>Input</td>
<td>ANY_REAL</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY_REAL</td>
</tr>
</tbody>
</table>

Processing details

Operation processing

• Performs exponentiation (s2) on single-precision real or double-precision real type data input to (s2) and word (signed), double word (signed), single-precision real or double-precision real type data input to (s1), and outputs the operation result from (d).

• Rounding error may occur when specifying single-precision real or double-precision real type data to (s1), (s2) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

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

Operation result

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

• 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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.
### Operation error

These functions consist of the following common instructions depending on the data type of (s1) and (s2).

<table>
<thead>
<tr>
<th>Data type of (s1)</th>
<th>Data type of (s2)</th>
<th>Common instruction used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-precision real type</td>
<td>Word (signed) type</td>
<td>LOG, FLT</td>
</tr>
<tr>
<td>Double word (signed) type</td>
<td>LOG, DFLT</td>
<td></td>
</tr>
<tr>
<td>Single-precision real type</td>
<td>LOG</td>
<td></td>
</tr>
<tr>
<td>Double-precision real type</td>
<td>LOGD, DFLTD</td>
<td></td>
</tr>
<tr>
<td>Double-precision real type</td>
<td>LOGD</td>
<td></td>
</tr>
</tbody>
</table>

• For details of an error which occurs when the function is executed, refer to the following.
  □ □ □ □ MELSEC-Q/L Structured Programming Manual (Common Instructions)

### Program example

#### EXPT(E)

The program which performs exponentiation and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

• Function without EN/ENO (EXPT)

[Structured ladder/FBD]

```plaintext
  1
  [ST] g_real2 := EXPT(g_real1, g_int1);
```

• Function with EN/ENO (EXPT_E)

[Structured ladder/FBD]

```plaintext
  2
  [ST] g_bool3 := EXPT_E(g_bool1, g_real1, g_int1, g_real2);
```
Move operation

MOVE(_E)

The following function(s) can go in the dotted squares.
MOVE, MOVE_E

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>坭(T _IN)</td>
<td>s(_IN)</td>
<td>Input</td>
<td>ANY</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>坭(T d)</td>
<td>d</td>
<td>Output</td>
<td>ANY</td>
</tr>
</tbody>
</table>

Processing details

Operation processing

- Moves the data input for (s) from (d) in the same data type as that of (s).
- The values to be specified to (s) and (d) 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 (s) and (d).

Word (signed) type

<table>
<thead>
<tr>
<th>s</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>12</td>
</tr>
</tbody>
</table>

Double word (signed) type

<table>
<thead>
<tr>
<th>s</th>
<th>2147483647</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>2147483647</td>
</tr>
</tbody>
</table>

Word (unsigned)/16-bit string type

<table>
<thead>
<tr>
<th>s</th>
<th>65535</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>65535</td>
</tr>
</tbody>
</table>

Double word (unsigned)/32-bit string type

<table>
<thead>
<tr>
<th>s</th>
<th>4294967295</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

Single-precision real type

<table>
<thead>
<tr>
<th>s</th>
<th>3.402823×10^38</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>3.402823×10^38</td>
</tr>
</tbody>
</table>

Double-precision real type

<table>
<thead>
<tr>
<th>s</th>
<th>1.79769313486231×10^308</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>1.79769313486231×10^308</td>
</tr>
</tbody>
</table>
• Rounding error may occur when specifying single-precision real or double-precision real type data to (s) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.
* MELSEC-Q/L/F Structured Programming Manual (Fundamentals)

**Operation result**

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

**Program example**

**MOVE(_E)**

The program which moves the word (signed) type data input to (s) to (d).

- Function without EN/ENO (MOVE)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

  ```plaintext
  g_int2 := MOVE(g_int1);
  ```

- Function with EN/ENO (MOVE_E)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

  ```plaintext
  g_bool3 := MOVE_E(g_bool1, g_int1, g_int2);
  ```
5.4 Standard Bitwise Boolean Functions

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

AND_E, OR_E, XOR_E, NOT(E)

| Basic | High performance | Process | Redundant | Universal | CPU |

Structured ladder/FBD

\[ \text{AND}_E(EN, s1, s2, \ldots, s28, d); \]

\[ \text{ENO} = \text{AND}_E(EN, s1, s2, \ldots, s28, d); \]

The following function(s) can go in the dotted squares.
AND_E, OR_E, XOR_E, NOT, NOT_E

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s1 to s28(_IN)</td>
<td>Input</td>
<td>ANY_BIT</td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>ANY_BIT</td>
<td></td>
</tr>
</tbody>
</table>

Processing details

Operation processing

- AND_E

Performs Boolean AND on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to (s1) to (s28) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

Ex.

Word (unsigned)/16-bit string type data

\[ \begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\end{array} \]

AND

\[ \begin{array}{cccccccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array} \]

Boolean AND

\[ \begin{array}{cccccccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array} \]

The number of pins of variable 's' can be changed in the range from 2 to 28.
• OR_E
Performs Boolean OR on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to (s1) to (s28) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

Ex
Word (unsigned)/16-bit string type data

<table>
<thead>
<tr>
<th>s1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The number of pins of variable 's' can be changed in the range from 2 to 28.

• XOR_E
Performs Boolean exclusive OR on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to (s1) to (s28) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

Ex
Word (unsigned)/16-bit string type data

<table>
<thead>
<tr>
<th>s1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The number of pins of variable 's' can be changed in the range from 2 to 28.

When three or more variables 's' exist, XOR is performed between (s1) and (s2) first, and XOR is successively performed between the result and (s3).

When the expression includes (s4), XOR is performed between the result of XOR with (s3) and (s4).

In this manner, XOR is repeated by the number of variables 's' in the order with (s5), (s6) and so on.

Ex
Bit type data

For 3 INs

\[\begin{array}{c}
\text{FALSE} \\
\text{XOR} \\
\text{TRUE} \\
\end{array}\]

Result

For 4 INs

\[\begin{array}{c}
\text{FALSE} \\
\text{XOR} \\
\text{FALSE} \\
\text{XOR} \\
\text{TRUE} \\
\end{array}\]

Result

For 5 INs

\[\begin{array}{c}
\text{TRUE} \\
\text{XOR} \\
\text{TRUE} \\
\text{XOR} \\
\text{FALSE} \\
\end{array}\]

Result

Hereafter, XOR is repeated by the number of s.

• NOT, NOT_E
Performs Boolean NOT on bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data input to (s1) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1).

Ex
Word (unsigned)/16-bit string type data

<table>
<thead>
<tr>
<th>s1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The value to be input to variables (s1) to (s28) is bit, word (unsigned)/16-bit string or double word (unsigned)/32-bit string type data.
**Operation result**

- Function without EN/ENO

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

- 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 (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 (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.

**Program example**

- **AND_E**

The program which performs Boolean AND on bit, word (unsigned)/16-bit string type data input to (s1) to (s28) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

```
[Structured ladder/FBD]

[ST] g_bool2 := AND_E(g_bool1, g_word1, g_word2, g_word3);
```

- **OR_E**

The program which performs Boolean OR on bit, word (unsigned)/16-bit string type data input to (s1) to (s28) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

```
[Structured ladder/FBD]

[ST] g_bool2 := OR_E(g_bool1, g_word1, g_word2, g_word3);
```

- **XOR_E**

The program which performs Boolean XOR on bit, word (unsigned)/16-bit string type data input to (s1) to (s28) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

```
[Structured ladder/FBD]

[ST] g_bool2 := XOR_E(g_bool1, g_word1, g_word2, g_word3);
```
5 APPLICATION FUNCTIONS

5.4 Standard Bitwise Boolean Functions

- **NOT(E)**
  The program which performs Boolean NOT on bit, word (unsigned)/16-bit string type data input to (s1) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1).

  - Function without EN/ENO (NOT)
    [Structured ladder/FBD]

```
[ST]
g_word2 := NOT(g_word1);
```

  - Function with EN/ENO (NOT_E)
    [Structured ladder/FBD]

```
[ST]
g_bool2 := NOT_E(g_bool1, g_word1, g_word2);
```
5.5 Standard Selection Functions

Selection

SEL(_E)

The following function(s) can go in the dotted squares.

SEL, SEL_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_G)</td>
<td>Output condition (TRUE: s3 output, FALSE: s2 output)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s2(_IN0)</td>
<td>Input</td>
<td>ANY</td>
</tr>
<tr>
<td></td>
<td>s3(_IN1)</td>
<td>Input</td>
<td>ANY</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- Selects either of values input to (s2) and (s3) according to the bit type data input to (s1), and outputs the operation result from (d) in the same data type as that of (s2) and (s3).
- When the input value of (s1) is FALSE, the value input to (s2) is output from (d).
- When the input value of (s1) is TRUE, the value input to (s3) is output from (d).

**Example**

(s2) and (s3) are word (signed) type data

- The input value to (s1) is data value of bit type.
- The input value to (s2), (s3) is data value of bit type/word (signed) type/double word (signed) type/word (unsigned) type/16-bit string 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.
- Rounding error may occur when specifying single-precision real or double-precision real type data to (s2), (s3) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

[ MELSEC-Q/L/F Structured Programming Manual (Fundamentals) ]
5.5 Standard Selection Functions

■ Operation result

- Function without EN/ENO

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

- 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 (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 (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.

■ Program example

■ SEL(_E)

The program which selects either of values input to (s2) and (s3) according to the value input to (s1), and outputs the operation result from (d) in the same data type as that of (s2) and (s3).

- Function without EN/ENO (SEL)

[Structured ladder/FBD]

```
1

1

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

[ST]

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

- Function with EN/ENO (SEL_E)

[Structured ladder/FBD]

```
2

2

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

[ST]

```
g_bool3 := SEL_E(g_bool1, g_bool2, g_word1, g_word2, g_word3);
```
Maximum/Minimum selection

MAXIMUM(E), MINIMUM(E)

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s1 to s28(_IN)</td>
<td></td>
<td>Input</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td></td>
<td>Output</td>
<td>ANY_SIMPLE</td>
</tr>
</tbody>
</table>

### Operation details

#### Maximum processing
- **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 (s1) to (s28), and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

**Example**

Word (signed) type data

```
(s1) ... (s28)
  1234
 Word (signed) type
  5678
 Word (signed) type
```

#### Minimum processing
- **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 (s28), and outputs the operation result from (d) in the same data type as that of (s1) to (s28).

**Example**

Word (signed) type data

```
(s1) ... (s28)
  1234
 Word (signed) type
  5678
 Word (signed) type
```
• The values to be input to (s1) to (s28) 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.
• Rounding error may occur when specifying single-precision real or double-precision real type data to (s1) through (s28) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

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

• The number of pins of (s) can be changed in the range from 2 to 28.
• If word (unsigned) type/16-bit string type/double word (unsigned) type/32-bit string type is specified for (d), warning C9026 occurs.

• Conditions for comparing the STRING data type values are as follows:

  Match:
  • All characters matched
  Bigger string:
  • The one having a character with a bigger code (when strings consist of different characters)
  • The one having a longer length (when strings are of different lengths)
  Smaller string:
  • The one having a character with a smaller code (when strings consist of different characters)
  • The one having a shorter length (when strings are of different lengths)

### Operation result

- Function without EN/ENO
  An operation is executed and the operation value is output from (d).
- 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 (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 (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.
5 APPLICATION FUNCTIONS
5.5 Standard Selection Functions

Program example

**MAXIMUM(_E)**
The program which outputs the maximum value of the word (signed) data input to variables (s1) to (s28) from (d) in the same data type as that of (s1) to (s28).

- Function without EN/ENO (MAXIMUM)

```
[Structured ladder/FBD]

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

- Function with EN/ENO (MAXIMUM_E)

```
[Structured ladder/FBD]

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

**MINIMUM(_E)**
The program which outputs the minimum value of the word (signed) data input to variables (s1) to (s28) from (d) in the same data type as that of (s1) to (s28).

- Function without EN/ENO (MINIMUM)

```
[Structured ladder/FBD]

ST g_int3 := MINIMUM(g_int1, g_int2);
```
**Upper/Lower limit control**

**LIMITATION(_E)**

- **Basic**
- **Process**
- **Redundant**
- **Universal**
- **LCPU**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s1</td>
<td>d</td>
</tr>
<tr>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td></td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.
LIMITATION, LIMITATION_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s1(_MN)</td>
<td>Lower limit value (minimum output limit value)</td>
<td>ANY_SIMPLE</td>
<td></td>
</tr>
<tr>
<td>s2(_IN)</td>
<td>Input controlled by the upper/lower limit control</td>
<td>ANY_SIMPLE</td>
<td></td>
</tr>
<tr>
<td>s3(_MX)</td>
<td>Upper limit value (maximum output limit value)</td>
<td>ANY_SIMPLE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>ANY_SIMPLE</td>
<td></td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- 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 (s1), (s2), and (s3) according to their values, and outputs the operation result from (d) in the same data type as that of (s1) to (s3).

When the input value of (s2) > the input value of (s3), outputs the input value (s3) from (d).

When the input value of (s2) < the input value of (s1), outputs the input value (s1) from (d).

When the input value of (s1) ≤ the input value of (s2) ≤ the input value of (s3), outputs the input value of (s2) from (d).

**Ex**

Word (signed) type data

- The values to be input to (s1), (s2), and (s3) 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 (s1) < the input value of (s3))

- Rounding error may occur when specifying single-precision real or double-precision real type data to (s1), (s2), or (s3) by programming tool.
For precautions when setting an input value using a programming tool, refer to the following.

- If word (unsigned) type/16-bit string type/double word (unsigned) type/32-bit string type is specified for (d), warning C9026 occurs at compilation.

- Conditions for comparing the STRING data type values are as follows:
  
  **Match:**
  - All characters matched
  
  **Bigger string:**
  - The one having a character with a bigger code (when strings consist of different characters)
  - The one having a longer length (when strings are of different lengths)

  **Smaller string:**
  - The one having a character with a smaller code (when strings consist of different characters)
  - The one having a shorter length (when strings are of different lengths)

### Operation result

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

- 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 (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 (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.

### Program example

#### LIMITATION(_E)

The program which outputs the values input to variables (s1), (s2), and (s3) according to the word (signed) data from (d) in the same data type as that of (s1), (s2), and (s3).

- Function without EN/ENO (LIMITATION)
  
  [Structured ladder/FBD]

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

- Function with EN/ENO (LIMITATION_E)
  
  [Structured ladder/FBD]

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

**MUX(_E)**

The following function(s) can go in the dotted squares.

MUX, MUX_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n(_K)</td>
<td>Output value selection</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>s1 to s28(_IN)</td>
<td>Input</td>
<td>ANY</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>ANY</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- Selects the value to be output among the values input to variables (s1) to (s28) according to the value input to n, and outputs the operation result from (d) in the same data type as that of variables (s1) to (s28).

When the input value of n is 1, the value input to (s1) is output from (d).
When the input value of n is n, the value input to (sn) is output from (d).

#### Example

Word (signed) type data

- 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.)

- 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')

- 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.

- Rounding error may occur when specifying single-precision real or double-precision real type data to (s1) through (s28) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

- MELSEC-Q/L/F Structured Programming Manual (Fundamentals)
- The number of pins of (s) can be changed in the range from 2 to 28.
### Operation result

- **Function without EN/ENO**

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

- **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 (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 (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.

### Program example

#### MUX(_E)

The program which selects the value to be output among the values input to variables (s1) and (s2) according to the value input to n, and outputs the operation result from (d) in the same data type as that of (s1) or (s2).

- **Function without EN/ENO (MUX)**

[Structured ladder/FBD]

```plaintext
MUX(g_int1, g_int2, g_int3); => g_int4
```

- **Function with EN/ENO (MUX_E)**

[Structured ladder/FBD]

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

5.6 Standard Comparison Functions

Comparison

GT_E, GE_E, EQ_E, LE_E, LT_E, NE_E

The following function(s) can go in the dotted squares.

GT_E, GE_E, EQ_E, LE_E, LT_E, NE_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1 to s28(_IN)</td>
<td>(s1 and s2 only for NE_E)</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output (TRUE: True value, FALSE: False value)</td>
<td>Bit</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- Performs comparison operation between the values input to variables (s1) to (s28), and outputs the operation result from (d) in bit type as that of variables (s1) to (s28).

**Operation**

- **GT_E**
  - Performs comparison of \[(s1) > (s2) \& \& \& \& (s2) > (s3) \& \& \& \& \& (s(n-1)) > (s(n))\].
  - Outputs TRUE if all comparisons satisfy \((s)_{n-1} > (s)n\).
  - Outputs FALSE if any of comparisons satisfies \((s)_{n-1} \leq (s)n\).

- **GE_E**
  - Performs comparison of \[(s1) \geq (s2) \& \& \& \& (s2) \geq (s3) \& \& \& \& \& (s(n-1)) \geq (s(n))\].
  - Outputs TRUE if all comparisons satisfy \((s)_{n-1} \geq (s)n\).
  - Outputs FALSE if any of comparisons satisfies \((s)_{n-1} < (s)n\).

- **EQ_E**
  - Performs comparison of \[(s1) = (s2) \& \& \& \& (s2) = (s3) \& \& \& \& \& \& (s(n-1)) = (s(n))\].
  - Outputs TRUE if all comparisons satisfy \((s)_{n-1} = (s)n\).
  - Outputs FALSE if any of comparisons satisfies \((s)_{n-1} \neq (s)n\).

- **LE_E**
  - Performs comparison of \[(s1) \leq (s2) \& \& \& \& (s2) \leq (s3) \& \& \& \& \& \& (s(n-1)) \leq (s(n))\].
  - Outputs TRUE if all comparisons satisfy \((s)_{n-1} \leq (s)n\).
  - Outputs FALSE if any of comparisons satisfies \((s)_{n-1} > (s)n\).

- **LT_E**
  - Performs comparison of \[(s1) < (s2) \& \& \& \& (s2) < (s3) \& \& \& \& \& \& (s(n-1)) < (s(n))\].
  - Outputs TRUE if all comparisons satisfy \((s)_{n-1} < (s)n\).
  - Outputs FALSE if any of comparisons satisfies \((s)_{n-1} \geq (s)n\).

- **NE_E**
  - Performs comparison of \[(s1) \neq (s2)\].
  - Outputs TRUE if \((s1) \neq (s2)\).
  - Outputs FALSE if \((s1) = (s2)\).
• The values to be input to (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 type data.
• Rounding error may occur when specifying single-precision real or double-precision real type data to (s1) through (s28) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.
MELSEC-Q/L/F Structured Programming Manual (Fundamentals)
• The number of pins of (s) can be changed in the range from 2 to 28. (The number of pins of (s) for comparison operator NE(_E)) is fixed at (s1) and (s2).
• Conditions for comparing the STRING data type values are as follows:
  Match: • All characters matched
  Bigger string: • The one having a character with a bigger code (when strings consist of different characters)
  • The one having a longer length (when strings are of different lengths)
  Smaller string: • The one having a character with a smaller code (when strings consist of different characters)
  • The one having a shorter length (when strings are of different lengths)

### Operation result
• Function without EN/ENO
An operation is executed and the operation value is output from (d).
• 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 (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 (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.

### Program example
**GT_E**
The program which performs comparison operation between the values input to (s1) and (s2), and outputs the operation result from (d).

[Structured ladder/FBD]

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

[ST]
5.7 Standard Character String Functions

Extract mid string

MID(_E)

The following function(s) can go in the dotted squares.

MID, MID_E

■ Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_IN)</td>
<td>Input</td>
<td>String (255)</td>
</tr>
<tr>
<td></td>
<td>n1(_L)</td>
<td>Number of characters to be extracted</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>n2(_P)</td>
<td>Start position to be extracted</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String (255)</td>
</tr>
</tbody>
</table>

■ Processing details

■ Operation processing

• Extracts the specified number of characters from the specified start position in the character string input to (s), and outputs the operation result from (d). The number of characters to be extracted is specified by the value input to n1. The start position of the characters to be extracted is specified by the value input to n2.

Ex

Values input to n1 and n2 are 5

<table>
<thead>
<tr>
<th></th>
<th>'ABCDEF12345'</th>
<th>'EF123'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>42H(B) : 41H(A)</td>
<td></td>
</tr>
<tr>
<td>2nd word</td>
<td>44H(D) : 43H(C)</td>
<td></td>
</tr>
<tr>
<td>3rd word</td>
<td>46H(F) : 45H(E)</td>
<td></td>
</tr>
<tr>
<td>4th word</td>
<td>32H(2) : 31H(1)</td>
<td></td>
</tr>
<tr>
<td>5th word</td>
<td>34H(4) : 33H(3)</td>
<td></td>
</tr>
<tr>
<td>6th word</td>
<td>00H : 35H(5)</td>
<td>Number of characters to be extracted n1: 5 characters</td>
</tr>
</tbody>
</table>

• The value to be input to (s) is string type data within the range from 0 to 255 bytes.
• 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 (s).)
• 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 (s).)
### Operation result

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

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

### Operation error

These functions consist of the following instructions.

**MID(E): MIDR**

- 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.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/ Q00/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4101</td>
<td>The value of n2 exceeds the number of characters specified for (s). The number of characters from the start position of (d) to n1 exceeds the device range of (d). The value of n2 is 0. &quot;00H&quot; does not exist at the device number or later specified by (s) in the range of the corresponding device.</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Program example

**MID(E)**

The program which extracts the specified number of characters from the specified start position in the character string input to (s), and outputs the operation result from (d).

- **Function without EN/ENO (MID)**

  [Structured ladder/FBD]

  ```plaintext
  [ST]
  g_string2 := MID(g_string1, g_int1, g_int2);
  ```

- **Function with EN/ENO (MID_E)**

  [Structured ladder/FBD]

  ```plaintext
  [ST]
  g_bool3 := MID_E(g_bool1, g_string1, g_int1, g_int2, g_string2);
  ```
String concatenation

**CONCAT(_E)**

The following function(s) can go in the dotted squares.
CONCAT, CONCAT_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1 to s28(_IN)</td>
<td>Input</td>
<td>String (255)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String (255)</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing
- Concatenates the character string input to (s2) to (s28) following the one input to (s1), and outputs the operation result from (d).

This function concatenates character string (s2) to (s28) with ignoring '00H', which indicates the end of character string (s1). If the concatenated character string has over 255 bytes, the character string up to 255 bytes is output.

- The values to be input to (s1) and (s2) to (s28) are string type data within the range from 0 to 255 bytes.
- The number of pins of (s) can be changed in the range from 2 to 28.

![Structured ladder/FBD](image)

#### Operation result
- Function without EN/ENO
  - An operation is executed and the operation value is output from (d).
- 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 (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 (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.

Program example

CONCAT(_E)
The program which concatenates the character string input to (s2) following the one input to (s1), and outputs the operation result from (d).

- Function without EN/ENO (CONCAT)

[Structured ladder/FBD]

```
g_string3 := CONCAT(g_string1, g_string2);
```

- Function with EN/ENO (CONCAT_E)

[Structured ladder/FBD]

```
g_bool3 := CONCAT_E(g_bool1, g_string1, g_string2, g_string3);
```
## String insertion

**INSERT(_E)**

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_IN1)</td>
<td>Input</td>
<td>String (255)</td>
</tr>
<tr>
<td></td>
<td>s2(_IN2)</td>
<td>Input</td>
<td>String (255)</td>
</tr>
<tr>
<td></td>
<td>n(_P)</td>
<td>Start position to be inserted</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String (255)</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- Inserts the character string input to (s2) to the specified position in the character string input to (s1), 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 (s2) to character string (s1), '00H' that indicates the end of character string (s2) is ignored. If the character string after insertion has over 255 bytes, the character string up to 255 bytes is output.

**Ex**

Value input to n is 4

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>Start position to be inserted n: 4th character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>42H(B)</td>
<td>41H(A)</td>
</tr>
<tr>
<td>2nd word</td>
<td>44H(D)</td>
<td>43H(C)</td>
</tr>
<tr>
<td>3rd word</td>
<td>00H</td>
<td>45H(E)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>Output value</th>
<th>‘ABC123456DE’</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>42H(B)</td>
<td>41H(A)</td>
<td>1st word</td>
</tr>
<tr>
<td>2nd word</td>
<td>31H(1)</td>
<td>43H(C)</td>
<td>2nd word</td>
</tr>
<tr>
<td>3rd word</td>
<td>33H(3)</td>
<td>32H(2)</td>
<td>3rd word</td>
</tr>
<tr>
<td>4th word</td>
<td>35H(5)</td>
<td>34H(4)</td>
<td>4th word</td>
</tr>
<tr>
<td>5th word</td>
<td>44H(D)</td>
<td>36H(6)</td>
<td>5th word</td>
</tr>
<tr>
<td>6th word</td>
<td>00H</td>
<td>45H(E)</td>
<td>6th word</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High-order byte</th>
<th>Low-order byte</th>
<th>Input value to 1</th>
<th>‘ABCDE’</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st word</td>
<td>32H(2)</td>
<td>31H(1)</td>
<td>1st word</td>
</tr>
<tr>
<td>2nd word</td>
<td>34H(4)</td>
<td>33H(3)</td>
<td>2nd word</td>
</tr>
<tr>
<td>3rd word</td>
<td>36H(6)</td>
<td>35H(5)</td>
<td>3rd word</td>
</tr>
<tr>
<td>4th word</td>
<td>00H</td>
<td></td>
<td>4th word</td>
</tr>
</tbody>
</table>
• The values to be input to (s1) and (s2) are string type data within the range from 0 to 255 bytes.
• 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 input variable (s1).)

### Operation result

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

### Program example

#### INSERT(_E)
The program which inserts the character string input to (s2) to the specified position in the character string input to (s1), and outputs the operation result from (d).
- Function without EN/ENO (INSERT)

[Structured ladder/FBD]

```
[ST]
g_string3 := INSERT(g_string1, g_string2, g_int1);
```

- Function with EN/ENO (INSERT_E)

[Structured ladder/FBD]

```
[ST]
g_bool3 := INSERT_E(g_bool1, g_string1, g_string2, g_int1, g_string3);
```
5 APPLICATION FUNCTIONS

5.7 Standard Character String Functions

String deletion

DELETE(_E)

The following function(s) can go in the dotted squares.
DELETE, DELETE_E

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_IN)</td>
<td>Input String (255)</td>
<td>String (255)</td>
</tr>
<tr>
<td></td>
<td>n1(_L)</td>
<td>Number of characters to be deleted</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>n2(_P)</td>
<td>Start position to be deleted</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output String (255)</td>
<td>String (255)</td>
</tr>
</tbody>
</table>

Processing details

Operation processing

- 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.

Ex

Values input to n1 and n2 are 5

```
'ABCDEF12345'  'ABCD12345'
```

<table>
<thead>
<tr>
<th>1st word</th>
<th>2nd word</th>
<th>3rd word</th>
<th>4th word</th>
<th>5th word</th>
<th>6th word</th>
</tr>
</thead>
<tbody>
<tr>
<td>42h(B)</td>
<td>44h(D)</td>
<td>46h(F)</td>
<td>32h(2)</td>
<td>34h(4)</td>
<td>00h</td>
</tr>
<tr>
<td>41h(A)</td>
<td>43h(C)</td>
<td>45h(E)</td>
<td>31h(1)</td>
<td>33h(3)</td>
<td>35h(5)</td>
</tr>
</tbody>
</table>

- The value to be input to (s) is string type data within the range from 0 to 255 bytes.
- The value to be input to n1 is word (signed) type data within the range from 0 to 255.
The value input must not exceed the number of characters of character string input to (s).
- The value to be input to n2 is word (signed) type data within the range from 1 to 255.
The value input must not exceed the number of characters of character string input to (s).
5 APPLICATION FUNCTIONS
5.7 Standard Character String Functions

Operation result

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

Program example

DELETE(_E)

The program which 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).

- Function without EN/ENO (DELETE)

  [Structured ladder/FBD]

  [ST]
  g_string2 := DELETE(g_string1, g_int1, g_int2);

- Function with EN/ENO (DELETE_E)

  [Structured ladder/FBD]

  [ST]
  g_bool3 := DELETE_E(g_bool1, g_string1, g_int1, g_int2, g_string2);
# String replacement

## REPLACE(_E)

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s1</td>
<td>=&gt;</td>
</tr>
<tr>
<td>n1</td>
<td>=&gt;</td>
</tr>
<tr>
<td>n2</td>
<td>=&gt;</td>
</tr>
<tr>
<td>d</td>
<td>=&gt;</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.

REPLACE, REPLACE_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_IN1)</td>
<td>Input</td>
<td>String (255)</td>
</tr>
<tr>
<td></td>
<td>s2(_IN2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n1(_L)</td>
<td>Number of characters to be replaced</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>n2(_P)</td>
<td>Start position to be replaced</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>String (255)</td>
</tr>
</tbody>
</table>
### Operation processing

- Replaces the specified number of characters from the specified position in the character string input to \((s1)\) with the character string input to \((s2)\), and outputs the operation result from \((d)\).

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 \(n2\).

**Ex.**

Values input to \(n1\) and \(n2\) are 5

<table>
<thead>
<tr>
<th>Input value to %(s1)%</th>
<th>Output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ABCDEFGH123'</td>
<td>'ABCD1234523'</td>
</tr>
</tbody>
</table>

```
High-order byte | Low-order byte
1st word       | 42H(B) : 41H(A) |
2nd word       | 44H(D) : 43H(C) |
3rd word       | 46H(F) : 45H(E) |
4th word       | 48H(H) : 47H(G) |
5th word       | 32H(2) : 31H(1) |
6th word       | 00H : 33H(3)   |
```

**Start position to be replaced n2: 5th character**

Number of characters to be replaced n1: 5 characters

```
High-order byte | Low-order byte
1st word       | 42H(B) : 41H(A) |
2nd word       | 44H(D) : 43H(C) |
3rd word       | 32H(2) : 31H(1) |
4th word       | 00H : 33H(3)   |
```

- The values to be input to \((s1)\) and \((s2)\) are string type data within the range from 0 to 255 bytes.
- 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 \((s1)\).)
- 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 input variable \((s1)\).)

### Operation result

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

- 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 (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 \((d)\) is undefined. In this case, create a program so that the data output from \((d)\) is not used.

### Operation error

- No operation error occurs.
- REPLACE(_E)

The program which replaces the specified number of characters from the specified position in the character string input to (s1) with the character string input to (s2), and outputs the operation result from (d).

- Function without EN/ENO (REPLACE)

[Structured ladder/FBD]

Program example

```plaintext
■REPLACE(_E)

The program which replaces the specified number of characters from the specified position in the character string input to (s1) with the character string input to (s2), and outputs the operation result from (d).

- Function without EN/ENO (REPLACE)

[Structured ladder/FBD]

Program example

```

```plaintext
• Function with EN/ENO (REPLACE_E)

[Structured ladder/FBD]

Program example

```
5.8 Functions of Time Data Type

Addition

ADD_TIME(_E)

The following function(s) can go in the dotted squares.
ADD_TIME, ADD_TIME_E

■ Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_IN1)</td>
<td>Input</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>s2(_IN2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Time</td>
</tr>
</tbody>
</table>

■ Processing details

■ Operation processing
- Performs addition ((s1) + (s2)) on time type data input to (s1) and (s2), and outputs the operation result from (d) in time type.

**Ex**
When the input value to (s1) and (s2) are T#1d2h33m44s55ms (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and T#2ms (2 milliseconds).

\[ 1 \quad \boxed{T#1d2h33m44s55ms} + \boxed{T#2ms} \rightarrow \boxed{T#1d2h33m44s57ms} \]

- The value to be input to (s1), (s2) are time type data.
- No operation error occurs even if an underflow/overflow occurs. Data is output from (d) as follows.

In case of ADD_TIME_E, TRUE is output from ENO.

**Ex**
Overflow

\[ \boxed{T#24d20h31m23s647m} + \boxed{T#2ms} \rightarrow \boxed{T#-24d20h31m23s647ms} \]

Since the highest-order bit is 1, the result value is negative.
**Example**

**Underflow**

\[ \text{T#24d20h31m23s648ms} + \text{T#-2ms} \rightarrow \text{T#24d20h31m23s646ms} \]  
\[(80000000h) + (FFFFFFEh) \rightarrow (7FFFDFFFEh)\]

Since the highest-order bit is 0, the result value is positive.

### Operation result

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

- **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 (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 (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.

### Program example

**ADD_TIME(_E)**

The program which performs addition ((s1) + (s2)) on time type data input to (s1) and (s2), and outputs the operation result from (d) in time type.

- **Function without EN/ENO (ADD_TIME)**
  
  [Structured ladder/FBD]

  \[
  g_{\text{time}3} := \text{ADD\_TIME}(g_{\text{time}1}, g_{\text{time}2});
  \]

- **Function with EN/ENO (ADD\_TIME\_E)**
  
  [Structured ladder/FBD]

  \[
  g_{\text{bool}3} := \text{ADD\_TIME\_E}(g_{\text{bool}1}, g_{\text{time}1}, g_{\text{time}2}, g_{\text{time}3});
  \]
Subtraction

SUB_TIME(_E)

Structured ladder/FBD

Input/output argument | Name | Description | Data type |
--- | --- | --- | --- |
Input argument | EN | Executing condition (TRUE: Execution, FALSE: Stop) | Bit |
 | s1(_IN1) | Input | Time |
 | s2(_IN2) | Input | Time |
Output argument | ENO | Execution result (TRUE: Normal, FALSE: Error) | Bit |
 | d | Output | Time |

Processing details

Operation processing

- Performs subtraction ((s1) - (s2)) on time type data input to (s1) and (s2), and outputs the operation result from (d) in time type.

Ex)

When the input value to (s1) and (s2) are T#1d2h33m44s55ms (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and T#2ms (2 milliseconds).

- The value to be input to (s1), (s2) are time type data.
- No operation error occurs even if an underflow/overflow occurs. Data is output from (d) as follows.

In case of SUB_TIME_E, TRUE is output from ENO.

Ex)

Overflow

Since the highest-order bit is 1, the result value is negative.

Ex)

Underflow

Since the highest-order bit is 0, the result value is positive.
### Operation result

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

- **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 (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 (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.

### Program example

#### SUB_TIME(E)

The program which performs subtraction ((s1) - (s2)) on time type data input to (s1) and (s2), and outputs the operation result from (d) in time type.

- **Function without EN/ENO (SUB_TIME)**

  [Structured ladder/FBD]

  ```
  g_time3 := SUB_TIME(g_time1, g_time2);
  ```

- **Function with EN/ENO (SUB_TIME_E)**

  [Structured ladder/FBD]

  ```
  g_bool3 := SUB_TIME_E(g_bool1, g_time1, g_time2, g_time3);
  ```
## Multiplication

**MUL_TIME(E)**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN ENO s1 d s2</td>
<td>ENO=MUL_TIME_E[EN, s1, s2, d];</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.

**MUL_TIME, MUL_TIME_E**

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s1(_IN1)</td>
<td>Input</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>s2(_IN2)</td>
<td>Input</td>
<td>ANY_NUM</td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td>d</td>
<td>Output</td>
<td>Time</td>
<td></td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

- Performs multiplication \((s1 \times s2)\) on time type data input to \((s1)\) and the word (signed), double word (signed), single-precision real or double-precision real type data input to \((s2)\), and outputs the operation result from \((d)\) in time type.

#### Ex

When the input value to \((s1)\) and \((s2)\) are \(T\#1d2h33m44s55ms\) (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and 2.

\[ \begin{align*} 
T\#1d2h33m44s55ms & \times \; \begin{array}{|c|} \hline \text{2} \; \text{Word (signed)} \; \text{Time type} \; \hline \end{array} \\
\rightarrow \; \text{Time type} & 
\end{align*} \]

- The value to be input to \((s1)\) is time type data.
- The value to be input to \((s2)\) is word (signed), double word (signed), single-precision real or double-precision real type data.
- Rounding error may occur when specifying single-precision real or double-precision real type data to \((s2)\) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.

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

No operation error occurs even if an underflow/overflow occurs. Data is output from \((d)\) 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.)

#### Ex

**Overflow**

\[ \begin{align*} 
T\#24d20h31m23s647ms & \times \; \begin{array}{|c|} \hline \text{2} \; \text{(00000002)} \; \text{Double word (signed)} \; \hline \end{array} \\
\rightarrow \; \text{Time type} & 
\end{align*} \]

Since the highest-order bit is 1, the result value is negative.
Underflow

Ex. Since the highest-order bit is 0, the result value is positive.

- **Operation result**
  - Function without EN/ENO:
    An operation is executed and the operation value is output from (d).
  - 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 (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 (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.

- **Program example**

**MUL_TIME(E)**

The program which performs multiplication ((s1) × (s2)) on time type data input to (s1) and the word (signed) type data input to (s2), and outputs the operation result from (d) in time type.

- Function without EN/ENO (MUL_TIME)
  - [Structured ladder/FBD]
  - [ST]
    
    g_time2 := MUL_TIME(g_time1, g_int1);

- Function with EN/ENO (MUL_TIME_E)
  - [Structured ladder/FBD]
  - [ST]
    
    g_bool3 := MUL_TIME_E(g_bool1, g_time1, g_int1, g_time2);
Division

DIV_TIME(_E)

The following function(s) can go in the dotted squares.
DIV_TIME, DIV_TIME_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_IN1)</td>
<td>Input</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>s2(_IN2)</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Time</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**
- Performs division \((s1) \div (s2))\) on time type data input to \((s1)\) and the word (signed), double word (signed), single-precision real or double-precision real type data input to \((s2)\), and outputs the quotient of the operation result from \((d)\) in time type. Remainder is rounded down. Remainder is rounded down.

**Ex**

When the input value to \((s1)\) and \((s2)\) are T#1d2h33m44s55ms (1 day 2 hours 33 minutes 44 seconds 55 milliseconds) and 2.

1. T#1d2h33m44s55ms → 2 → (Quotient) (Remainder)
   - Time type → Word (signed) → Time type
   - Discarded
   - T#13h16m63s27ms, T#1ms

- The value to be input to \((s1)\) is time type data.
- The value to be input to \((s2)\) is word (signed), double word (signed), single-precision real or double-precision real type data. (The value to be input to \((s2)\) must be other than 0.)
- Rounding error may occur when specifying single-precision real or double-precision real type data to \((s2)\) by programming tool.

For precautions when setting an input value using a programming tool, refer to the following.
- MELSEC-Q/L/F Structured Programming Manual (Fundamentals)
### Operation result

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

- **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 (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 (d) is undefined. In this case, create a program so that the data output from (d) is not used.

### Operation error

- An operation error occurs in the following case.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J</th>
<th>Q00/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The value to be input to (s2) is 0. (Division by 0)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
</tr>
</tbody>
</table>

### Program example

#### DIV_TIME(E)

The program which performs division \((s1) \div (s2)\) on time type data input to (s1) and the word (signed) type data input to (s2), and outputs the quotient of the operation result from (d) in time type.

- **Function without EN/ENO (DIV_TIME)**
  
  [Structured ladder/FBD]

  ![](image1.png)

  
  \[ST\]
  
  \[
g_{\text{time}2} := \text{DIV}_{\text{TIME}}(g_{\text{time}1}, g_{\text{int}1});
  \]

- **Function with EN/ENO (DIV_TIME_E)**
  
  [Structured ladder/FBD]

  ![](image2.png)

  
  \[ST\]
  
  \[
g_{\text{bool}3} := \text{DIV}_{\text{TIME}_E}(g_{\text{bool}1}, g_{\text{time}1}, g_{\text{int}1}, g_{\text{time}2});
  \]
5.9 Standard Bistable Function Blocks

Standard bistable function blocks (Set-dominant)

**SR(_E)**

The following function(s) can go in the dotted squares.

SR, SR_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

Sets (d) when (s1) is turned ON and resets (d) when (s2) is turned ON while (s1) is OFF. (d) is not reset even when (s2) is turned ON while (s1) is ON.

#### Operation result

- Function without EN/ENO

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

[Timing chart]
• 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 (Op execution)</td>
<td>TRUE</td>
<td>Operation output value</td>
</tr>
<tr>
<td>FALSE (Op stop)</td>
<td>FALSE</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]

- No operation error occurs.

Program example

**SR(_E)**

The program which outputs bit type data input to (s1) from (d) and holds the output, and resets the value of (d) only when bit type data input to (s2) is 1 and the data input to (s1) is 0.

- Function without EN/ENO (SR)

  [Structured ladder/FBD]

  ```plaintext
  SR_inst(_S1:=g_bool1, RESET:=g_bool2, Q1:=g_bool3);
  ```

- Function with EN/ENO (SR_E)

  [Structured ladder/FBD]

  ```plaintext
  SR_E_inst(EN:= X0, _S1:=g_bool1, _R:=g_bool2, Q1:=g_bool3, ENO:=Y20);
  ```
## Standard bistable function blocks (Reset-dominant)

### RS(_E)

The following function(s) can go in the dotted squares.

RS, RS_E

#### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(_S)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s2(_R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d(Q1)</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

#### Processing details

**Operation processing**

Sets (d) when (s1) is turned ON, and resets (d) when (s2) is turned ON. (d) is not set even when (s1) is turned ON while (s2) is ON.

**Operation result**

- Function without EN/ENO

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

[Timing chart]

---

Turns ON when \( s1 = \text{ON} \) and \( s2 = \text{OFF} \).

Turns OFF when \( s2 = \text{ON} \).

---

### Structured ladder/FBD

ST

[Structured ladder/FBD diagram]
• 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 (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>

**Operation error**

• No operation error occurs.

**Program example**

**RS(\_E)**

The program which outputs bit type data input to (s1) from (d) and holds the output, and resets forcibly the value of (d) when bit type data input to (s2) is 1.

• Function without EN/ENO (RS)

[Structured ladder/FBD]

```
RS_inst(_S:=g_bool1, _R1:=g_bool2, Q1:=g_bool3);
```

• Function with EN/ENO (RS_E)

[Structured ladder/FBD]

```
RS_E_inst(EN:=X0, _S:=g_bool1, _R1:=g_bool2, Q1:=g_bool3, ENO:=Y20);
```
5.10 Standard Edge Detection Function Blocks

Rising edge detector

R_TRIG(_E)

The following function(s) can go in the dotted squares.
R_TRIG, R_TRIG_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_CLK)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d(Q)</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**
Turns ON (d) for one scan when (s) is turned ON.

**Operation result**
• Function without EN/ENO

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

[Timing chart]

5.10 Standard Edge Detection Function Blocks
• 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</td>
<td>Previous output value</td>
</tr>
</tbody>
</table>

[Timing chart]

Turns (①) ON when EN = ON and the rising of ③.

Turns (②) OFF at the next scan. When EN = OFF, ③ retains the previous scan output result.

Operation error

• No operation error occurs.

Program example

R_TRIG(E)
The program which turns ON (d) for one scan when bit type data input to (s) is turned from OFF to ON.

• Function without EN/ENO (R_TRIG)

[Structured ladder/FBD]

R_TRIG_Inst(_CLK:=g_bool1, Q:=g_bool2);

• Function with EN/ENO (R_TRIG_E)

[Structured ladder/FBD]

R_TRIG_E_Inst(EN:=X0, _CLK:=g_bool1, Q:=g_bool2, ENO:=Y20);
Falling edge detector

F_TRIG(_E)

Arguments

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(_CLK)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d(Q)</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Processing details

Operation processing

Turns ON (d) for one scan when (s) is turned OFF.

Operation result

- Function without EN/ENO

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

Timing chart

Turns (3) ON at the falling of (5).

Turns (3) OFF at the next scan.
• 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 (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]

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

Turns (2) ON when EN = ON and the falling of (5).

Turns (3) OFF at the next scan.

When EN = OFF, (4) retains the previous scan output result.

Operation error

• No operation error occurs.

Program example

**F_TRUNC(E)**

The program which turns ON (d) for one scan when bit type data input to (s) is turned from ON to OFF.

• Function without EN/ENO (F_TRUNC)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

  [ST]
  F_TRUNC.InstCLK:=g Bool1, Q:=g Bool2;

• Function with EN/ENO (F_TRUNC_E)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

  [ST]
  F_TRUNC_E.InstEN:=X0, _CLK:=g Bool1, Q:=g Bool2, ENO:=Y20;
5.11 Standard Counter Function Blocks

Up counter

CTU(_E)

The following function(s) can go in the dotted squares.
CTU, CTU_E

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(CU)</td>
<td>Count signal input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s2(RESET)</td>
<td>Count reset</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n(PV)</td>
<td>Maximum count value</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d1(Q)</td>
<td>Count match output</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d2(CV)</td>
<td>Count value</td>
<td>Word (signed)</td>
</tr>
</tbody>
</table>

Processing details

Operation processing
- Counts (d2) when (s1) is turned ON. When the count value (d2) reaches the value input to n, (d1) turns ON. When (s2) is turned ON, (d1) turns OFF and count value (d2) is reset.
- Valid setting range for n is -32768 to 32767. However, if 0 or less is set, (d1) is turned on regardless of the count value of (d2).

Operation result
- Function without EN/ENO
An operation is executed and the operation value is output from (d1) and (d2).

[Timing chart]
When n=3

Counts up when = ON. Clears 0 when = ON.
• 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 (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

- Operation error
  - No operation error occurs.

- Program example

**CTU(E)**

The program which counts the number of times that bit type data input to (s1) is turned from OFF to ON, and outputs the count value from (d2).

- 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);
```

- 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);
```
Down counter

**CTD(E)**

- **Structured ladder/FBD**
- **ST**

The following function(s) can go in the dotted squares.
CTD, CTD_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(CD)</td>
<td>Count signal input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s2(LOAD)</td>
<td>Count reset</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n(PV)</td>
<td>Count start value</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d1(Q)</td>
<td>Count match output</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d2(CV)</td>
<td>Count value</td>
<td>Word (signed)</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing
- Counts down (-1) (d2) when (s1) is turned ON. n sets the initial value for subtraction. (d1) turns ON when count value (d2) reaches 0. When (s2) is turned ON, (d1) turns OFF and initial value for subtraction n is set for count value (d2).
- Valid setting range for n is -32768 to 32767. However, if 0 or less is set, (d1) is turned on regardless of the count value of (d2).

#### Operation result
- Function without EN/ENO
An operation is executed and the operation value is output from (d1) and (d2).

[Timing chart]

When n=3

Initializes when = ON. Counts down at the falling of .
**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>

**Operation error**

- No operation error occurs.

**Program example**

**CTD(E)**

The program which counts the number of times that bit type data input to (s1) is turned from OFF to ON, and turns ON (d1) when the value of (d2) reaches 0.

- **Function without EN/ENO (CTD)**
  
  [Structured ladder/FBD]

  ```plaintext
  CTD_Cnt(CD:=g_bool1, LOAD:=g_bool2, PV:=g_int1, Q:=g_bool3, CV:=g_int2 );
  ```

- **Function with EN/ENO (CTD_E)**
  
  [Structured ladder/FBD]

  ```plaintext
  CTD_EInst (EN:=M10, CD:=g_bool1, LOAD:=g_bool2, PV:=g_int1, Q:=g_bool3, CV:=g_int2, ENO:=M11);
  ```
Up/Down counter

CTUD(_E)

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>ENO</td>
</tr>
<tr>
<td>s1(CU)</td>
<td>d1</td>
</tr>
<tr>
<td>s2(CD)</td>
<td>d2</td>
</tr>
<tr>
<td>s3(RESET)</td>
<td>d3</td>
</tr>
<tr>
<td>s4(LOAD)</td>
<td>n</td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.
CTUD, CTUD_E

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s1(CU)</td>
<td>Count-up signal input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s2(CD)</td>
<td>Count-down signal input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s3(RESET)</td>
<td>Count-up reset</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s4(LOAD)</td>
<td>Count-down reset</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n(PV)</td>
<td>Maximum count value</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d1(QU)</td>
<td>Count-up match output</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d2(QD)</td>
<td>Count-down match output</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d3(CV)</td>
<td>Current count value</td>
<td>Word (signed)</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**
- Counts up (+1) (D3) when (s1) is turned ON, and counts down (-1) (D3) when (s2) is turned ON. n sets the maximum value of counter.
- (d2) turns ON when (D3) reaches 0.
- (d1) turns ON when (D3) reaches the maximum value n.
- Resets (D3) when (s3) turns ON.
- The value of n is set to (D3) when (s4) is turned ON.
- Valid setting range for n is -32768 to 32767. However, if 0 or less is set, (d1), (d2) are turned on regardless of the count value of (D3).
### Operation result

- Function without EN/ENO

An operation is executed and the operation value is output from (d1), (d2), and (D3).

[Timing chart]

When n=3

Counts up when (1) is turned from OFF to ON.

Initializes (2) when (3) is turned from OFF to ON.

Counts down when (2) is turned from OFF to ON.

Initializes (3) when (2) is turned from OFF to ON.
• 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), (d3)</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

Counts up when EN = ON and is turned from OFF to ON.
Counts down when EN = ON and is turned from OFF to ON.
Clears 0 when EN = ON and is turned from OFF to ON.
Initializes when EN = ON and is turned from OFF to ON.
**Operation error**

- No operation error occurs.

**Program example**

**CTUD(_E)**
The program which counts the number of times that bit type data input to (s1) is turned from OFF to ON, and turns ON (d1) when the value of (D3) reaches the value set at (n). Simultaneously, it counts the number of times that bit type data input to (s2) is turned from OFF to ON, and turns ON (d2) when the value of (D3) reaches 0.

- Function without EN/ENO (CTD)

  [Structured ladder/FBD]

  ```plaintext
  CTUD_Inst(CU:=g_bool1, CD:=g_bool2, RESET:=g_bool3, LOAD:=g_bool4, PV:=g_int1, QU:=g_bool5, QD:=g_bool6, CV:=g_int2);
  ```

- Function with EN/ENO (CTD_E)

  [Structured ladder/FBD]

  ```plaintext
  CTUD_E_Inst(EN:=M0, CU:=g_bool1, CD:=g_bool2, RESET:=g_bool3, LOAD:=g_bool4, PV:=g_int1, QU:=g_bool5, QD:=g_bool6, CV:=g_int2, ENO:=M10);
  ```
Counter function blocks

5.11 Standard Counter Function Blocks

COUNTER_FB_M

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1  d1</td>
<td></td>
</tr>
<tr>
<td>s2  d2</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td></td>
</tr>
</tbody>
</table>

The following function(s) can go in the dotted squares.

COUNTER_FB_M

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1(Coil)</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>Input argument</td>
<td>s2(Preset)</td>
<td>Counter setting value</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Input argument</td>
<td>s3(ValueIn)</td>
<td>Counter initial value</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>d1(ValueOut)</td>
<td>Counter current value</td>
<td>ANY16</td>
</tr>
<tr>
<td>Output argument</td>
<td>d2(Status)</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

- Counts the detected rising edge (from OFF to ON) of (s1). It is not counted when (s1) stays ON. The count starts from the value input to (s3) and when the count value reaches the value input to (s2), (d2) turns ON. The current value is stored in (d1).
- Valid setting range for (s2) is 0 to 32767.
- Valid setting range for (s3) is -32768 to 32767. However, if negative value is specified, the initial value is 0.
- When resetting the current value of the counter, reset (s1).

**Ex**

When instance name is COUNTER_FB_M_1.

COUNTER_FB_M_1(Coil:=Var_M0,Preset:=10,ValueIn:=0,ValueOut:=Var_D10,Status:=Var_M10);

RST(M15,COUNTER_FB_M_1.Coil);

190 5 APPLICATION FUNCTIONS
5.11 Standard Counter Function Blocks
Operation error

- No operation error occurs.

Program example

**COUNTER_FB_M**

The program which counts the number of times that bit type data input to (s1) is turned from OFF to ON, and outputs the count value from (d1).

[Structured ladder/FBD]

```
15 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
```

Value of Var_D10
5.12 Standard Timer Function Blocks

Pulse timer

**TP(_E), TP_HIGH(_E)**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>s(IN)</td>
<td>Input</td>
<td></td>
<td>Bit</td>
</tr>
<tr>
<td>n(PT)</td>
<td>Output time setting value</td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td>d1(Q)</td>
<td>Output</td>
<td></td>
<td>Bit</td>
</tr>
<tr>
<td>d2(ET)</td>
<td>Elapsed time</td>
<td></td>
<td>Time</td>
</tr>
</tbody>
</table>

**Operation processing**

Turns ON (d1) for the duration set to n after (s) is turned ON. The duration (elapsed time) during which (d1) stays ON is set to (d2). When the elapsed time reaches the preset time, (d1) turns OFF. The elapsed time is not reset even when (d1) turns OFF. After (d1) turns OFF, it is reset when (s) is OFF.

• 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.

• 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.

The following function(s) can go in the dotted squares.

TP, TP_E, TP_HIGH, TP_HIGH_E
### Operation result

- **Function without EN/ENO**

An operation is executed and the operation value is output from (d1) and (d2).

[Timing chart]

When \( n = T\#5s \) (5 seconds)

- Turns (3) ON when (5) = ON.  
  - Starts measuring (4) when (3) = ON.
- Turns (6) OFF when (5) reaches the time set to \( n \).
- Initializes (4) when (5) = OFF and (3) = OFF.

- **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>

[Timing chart]

When \( n = T\#5s \) (5 seconds)

- Turns (3) ON when EN = ON and (5) = ON.  
  - Starts measuring (4) when EN = ON and (3) = ON.
- Turns (6) OFF when (5) reaches the time set to \( n \).
- Measuring time counts up every time when EN = ON after starting measuring.
- Initializes (4) when EN = ON, (3) = OFF and (3) = OFF.

### Precautions

- After the time is up, input to (s) is ignored within one scan. Create a program so that (s) is controlled from the next scan after the time up.

### Operation error

- No operation error occurs.
**TP(_E)**

The program which turns ON bit type data of (d1) for 10 seconds after bit type data input to (s) is turned ON.

- **Function without EN/ENO (TP)**
  
  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram](image1)

  ```plaintext
  TP_Inst(IN:=g_bool1, PT:=T#10s, Q:=g_bool2, ET:=g_time1);
  ```

- **Function with EN/ENO (TP_E)**
  
  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram](image2)

  ```plaintext
  TP_E_Inst(EN:=M0, IN:=g_bool1, PT:=T#10s, Q:=g_bool2, ET:=g_time1, ENO:=M10);
  ```
On delay timer

**TON(_E), TON_HIGH(_E)**

The following function(s) can go in the dotted squares.
TON, TON_E, TON_HIGH, TON_HIGH_E

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>EN</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s(IN)</td>
<td>Input</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>n(PT)</td>
<td>Delay time setting value</td>
<td>Time</td>
</tr>
<tr>
<td>Output argument</td>
<td>ENO</td>
<td>Execution result (TRUE: Normal, FALSE: Error or stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d1(Q)</td>
<td>Output</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>d2(ET)</td>
<td>Elapsed time</td>
<td>Time</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

Turns ON (d1) when (s) is turned ON after the elapse of the time set to n. Elapsed delay time until (d1) is turned ON is set to (d2).

When (s) is turned OFF, (d1) turns OFF and the elapsed delay time is reset.

- **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.

- **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

- **Function without EN/ENO**

An operation is executed and the operation value is output from (d1) and (d2).

[Timing chart]

When \( n = T \#5s \) (5 seconds)

- Starts measuring \( \square \) when \( \triangle \) = ON.
- Turns \( \odot \) ON when \( \square \) reaches the time set to \( n \).
- Resets \( \odot \) at the falling of \( \triangle \).

- **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>

[Timing chart]

When \( n = T \#5s \) (5 seconds)

- Starts measuring \( \square \) when EN ON and \( \triangle \) = ON.
- Turns \( \odot \) ON when \( \square \) reaches the time set to \( n \).
- Turns \( \odot \) OFF and resets \( \odot \) when EN ON and \( \square \) OFF.

### Precautions

- After the time is up, input to (s) is ignored within one scan. Create a program so that (s) is controlled from the next scan after the time up.
Operation error

- No operation error occurs.

Program example

### TON(E)

The program which turns ON bit type data of (d1) 10 seconds after bit type data input to (s) is turned ON.

- Function without EN/ENO (TON)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

  \[
  \text{TON}_{\text{Inst}}(\text{IN}:=\text{g_bool1}, \text{PT}:=\text{T}#10s, \text{Q}:=\text{g_bool2}, \text{ET}:=\text{g_time1});
  \]

- Function with EN/ENO (TON_E)

  [Structured ladder/FBD]

  ![Structured ladder/FBD diagram]

  \[
  \text{TON}_{\text{E} \text{Inst}}(\text{EN}:=\text{M0}, \text{IN}:=\text{g_bool1}, \text{PT}:=\text{T}#10s, \text{Q}:=\text{g_bool2}, \text{ET}:=\text{g_time1}, \text{ENO}:=\text{M10});
  \]
Off delay timer

**TOF(_E), TOF_HIGH(_E)**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Setting range</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOF(_E)</td>
<td>Uses a low-speed timer to count the elapsed time.</td>
<td>CPU module: Basic model QCPU, High Performance model QCPU, Process CPU, Redundant CPU</td>
</tr>
<tr>
<td>TOF_HIGH(_E)</td>
<td>Uses a high-speed timer to count the elapsed time.</td>
<td>CPU module: Universal model QCPU, LCPU</td>
</tr>
</tbody>
</table>

**Operation processing**

- Turns ON (d1) when (s) is turned ON.
- Turns OFF (d1) when (s) is turned from ON to OFF after the elapse of the time set to n. Elapsed time until (d1) is turned OFF is set to (d2).
- When (s) is turned ON again, (d1) turns ON and the elapsed time is reset.

**Data type**

- EN: Bit
- s(IN): Bit
- n(PT): Time
- ENO: Bit
- d1(Q): Bit
- d2(ET): Time

**Processing details**

The following function(s) can go in the dotted squares.

TOF, TOF_E, TOF_HIGH, TOF_HIGH_E
### Operation result

- **Function without EN/ENO**

An operation is executed and the operation value is output from (d1) and (d2).

[Timing chart]

When \( n = T\#5s \) (5 seconds)

- Starts measuring (d3) when \( \text{(5)} \) = OFF.
- Turns (d1) OFF when (d2) reaches the time set to \( n \).
- Resets (d2) when \( \text{(5)} \) = ON.

<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>

[Timing chart]

When \( n = T\#5s \) (5 seconds)

- Starts measuring (d3) when EN = ON and (5) = OFF.
- Resets (d2) when EN = ON and (5) = ON.

### Precautions

- After the time is up, input to (s) is ignored within one scan. Create a program so that (s) is controlled from the next scan after the time up.
Operation error

• No operation error occurs.

Program example

**TOF(E)**
The program which turns ON bit type data of (d1) when bit type data input to (s) is turned ON, and turns (d1) OFF 10 seconds after (s) is turned OFF.

• Function without EN/ENO (TOF)

[Structured ladder/FBD]

```
20
```

[ST]
TOF_Inst(IN:=g_bool1, PT:=T#10s, Q:=g_bool2, ET:=g_time1);

• Function with EN/ENO (TOF_E)

[Structured ladder/FBD]

```
21
```

[ST]
TOF_E_Inst(EN:=M0, IN:=g_bool1, PT:=T#10s, Q:=g_bool2, ET:=g_time1, ENO:=M10);
Timer function blocks

**TIMER_Q_M**

<table>
<thead>
<tr>
<th>Structured ladder/FBD</th>
<th>ST</th>
</tr>
</thead>
</table>
| ```
TIMERE_10_FB_M

s1  s2  s3  d1  d2
```
| ```
 TIMER_10_FB_M(s1, s2, s3, d1, d2)
```

The following function(s) can go in the dotted squares.
TIMER_10_FB_M, TIMER_100_FB_M, TIMER_HIGH_FB_M, TIMER_LOW_FB_M, TIMER_CONT_FB_M, TIMER_CONTHFB_M

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1(Coil)</td>
<td>Executing condition (TRUE: Execution, FALSE: Stop)</td>
<td>Bit</td>
</tr>
<tr>
<td></td>
<td>s2(Preset)</td>
<td>Timer setting value</td>
<td>Word (signed)</td>
</tr>
<tr>
<td></td>
<td>s3(ValueIn)</td>
<td>Timer initial value</td>
<td>Word (signed)</td>
</tr>
<tr>
<td>Output argument</td>
<td>d1(ValueOut)</td>
<td>Timer current value</td>
<td>ANY16</td>
</tr>
<tr>
<td></td>
<td>d2(Status)</td>
<td>Output</td>
<td>Bit</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

**[TIMER_10_FB_M]**
- Starts measuring the current value when the executing condition of (s1) turns ON.
- Starts measuring from the value input to (s3) × 10ms, and when the measuring value reaches to the value input to (s2) × 10ms, (d2) turns ON.
- The current value is output from (d1).
- When the executing condition of (s1) turns OFF, the current value is set to the value input to (s3), and (d2) turns OFF.
- 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.
- Valid setting range for (s2) is 0 to 32767.
- Valid setting range for (s3) is -32768 to 32767. However, if negative value is specified, the initial value is 0.

**[TIMER_100_FB_M]**
- Starts measuring the current value when the executing condition of (s1) turns ON. Starts measuring from the value input to (s3) × 100ms, and when the measuring value reaches to the value input to (s2) × 100ms, (d2) turns ON.
- The current value is output from (d1).
- When the executing condition of (s1) turns OFF, the current value is set to the value input to (s3), and (d2) turns OFF.
- 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.
- Valid setting range for (s2) is 0 to 32767.
- Valid setting range for (s3) is -32768 to 32767. However, if negative value is specified, the initial value is 0.
**[TIMER_HIGH_FB_M]**
- The high-speed timer with the unit of measurement from 0.1 to 100ms. Starts measuring the current value when the executing condition of (s1) turns ON. Starts measuring from the value input to (s3) \( \times 0.1 \) to 100ms, and when the measuring value reaches to the value input to (s2) \( \times 0.1 \) to 100ms, (d2) turns ON.
- The current value is output from (d1).
- When the executing condition of (s1) turns OFF, the current value is set to the value input to (s3), and (d2) turns OFF.
- The default value of the unit of measurement (time period) for the high-speed timer is 10ms.
- This setting is set in the PLC system setting of the PLC parameter.
- Valid setting range for (s2) is 0 to 32767.
- Valid setting range for (s3) is -32768 to 32767. However, if negative value is specified, the initial value is 0.

**[TIMER_LOW_FB_M]**
- The low-speed timer with the unit of measurement from 1 to 1000ms. Starts measuring the current value when the executing condition of (s1) turns ON. Starts measuring from the value input to (s3) \( \times 1 \) to 1000ms, and when the measuring value reaches to the value input to (s2) \( \times 1 \) to 1000ms, (d2) turns ON.
- The current value is output from (d1).
- When the executing condition of (s1) turns OFF, the current value is set to the value input to (s3), and (d2) turns OFF.
- 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 can be changed by unit of 1ms.
- This setting is set in the PLC system setting of the PLC parameter.
- Valid setting range for (s2) is 0 to 32767.
- Valid setting range for (s3) is -32768 to 32767. However, if negative value is specified, the initial value is 0.

**[TIMER_CONT_FB_M,TIMER_CONTHFB_M]**
- The retentive timer that measures the time during variable is ON. Starts measuring the current value when the executing condition of (s1) 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 (s3) \( \times 1 \) to 1000ms, and when the count value reaches to the value input to (s2) \( \times 1 \) to 1000ms, (d2) turns ON. The current value is output from (d1).
- Even when the executing condition of (s1) turns OFF, the ON/OFF statuses of measuring value (d1) and (d2) are retained. When the executing condition of (s1) turns ON again, restarts measuring from the values that are retained.
- 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
- Valid setting range for (s2) is 0 to 32767.
- Valid setting range for (s3) is -32768 to 32767. However, if negative value is specified, the initial value is 0.
- When resetting the current value of the retentive timer, reset (s1).

---

**Ex**

When instance name is TIMER_CONT_FB_M_1.

```
[Structured ladder/FBD]

TIMER_CONT_FB_M_1(Coil:=Var_M0,Preset:=10,ValueIn:=0,ValueOut:=Var_D10,Status:=Var_M10);
RST(M15,TIMER_CONT_FB_M_1.Coil);
```

---

**CPU module**

<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>

---

202

5 APPLICATION FUNCTIONS

5.12 Standard Timer Function Blocks
5 APPLICATION FUNCTIONS

5.12 Standard Timer Function Blocks

**Precautions**

• After the time is up, input to (s) is ignored within one scan. Create a program so that (s) is controlled from the next scan after the time up.

**Operation error**

• No operation error occurs.

**Program example**

**TIMER_10_FB_M**

The program which starts measuring from \((s3) \times 10\text{ms}\) when the executing condition of \((s1)\) turns ON, and when the measuring value reaches to the value input to \((s2) \times 10\text{ms}\), \((d2)\) turns ON.

[Structured ladder/FBD]

```plaintext
TIMER_10_FB_M_Inst(Coil:=Var_M0, Preset:=10, ValueIn:=1, ValueOut:=Var_D10, Status:=Var_M10);
```

[Timing chart]

```
Var_M0 OFF
Value of Var_D10
Var_M10 OFF
```

**TIMER_HIGH_FB_M**

The program which starts measuring from \((s3) \times 10\text{ms}\) when the executing condition of \((s1)\) turns ON, and when the measuring value reaches to the value input to \((s2) \times 10\text{ms}\), \((d2)\) turns ON.

[Structured ladder/FBD]

```plaintext
TIMER_HIGH_FB_M_Inst(Coil:=Var_M0, Preset:=10, ValueIn:=1, ValueOut:=Var_D10, Status:=Var_M10);
```

[Timing chart]

```
Var_M0 OFF
Value of Var_D10
Var_M10 OFF
```
5 APPLICATION FUNCTIONS

5.12 Standard Timer Function Blocks

- **TIMER_LOW_FB_M**
  The program which starts measuring from \((s3) \times 10\text{ms}\) when the executing condition of \((s1)\) turns ON, and when the measuring value reaches to the value input to \((s2) \times 100\text{ms}\), \((d2)\) 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]

  Var_M0 OFF
  Value of Var_D10 1
  Var_M10 OFF
  Value of Var_D10 1

- **TIMER_CONT_FB_M**
  The program which measures from \((s3) \times 10\text{ms}\), and when the measuring value reaches to the value input to \((s2) \times 100\text{ms}\), \((d2)\) 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]

  Var_M0 OFF
  Value of Var_D10 0
  Var_M10 OFF
  Value of Var_D10 0
6 OPERATOR

6.1 Arithmetic Operations

Addition

**ADD, +**

Structured ladder/FBD

<table>
<thead>
<tr>
<th>s1</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>s28</td>
<td></td>
</tr>
</tbody>
</table>

ST

Not supported

The following operator(s) can go in the dotted squares.

**ADD**

Structured ladder/FBD

<table>
<thead>
<tr>
<th>s1</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>s28</td>
<td></td>
</tr>
</tbody>
</table>

ST

d:=s1+s2+...+s28;

The following operator(s) can go in the dotted squares.

+  

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1 to s28</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

For details of the operation processing, refer to the following.

Page 126 Addition

**Operation error**

- No operation error occurs.
**Program example**

**ADD, +**

The program which performs addition \(((s1) + (s2))\) on double word (signed) type data input to \((s1)\) and \((s2)\), and outputs the operation result from \((d)\) in the same data type as that of \((s1)\) and \((s2)\).

[Structured ladder/FBD]

```
g_dint3 := (g_dint1) + (g_dint2);
```

[ST]

```
g_dint3 := (g_dint1) + (g_dint2);
```
Multiplication

MUL, *

<table>
<thead>
<tr>
<th>Basic</th>
<th>High Perform.</th>
<th>Process</th>
<th>Redundant</th>
<th>Universal</th>
<th>CPU</th>
</tr>
</thead>
</table>

**Structured ladder/FBD**

The following operator(s) can go in the dotted squares.

MUL

```
MUL

s1
d
s2
s28
```

**Structured ladder/FBD**

Not supported

```
d := s1 * s2 ... s28;
```

The following operator(s) can go in the dotted squares.

* 

### Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1 to s28</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

### Processing details

#### Operation processing

For details of the operation processing, refer to the following.

Page 128 Multiplication

#### Operation error

- No operation error occurs.

### Program example

**MUL, *

The program which performs multiplication ((s1) × (s2)) on double word (signed) type data input to (s1) and (s2), and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

```
[Structured ladder/FBD]

[ST]
g_dint3 := (g_dint1)*(g_dint2);
```
Subtraction

SUB, -

The following operator(s) can go in the dotted squares.

SUB

The following operator(s) can go in the dotted squares.

- ■

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

Processing details

Operation processing

For details of the operation processing, refer to the following.

Page 130 Subtraction

Operation error

• No operation error occurs.

Program example

SUB, -

The program which performs subtraction ((s1) - (s2)) on double word (signed) type data input to (s1) and (s2), and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

Structured ladder/FBD

\[
g_{\text{d int}3} = (g_{\text{d int}1}) - (g_{\text{d int}2})
\]
Division

DIV, /

The following operator(s) can go in the dotted squares.
DIV

The following operator(s) can go in the dotted squares.
/

■ Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>ANY_NUM</td>
</tr>
</tbody>
</table>

■ Processing details

■ Operation processing

For details of the operation processing, refer to the following.
Page 132 Division

■ Operation error

• An operation error occurs in the following case.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/Q00/Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The value to be input to (s2) is 0. (Division by 0)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
Program example

**DIV, /**

The program which performs division \((\text{s1} \div \text{s2})\) on double word (signed) type data input to (s1) and (s2), and outputs the quotient of the operation result from (d) in the same data type as that of (s1) and (s2).

[Structured ladder/FBD]

\[
\text{g\_dint3}:=(\text{g\_dint1})/(\text{g\_dint2});
\]
Remainder

MOD

<table>
<thead>
<tr>
<th>Operator</th>
<th>arithmetic operations</th>
</tr>
</thead>
</table>

The following operator(s) can go in the dotted squares.
MOD

**Argument**

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1</td>
<td>Input</td>
<td>ANY_INT</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>ANY_INT</td>
</tr>
</tbody>
</table>

**Processing details**

**Operation processing**

For details of the operation processing, refer to the following.

Page 134 Remainder

**Operation error**

- An operation error occurs in the following cases.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
<th>Q00J/ Q000/ Q01</th>
<th>QnH</th>
<th>QnPH</th>
<th>QnPRH</th>
<th>QnU</th>
<th>LCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>The value to be input to (s2) is 0. (Division by 0)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

**Program example**

**MOD**

The program which performs division \((s1 \div s2)\) on double word (signed) type data input to (s1) and (s2), and outputs the remainder of the operation result from (d) in the same data type as that of (s1) and (s2).

ST

\(g\_\text{dint3} := (g\_\text{dint1}) \text{MOD} (g\_\text{dint2});\)
Exponentiation

**

- ** Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1</td>
<td>Input</td>
<td>ANY_REAL</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>Input</td>
<td>ANY_NUM</td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>ANY_REAL</td>
</tr>
</tbody>
</table>

- ** Processing details

- ** Operation processing

For details of the operation processing, refer to the following.

Page 136 Exponentiation

- ** Operation error

These functions consist of the following common instructions depending on the data type of (s1) and (s2).

<table>
<thead>
<tr>
<th>Data type of (s1)</th>
<th>Data type of (s2)</th>
<th>Common instruction used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-precision real type</td>
<td>Word (signed) type</td>
<td>LOG, FLT</td>
</tr>
<tr>
<td>Double word (signed) type</td>
<td>LOG, DFLT</td>
<td></td>
</tr>
<tr>
<td>Single-precision real type</td>
<td>LOG</td>
<td></td>
</tr>
<tr>
<td>Double-precision real type</td>
<td>LOGD, DFLTD</td>
<td></td>
</tr>
<tr>
<td>Double-precision real type</td>
<td>LOGD</td>
<td></td>
</tr>
<tr>
<td>Double-precision real type</td>
<td>LOGD</td>
<td></td>
</tr>
</tbody>
</table>

- ** Program example

The program which performs exponentiation and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

[ST] g_real2 := (g_real1)**(g_int1);
6.2 Logical Operations

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

AND, &, OR, XOR, NOT

The following operator(s) can go in the dotted squares.
AND, OR, XOR

AND, &, OR, XOR

Structured ladder/FBD

ST

The following operator(s) can go in the dotted squares.
AND, &, OR, XOR, NOT

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1 to s28</td>
<td>Input</td>
<td>ANY_BIT</td>
</tr>
<tr>
<td></td>
<td>(s1 only for NOT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output</td>
<td>ANY_BIT</td>
</tr>
</tbody>
</table>

Processing details

Operation processing
For details of the operation processing, refer to the following.
Page 140 Boolean AND, boolean OR, boolean exclusive OR, and boolean NOT

Operation error

• No operation error occurs.
Logical Operations

Program example

AND, &
The program which performs Boolean AND on bit, word (unsigned)/16-bit string type data input to (s1) and (s2) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

```
[Structured ladder/FBD]

[ST]
g_word3 := (g_word1) AND (g_word2);

or

g_word3 := (g_word1) & (g_word2);
```

OR
The program which performs Boolean OR on bit, word (unsigned)/16-bit string type data input to (s1) and (s2) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

```
[Structured ladder/FBD]

[ST]
g_word3 := (g_word1) OR (g_word2);
```

XOR
The program which performs Boolean XOR on bit, word (unsigned)/16-bit string type data input to (s1) and (s2) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1) and (s2).

```
[Structured ladder/FBD]

[ST]
g_word3 := (g_word1) XOR (g_word2);
```

NOT
The program which performs Boolean NOT on bit, word (unsigned)/16-bit string type data input to (s1) bit by bit, and outputs the operation result from (d) in the same data type as that of (s1).

```
[Structured ladder/FBD]

[ST]
g_word2 := NOT(g_word1);
```
6.3 Comparison Operations

Comparison

GT, GE, EQ, LE, LT, NE, >, >=, =, <=, <, <>

The following operator(s) can go in the dotted squares.
GT, GE, EQ, LE, LT, NE

The following operator(s) can go in the dotted squares.
>, >=, =, <=, <, <>

Argument

<table>
<thead>
<tr>
<th>Input/output argument</th>
<th>Name</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input argument</td>
<td>s1 to s28 (s1 and s2 only for NE and &lt;&gt;)</td>
<td>Input</td>
<td>ANY_SIMPLE</td>
</tr>
<tr>
<td>Output argument</td>
<td>d</td>
<td>Output (TRUE: True value, FALSE: False value)</td>
<td>Bit</td>
</tr>
</tbody>
</table>

Processing details

Operation processing
For details of the operation processing, refer to the following.
Page 153 Comparison

Operation error
• No operation error occurs.

Program example

GT,>

The program which performs comparison operation between the values input to (s1) and (s2), and outputs the operation result from (d).

Structured ladder/FBD

```
G_T
s1  s2
   d
```
Not supported

Structured ladder/FBD

```
d:=s1> s2..s28;
```
Not supported

Program example

```
g_bool1:=(g_int1)>g_int2);
```
# INDEX

| A | Application function | 9 |
|   | Arithmetic operations table | 19 |
| B | Basic model QCPU | 9 |
| C | Common instruction | 9 |
|   | Comparison operations table | 19 |
|   | Configuration of functions | 20 |
| F | Function tables | 12 |
|   | Functions of time data types table | 17 |
| G | GX Works2 | 9 |
| H | High Performance model QCPU | 9 |
|   | How to read function tables | 12 |
|   | How to read functions | 22 |
| I | IEC61131-3 | 9 |
|   | Input pins variable function | 21 |
| L | LCPU | 9 |
|   | Logical operations table | 19 |
| O | Operator tables | 19 |
| P | Personal computer | 9 |
|   | Process CPU | 9 |
| Q | QCPU (Q mode) | 9 |
| R | Redundant CPU | 9 |
| S | Special instruction | 9 |
|   | Standard arithmetic functions table | 16 |
|   | Standard bistable function blocks table | 17 |
|   | Standard bitwise Boolean functions table | 16 |
|   | Standard character string functions table | 17 |
|   | Standard comparison functions table | 17 |
|   | Standard counter function blocks table | 18 |
|   | Standard edge detection function blocks table | 18 |
|   | Standard functions of one numeric variable table | 16 |
|   | Standard selection functions table | 16 |
|   | Standard timer function blocks table | 18 |
| T | Type conversion functions table | 13 |
| U | Universal model QCPU | 9 |
# INSTRUCTION INDEX

## Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>205</td>
</tr>
<tr>
<td>-</td>
<td>208</td>
</tr>
<tr>
<td>*</td>
<td>207</td>
</tr>
<tr>
<td>**</td>
<td>212</td>
</tr>
<tr>
<td>/</td>
<td>209</td>
</tr>
<tr>
<td>&amp;</td>
<td>213</td>
</tr>
<tr>
<td>+</td>
<td>205</td>
</tr>
<tr>
<td>&lt;</td>
<td>215</td>
</tr>
<tr>
<td>&lt;=</td>
<td>215</td>
</tr>
<tr>
<td>&gt;</td>
<td>215</td>
</tr>
<tr>
<td>&gt;=</td>
<td>215</td>
</tr>
</tbody>
</table>

## A

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(_E)</td>
<td>123</td>
</tr>
<tr>
<td>ADD</td>
<td>205</td>
</tr>
<tr>
<td>ADD_E</td>
<td>126</td>
</tr>
<tr>
<td>ADD_TIME(_E)</td>
<td>166</td>
</tr>
<tr>
<td>AND</td>
<td>213</td>
</tr>
<tr>
<td>AND_E</td>
<td>140</td>
</tr>
</tbody>
</table>

## B

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCD_TO_DINT(_E)</td>
<td>96</td>
</tr>
<tr>
<td>BCD_TO_DINT(_E)</td>
<td>96</td>
</tr>
<tr>
<td>BCD_TO_INT(_E)</td>
<td>99</td>
</tr>
<tr>
<td>BITARR_TO_DINT(_E)</td>
<td>109</td>
</tr>
<tr>
<td>BITARR_TO_INT(_E)</td>
<td>109</td>
</tr>
<tr>
<td>BOOL_TO_DINT(_E)</td>
<td>24</td>
</tr>
<tr>
<td>BOOL_TO_DINT(_E)</td>
<td>28</td>
</tr>
<tr>
<td>BOOL_TO_DINT(_E)</td>
<td>28</td>
</tr>
<tr>
<td>BOOL_TO_INT(_E)</td>
<td>24</td>
</tr>
<tr>
<td>BOOL_TO_INT(_E)</td>
<td>26</td>
</tr>
<tr>
<td>BOOL_TO_TIME(_E)</td>
<td>30</td>
</tr>
<tr>
<td>BOOL_TO_DINT(_E)</td>
<td>28</td>
</tr>
</tbody>
</table>

## C

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCAT(_E)</td>
<td>157</td>
</tr>
<tr>
<td>COUNTER_FB M.</td>
<td>190</td>
</tr>
<tr>
<td>CPY_BITARR(_E)</td>
<td>113</td>
</tr>
<tr>
<td>CPY_BIT_OF_INT(_E)</td>
<td>119</td>
</tr>
<tr>
<td>CTD(_E)</td>
<td>184</td>
</tr>
<tr>
<td>CTU(_E)</td>
<td>182</td>
</tr>
<tr>
<td>CTUD(_E)</td>
<td>186</td>
</tr>
</tbody>
</table>

## D

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE(_E)</td>
<td>161</td>
</tr>
<tr>
<td>DIV</td>
<td>209</td>
</tr>
<tr>
<td>DINT_TO_BCD(_E)</td>
<td>49</td>
</tr>
<tr>
<td>DINT_TO_BITARR(_E)</td>
<td>111</td>
</tr>
<tr>
<td>DINT_TO_DINT(_E)</td>
<td>111</td>
</tr>
<tr>
<td>DINT_TO_BOOL(_E)</td>
<td>36</td>
</tr>
<tr>
<td>DINT_TO_DWORD(_E)</td>
<td>47</td>
</tr>
<tr>
<td>DINT_TO_INT(_E)</td>
<td>34</td>
</tr>
<tr>
<td>DINT_TO_DWORD(_E)</td>
<td>40</td>
</tr>
<tr>
<td>DINT_TO_REAL(_E)</td>
<td>40</td>
</tr>
<tr>
<td>DINT_TO_REAL(_E)</td>
<td>38</td>
</tr>
<tr>
<td>DINT_TO_DINT(_E)</td>
<td>38</td>
</tr>
<tr>
<td>DINT_TO_DWORD(_E)</td>
<td>42</td>
</tr>
<tr>
<td>DINT_TO_DINT(_E)</td>
<td>42</td>
</tr>
<tr>
<td>DINT_TO_DINT(_E)</td>
<td>42</td>
</tr>
<tr>
<td>DIV</td>
<td>209</td>
</tr>
</tbody>
</table>

## E

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>215</td>
</tr>
<tr>
<td>EQ_E</td>
<td>153</td>
</tr>
<tr>
<td>EXPT(_E)</td>
<td>136</td>
</tr>
</tbody>
</table>

## F

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_TRIG(_E)</td>
<td>180</td>
</tr>
</tbody>
</table>

## G

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GE</td>
<td>215</td>
</tr>
<tr>
<td>GE_E</td>
<td>153</td>
</tr>
<tr>
<td>GET_BIT_OF_INT(_E)</td>
<td>115</td>
</tr>
<tr>
<td>GET_BOOL_ADDR</td>
<td>121</td>
</tr>
<tr>
<td>GET_INT_ADDR</td>
<td>121</td>
</tr>
<tr>
<td>GET_WORD_ADDR</td>
<td>121</td>
</tr>
<tr>
<td>GT</td>
<td>215</td>
</tr>
<tr>
<td>GT_E</td>
<td>153</td>
</tr>
</tbody>
</table>

## I

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT(_E)</td>
<td>159</td>
</tr>
<tr>
<td>INT_TO_BCD(_E)</td>
<td>49</td>
</tr>
<tr>
<td>INT_TO_BITARR(_E)</td>
<td>111</td>
</tr>
<tr>
<td>INT_TO_BOOL(_E)</td>
<td>36</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>32</td>
</tr>
<tr>
<td>INT_TO_DWORD(_E)</td>
<td>47</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>40</td>
</tr>
<tr>
<td>INT_TO_REAL(_E)</td>
<td>38</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>42</td>
</tr>
<tr>
<td>INT_TO_TIME(_E)</td>
<td>52</td>
</tr>
<tr>
<td>INT_TO_DINT(_E)</td>
<td>45</td>
</tr>
</tbody>
</table>

## L

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE</td>
<td>215</td>
</tr>
<tr>
<td>LE_E</td>
<td>153</td>
</tr>
<tr>
<td>LIMITATION(_E)</td>
<td>149</td>
</tr>
<tr>
<td>LREAL_TO_DINT(_E)</td>
<td>56</td>
</tr>
<tr>
<td>LREAL_TO_DINT(_E)</td>
<td>56</td>
</tr>
<tr>
<td>LREAL_TO_REAL(_E)</td>
<td>60</td>
</tr>
<tr>
<td>LT</td>
<td>215</td>
</tr>
<tr>
<td>LT_E</td>
<td>153</td>
</tr>
</tbody>
</table>

## M

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXIMUM(_E)</td>
<td>146</td>
</tr>
<tr>
<td>MID(_E)</td>
<td>155</td>
</tr>
<tr>
<td>MINIMUM(_E)</td>
<td>146</td>
</tr>
<tr>
<td>MOD</td>
<td>211</td>
</tr>
<tr>
<td>Operation</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
</tr>
<tr>
<td>MOD(_E)</td>
<td>134</td>
</tr>
<tr>
<td>MOVE(_E)</td>
<td>138</td>
</tr>
<tr>
<td>MUL</td>
<td>207</td>
</tr>
<tr>
<td>MUL_E</td>
<td>128</td>
</tr>
<tr>
<td>MUL_TIME(_E)</td>
<td>170</td>
</tr>
<tr>
<td>MUX(_E)</td>
<td>151</td>
</tr>
<tr>
<td>NE</td>
<td>215</td>
</tr>
<tr>
<td>NE_E</td>
<td>153</td>
</tr>
<tr>
<td>NOT</td>
<td>213</td>
</tr>
<tr>
<td>NOT(_E)</td>
<td>140</td>
</tr>
<tr>
<td>OR</td>
<td>213</td>
</tr>
<tr>
<td>OR_E</td>
<td>140</td>
</tr>
<tr>
<td>REAL_TO_DINT(_E)</td>
<td>54</td>
</tr>
<tr>
<td>REAL_TO_INT(_E)</td>
<td>54</td>
</tr>
<tr>
<td>REAL_TO_LREAL(_E)</td>
<td>58</td>
</tr>
<tr>
<td>REAL_TO_STR(_E)</td>
<td>62</td>
</tr>
<tr>
<td>REPLACE(_E)</td>
<td>163</td>
</tr>
<tr>
<td>RS(_E)</td>
<td>176</td>
</tr>
<tr>
<td>R_TRIG(_E)</td>
<td>178</td>
</tr>
<tr>
<td>SEL(_E)</td>
<td>144</td>
</tr>
<tr>
<td>SET_BIT_OF_INT(_E)</td>
<td>117</td>
</tr>
<tr>
<td>SR(_E)</td>
<td>174</td>
</tr>
<tr>
<td>STR_TO_BCD(_E)</td>
<td>93</td>
</tr>
<tr>
<td>STR_TO_BOOL(_E)</td>
<td>79</td>
</tr>
<tr>
<td>STR_TO_DINT(_E)</td>
<td>81</td>
</tr>
<tr>
<td>STR_TO_DWORD(_E)</td>
<td>88</td>
</tr>
<tr>
<td>STR_TO_INT(_E)</td>
<td>81</td>
</tr>
<tr>
<td>STR_TO_REAL(_E)</td>
<td>84</td>
</tr>
<tr>
<td>STR_TO_TIME(_E)</td>
<td>91</td>
</tr>
<tr>
<td>STR_TO_WORD(_E)</td>
<td>88</td>
</tr>
<tr>
<td>SUB</td>
<td>208</td>
</tr>
<tr>
<td>SUB_E</td>
<td>130</td>
</tr>
<tr>
<td>SUB_TIME(_E)</td>
<td>168</td>
</tr>
<tr>
<td>TIMER_100_FB_M</td>
<td>201</td>
</tr>
<tr>
<td>TIMER_10_FB_M</td>
<td>201</td>
</tr>
<tr>
<td>TIMER_CONT_FB_M</td>
<td>201</td>
</tr>
<tr>
<td>TIMER_CONTFB_M</td>
<td>201</td>
</tr>
<tr>
<td>TIMER_HIGH_FB_M</td>
<td>201</td>
</tr>
<tr>
<td>TIMER_LOW_FB_M</td>
<td>201</td>
</tr>
<tr>
<td>TIME_TO_BOOL(_E)</td>
<td>101</td>
</tr>
<tr>
<td>TIME_TO_DINT(_E)</td>
<td>103</td>
</tr>
<tr>
<td>TIME_TO_DWORD(_E)</td>
<td>107</td>
</tr>
<tr>
<td>TIME_TO_INT(_E)</td>
<td>103</td>
</tr>
<tr>
<td>TIME_TO_STR(_E)</td>
<td>105</td>
</tr>
<tr>
<td>TIME_TO_WORD(_E)</td>
<td>107</td>
</tr>
<tr>
<td>TOF(_E)</td>
<td>198</td>
</tr>
<tr>
<td>TOF_HIGH(_E)</td>
<td>198</td>
</tr>
<tr>
<td>TON(_E)</td>
<td>195</td>
</tr>
<tr>
<td>TON_HIGH(_E)</td>
<td>195</td>
</tr>
<tr>
<td>TP(_E)</td>
<td>192</td>
</tr>
<tr>
<td>TP_HIGH(_E)</td>
<td>192</td>
</tr>
<tr>
<td>WORD_TO_BOOL(_E)</td>
<td>65</td>
</tr>
<tr>
<td>WORD_TO_DINT(_E)</td>
<td>67</td>
</tr>
<tr>
<td>WORD_TO_DWORD(_E)</td>
<td>71</td>
</tr>
<tr>
<td>WORD_TO_INT(_E)</td>
<td>67</td>
</tr>
<tr>
<td>WORD_TO_STR(_E)</td>
<td>75</td>
</tr>
<tr>
<td>WORD_TO_TIME(_E)</td>
<td>77</td>
</tr>
<tr>
<td>XOR</td>
<td>213</td>
</tr>
<tr>
<td>XOR_E</td>
<td>140</td>
</tr>
</tbody>
</table>
## REVISIONS

*The manual number is given on the bottom left of the back cover.*

<table>
<thead>
<tr>
<th>Revision date</th>
<th><em>Manual number</em></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>July, 2008</td>
<td>SH(NA)-080784ENG-A</td>
<td>Due to the transition to the e-Manual, the details of revision have been deleted.</td>
</tr>
<tr>
<td></td>
<td>SH(NA)-080784ENG-K</td>
<td></td>
</tr>
<tr>
<td>June, 2013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>January, 2017</td>
<td>SH(NA)-080784ENG-L</td>
<td>Complete revision (layout change)</td>
</tr>
<tr>
<td>September, 2018</td>
<td>SH(NA)-080784ENG-M</td>
<td>Descriptions regarding the QnUDPV CPU are added.</td>
</tr>
<tr>
<td>February, 2020</td>
<td>SH(NA)-080784ENG-N</td>
<td>Added or modified parts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 5.5, 5.6, 5.12</td>
</tr>
</tbody>
</table>

Japanese manual version SH-080737-Q

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
Please confirm the following product warranty details before using this product.

1. **Gratis Warranty Term and Gratis Warranty Range**
   If any faults or defects (hereinafter “Failure”) found to be the responsibility of Mitsubishi occurs during use of the product within the gratis warranty term, the product shall be repaired at no cost via the sales representative or Mitsubishi Service Company.

   However, if repairs are required onsite at domestic or overseas location, expenses to send an engineer will be solely at the customer's discretion. Mitsubishi shall not be held responsible for any re-commissioning, maintenance, or testing on-site that involves replacement of the failed module.

   **[Gratis Warranty Term]**
   The gratis warranty term of the product shall be for one year after the date of purchase or delivery to a designated place. Note that after manufacture and shipment from Mitsubishi, the maximum distribution period shall be six (6) months, and the longest gratis warranty term after manufacturing shall be eighteen (18) months. The gratis warranty term of repair parts shall not exceed the gratis warranty term before repairs.

   **[Gratis Warranty Range]**
   (1) The range shall be limited to normal use within the usage state, usage methods and usage environment, etc., which follow the conditions and precautions, etc., given in the instruction manual, user's manual and caution labels on the product.

   (2) Even within the gratis warranty term, repairs shall be charged for in the following cases.

   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:

   (1) Damages caused by any cause found not to be the responsibility of Mitsubishi.

   (2) Loss in opportunity, lost profits incurred to the user by Failures of Mitsubishi products.

   (3) Special damages and secondary damages whether foreseeable or not, compensation for accidents, and compensation for damages to products other than Mitsubishi products.

   (4) 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.
TRADEMARKS

Ethernet is a registered trademark of Fuji Xerox Co., Ltd. in Japan.
Microsoft, Microsoft Access, Excel, SQL Server, Visual Basic, Visual C++, Visual Studio, Windows, Windows NT, Windows Server, Windows Vista, and Windows XP are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
The company names, system names and product names mentioned in this manual are either registered trademarks or trademarks of their respective companies.
In some cases, trademark symbols such as "™" or "®" are not specified in this manual.
SH(NA)-080784ENG-N(2002)KWIX

MODEL: Q-KP-OK-E
MODEL CODE: 13JW08

MITSUBISHI ELECTRIC CORPORATION
HEAD OFFICE: TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
NAGOYA WORKS: 1-14, YADA-MINAMI 5-CHOME, 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.