OMRON

Machine Automation Controller

NJ-series

Instructions Reference Manual

NJ501-1300
NJ501-1400
NJ501-1500

SYSMAC
always in control

W502-E1-01

Introduction

Thank you for purchasing an NJ-series CPU Unit.
This manual contains information that is necessary to use the NJ-series CPU Unit. Please read this manual and make sure you understand the functionality and performance of the NJ-series CPU Unit before you attempt to use it in a control system.
Keep this manual in a safe place where it will be available for reference during operation.

Intended Audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of introducing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of installing and maintaining FA systems.
- Personnel in charge of managing FA systems and facilities.

For programming, this manual is intended for personnel who understand the programming language specifications in international standard IEC 61131-3 or Japanese standard JIS B3503.

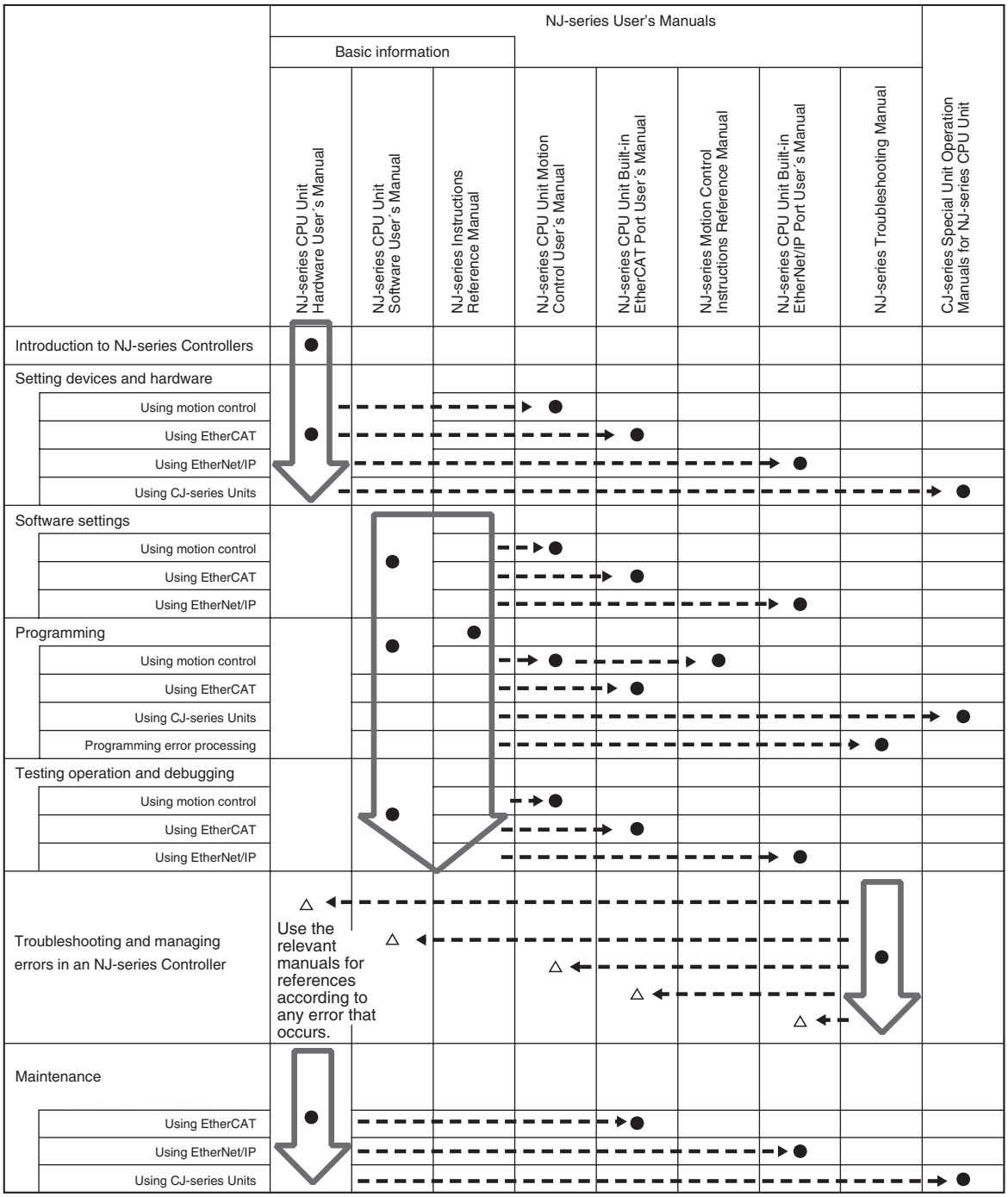
Applicable Products

This manual covers the following products.

- NJ-series CPU Units
  - NJ501-1300
  - NJ501-1400
  - NJ501-1500

Relevant Manuals

There are three manuals that provide basic information on the NJ-series CPU Units: the *NJ-series CPU Unit Hardware User's Manual*, the *NJ-series CPU Unit Software User's Manual* (this manual), and the *NJ-series Instructions Reference Manual*.
Most operations are performed from the Sysmac Studio Automation Software. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504) for information on the Sysmac Studio.
Other manuals are necessary for specific system configurations and applications.
Read all of the manuals that are relevant to your system configuration and application to make the most of the NJ-series CPU Unit.



Manual Configuration

NJ-series CPU Unit Hardware User's Manual (Cat. No. W500)

Section	Description
Section 1 Introduction	This section provides an introduction to the NJ-series Controllers and their features, and gives the NJ-series Controller specifications.
Section 2 System Configuration	This section describes the system configuration used for NJ-series Controllers.
Section 3 Configuration Units	This section describes the parts and functions of the configuration devices in the NJ-series Controller configuration, including the CPU Unit and Configuration Units.
Section 4 Installation and Wiring	This section describes where and how to install the CPU Unit and Configuration Units and how to wire them.
Section 5 Troubleshooting	This section describes the event codes, error confirmation methods, and corrections for errors that can occur.
Section 6 Inspection and Maintenance	This section describes the contents of periodic inspections, the service life of the Battery and Power Supply Units, and replacement methods for the Battery and Power Supply Units.
Appendices	The appendices provide the specifications of the Basic I/O Units, Unit dimensions, load short-circuit protection detection, line disconnection detection, and measures for EMC Directives.

NJ-series CPU Unit Software User's Manual (Cat. No. W501)

Section	Description
Section 1 Introduction	This section provides an introduction to the NJ-series Controllers and their features, and gives the NJ-series Controller specifications.
Section 2 CPU Unit Operation	This section describes the variables and control systems of the CPU Unit and CPU Unit status.
Section 3 I/O Ports, Slave Configuration, and Unit Configuration	This section describes how to use I/O ports, how to create the slave configuration and unit configuration and how to assign functions.
Section 4 Controller Setup	This section describes the initial settings of the function modules.
Section 5 Designing Tasks	This section describes the task system and types of tasks.
Section 6 Programming	This section describes programming, including the programming languages and the variables and instructions that are used in programming.
Section 7 Simulation, Transferring Projects to the Physical CPU Unit, and Operation	This section describes simulation of Controller operation and how to use the results of simulation.
Section 8 CPU Unit Status	This section describes CPU Unit status.
Section 9 CPU Unit Functions	This section describes the functionality provided by the CPU Unit.
Section 10 Communications Setup	This section describes how to go online with the CPU Unit and how to connect to other devices.
Section 11 Example of Actual Application Procedures	This section describes the procedures that are used to actually operate an NJ-series Controller.
Section 12 Troubleshooting	This section describes the event codes, error confirmation methods, and corrections for errors that can occur.
Appendices	The appendices provide the CPU Unit specifications, task execution times, system-defined variable lists, data attribute lists, CJ-series Unit memory information, CJ-series Unit memory allocation methods, and data type conversion information.

NJ-series Instructions Reference Manual (Cat. No. W502) (This Manual)

Section	Description
Section 1 Instruction Set	This section provides a table of the instructions that are described in this manual.
Section 2 Instruction Descriptions	This section describes instruction specifications in detail.
Appendices	The appendices provide a table of error codes and other supplemental information to use instructions.

Sections in this Manual

1 Instruction Set

2 Instruction Descriptions

A Appendices

I Index

CONTENTS

Introduction **1**

Relevant Manuals **2**

Manual Configuration **3**

Sections in this Manual **5**

Read and Understand this Manual **15**

Safety Precautions **19**

Precautions for Safe Use **20**

Precautions for Correct Use **21**

Regulations and Standards **22**

Unit Versions **24**

Related Manuals **27**

Revision History **29**

Section 1 Instruction Set

Instruction Set 1-2

Section 2 Instruction Descriptions

Using this Section 2-2

Ladder Diagram Instructions 2-13

LD and LDN 2-14

AND and ANDN 2-16

OR and ORN 2-18

Out and OutNot 2-20

ST Statement Instructions 2-23

IF 2-24

CASE 2-28

WHILE 2-32

REPEAT 2-34

RETURN 2-36

FOR 2-37

EXIT 2-38

Sequence Input Instructions 2-39

R_TRIG (Up) and F_TRIG (Down) 2-40

TestABit and TestABitN 2-43

Sequence Output Instructions 2-45

RS 2-46

SR 2-48

Set and Reset 2-50

SetBits and ResetBits 2-53

SetABit and ResetABit 2-55

OutABit 2-57
Sequence Control Instructions 2-59
End 2-60
RETURN 2-61
MC and MCR 2-62
JMP 2-74
FOR and NEXT 2-76
BREAK 2-81
Comparison Instructions 2-83
EQ (=) 2-84
NE (<>) 2-86
LT (<), LE (<=), GT (>), and GE (>=) 2-88
EQascii 2-91
NEascii 2-93
LTascii, LEascii, GTascii, and GEascii 2-95
Cmp 2-98
ZoneCmp 2-100
TableCmp 2-102
AryCmpEQ and AryCmpNE 2-105
AryCmpLT, AryCmpLE, AryCmpGT, and AryCmpGE 2-107
AryCmpEQV and AryCmpNEV 2-110
AryCmpLTV, AryCmpLEV, AryCmpGTV, and AryCmpGEV 2-112
Timer Instructions 2-115
TON 2-116
TOF 2-120
TP 2-123
AccumulationTimer 2-126
Timer 2-129
Counter Instructions 2-133
CTD 2-134
CTD_** 2-136
CTU 2-138
CTU_** 2-140
CTUD 2-142
CTUD_** 2-146
Math Instructions 2-151
ADD (+) 2-152
AddOU (+OU) 2-154
SUB (-) 2-156
SubOU (-OU) 2-158
MUL (*) 2-161
MulOU (*OU) 2-163
DIV (/) 2-166
MOD 2-168
ABS 2-170
RadToDeg and DegToRad 2-172
SIN, COS, and TAN 2-174
ASIN, ACOS, and ATAN 2-177
SQRT 2-180
LN and LOG 2-182
EXP 2-185
EXPT (**) 2-187
Inc and Dec 2-189
Rand 2-191
AryAdd 2-193
AryAddV 2-195
ArySub 2-197
ArySubV 2-199

AryMean 2-201
ArySD 2-203
ModReal 2-205
Fraction 2-207
CheckReal 2-209
BCD Conversion Instructions 2-211
_BCD_TO_* 2-212
_TO_BCD_* 2-215
BCD_TO_** 2-218
BCDsToBin 2-221
BinToBCDs_** 2-224
AryToBCD 2-227
AryToBin 2-229
Data Type Conversion Instructions 2-231
_TO_* (Integer-to-Integer Conversion Group) 2-232
_TO_* (Integer-to-Bit String Conversion Group) 2-235
_TO_* (Integer-to-Real Number Conversion Group) 2-237
_TO_* (Bit String-to-Integer Conversion Group) 2-239
_TO_* (Bit String-to-Bit String Conversion Group) 2-242
_TO_* (Bit String-to-Real Number Conversion Group) 2-244
_TO_* (Real Number-to-Integer Conversion Group) 2-246
_TO_* (Real Number-to-Bit String Conversion Group) 2-249
_TO_* (Real Number-to-Real Number Conversion Group) 2-251
**_TO_STRING (Integer-to-Text String Conversion Group) 2-253
**_TO_STRING (Bit String-to-Text String Conversion Group) 2-255
**_TO_STRING (Real Number-to-Text String Conversion Group) 2-257
RealToFormatString 2-259
LrealToFormatString 2-264
STRING_TO_** (Text String-to-Integer Conversion Group) 2-270
STRING_TO_** (Text String-to-Bit String Conversion Group) 2-272
STRING_TO_** (Text String-to-Real Number Conversion Group) 2-274
TO_** (Integer Conversion Group) 2-277
TO_** (Bit String Conversion Group) 2-279
TO_** (Real Number Conversion Group) 2-281
TRUNC, Round, and RoundUp 2-283
Bit String Processing Instructions 2-285
AND (&), OR, and XOR 2-286
XORN 2-289
NOT 2-291
AryAnd, AryOr, AryXor, and AryXorN 2-293
Selection Instructions 2-297
SEL 2-298
MUX 2-300
LIMIT 2-302
Band 2-304
Zone 2-307
MAX and MIN 2-310
AryMax and AryMin 2-312
ArySearch 2-314
Data Movement Instructions 2-317
MOVE 2-318
MoveBit 2-321
MoveDigit 2-323
TransBits 2-325
MemCopy 2-327
SetBlock 2-329
Exchange 2-331
AryExchange 2-333

AryMove 2-335
Clear 2-337
Copy**ToNum (Bit String to Signed Integer) 2-339
Copy**To*** (Bit String to Real Number) 2-341
CopyNumTo** (Signed Integer to Bit String) 2-343
CopyNumTo** (Signed Integer to Real Number) 2-345
Copy**To*** (Real Number to Bit String) 2-347
Copy**ToNum (Real Number to Signed Integer) 2-349
Shift Instructions 2-351
AryShiftReg 2-352
AryShiftRegLR 2-354
ArySHL and ArySHR 2-357
SHL and SHR 2-360
NSHLC and NSHRC 2-362
ROL and ROR 2-364
Conversion Instructions 2-367
Swap 2-368
Neg 2-369
Decoder 2-371
Encoder 2-374
BitCnt 2-376
ColmToLine_** 2-377
LineToColm 2-379
Gray 2-381
PWLApprox 2-384
MovingAverage 2-387
PIDAT 2-393
DispartReal 2-418
UniteReal 2-421
NumToDecString and NumToHexString 2-423
HexStringToNum_** 2-426
FixNumToString 2-428
StringToFixNum 2-430
DtToString 2-433
DateToString 2-435
TodToString 2-436
GrayToBin_** and BinToGray_** 2-438
StringToAry 2-441
AryToString 2-443
DispartDigit 2-445
UniteDigit_** 2-447
Dispart8Bit 2-449
Unite8Bit_** 2-451
ToAryByte 2-453
AryByteTo 2-458
SizeOfAry 2-463
Stack and Table Instructions 2-465
StackPush 2-466
StackFIFO and StackLIFO 2-475
StackIns 2-478
StackDel 2-480
RecSearch 2-482
RecRangeSearch 2-487
RecSort 2-492
RecNum 2-497
RecMax and RecMin 2-499
FCS Instructions 2-503
StringSum 2-504
StringLRC 2-506

StringCRCCCITT 2-508
StringCRC16 2-510
AryLRC_** 2-512
AryCRCCCITT 2-514
AryCRC16 2-516
Text String Instructions 2-519
CONCAT 2-520
LEFT and RIGHT 2-522
MID 2-524
FIND 2-526
LEN 2-528
REPLACE 2-529
DELETE 2-531
INSERT 2-533
GetByteLen 2-535
ClearString 2-537
ToUCase and ToLCase 2-538
TrimL and TrimR 2-540
Time and Time of Day Instructions 2-543
ADD_TIME 2-544
ADD_TOD_TIME 2-546
ADD_DT_TIME 2-548
SUB_TIME 2-550
SUB_TOD_TIME 2-552
SUB_TOD_TOD 2-554
SUB_DATE_DATE 2-555
SUB_DT_DT 2-556
SUB_DT_TIME 2-558
MULTIME 2-560
DIVTIME 2-562
CONCAT_DATE_TOD 2-564
DT_TO_TOD 2-566
DT_TO_DATE 2-568
SetTime 2-570
GetTime 2-572
DtToSec 2-574
DateToSec 2-576
TodToSec 2-577
SecToDt 2-578
SecToDate 2-580
SecToTod 2-582
TimeToNanoSec 2-583
TimeToSec 2-584
NanoSecToTime 2-585
SecToTime 2-586
ChkLeapYear 2-588
GetDaysOfMonth 2-589
DaysToMonth 2-591
GetDayOfWeek 2-593
GetWeekOfYear 2-595
DtToDateStruct 2-597
DateStructToDt 2-599
System Control Instructions 2-601
TraceSamp 2-602
TraceTrig 2-605
GetTraceStatus 2-607
SetAlarm 2-610
ResetAlarm 2-615
GetAlarm 2-617

ResetPLCError 2-619
GetPLCError 2-622
ResetCJBError 2-624
GetCJBError 2-626
GetEIPError 2-628
ResetMCError 2-630
GetMCError 2-634
ResetECError 2-636
GetECError 2-637
SetInfo 2-639
ResetUnit 2-641
GetNTPStatus 2-645
Communications Instructions 2-647
ExecPMCR 2-648
SerialSend 2-658
SerialRcv 2-665
SendCmd 2-674
CIPOpen 2-684
CIPRead 2-692
CIPWrite 2-696
CIPSend 2-701
CIPClose 2-704
CIPUCMMRead 2-706
CIPUCMMWrite 2-710
CIPUCMMSend 2-716
EC_CoESDOWrite 2-726
EC_CoESDORead 2-729
EC_StartMon 2-734
EC_StopMon 2-740
EC_SaveMon 2-742
EC_CopyMon 2-744
EC_DisconnectSlave 2-746
EC_ConnectSlave 2-752
SktUDPCreate 2-754
SktUDPRcv 2-761
SktUDPSend 2-764
SktTCPAccept 2-767
SktTCPConnect 2-770
SktTCPRcv 2-777
SktTCPSend 2-780
SktGetTCPStatus 2-783
SktClose 2-786
SktClearBuf 2-789
SD Memory Card Instructions 2-793
FileWriteVar 2-794
FileReadVar 2-799
FileOpen 2-803
FileClose 2-806
FileSeek 2-809
FileRead 2-812
FileWrite 2-819
FileGets 2-826
FilePuts 2-833
FileCopy 2-840
FileRemove 2-848
FileRename 2-852
DirCreate 2-857
DirRemove 2-860

Other Instructions 2-863
ReadNbit_** 2-864
WriteNbit_** 2-866
ChkRange 2-868
GetMyTaskStatus 2-870
Task_IsActive 2-873
Lock and Unlock 2-875
Get**Clk 2-880
Get**Cnt 2-881

Appendices

A-1 Error Codes Related to Instructions A-2
A-2 Error Code Descriptions A-18
A-3 Error Code Details A-24
A-4 SDO Abort Codes A-47

Index

Read and Understand this Manual

Please read and understand this manual before using the product. Please consult your OMRON representative if you have any questions or comments.

Warranty and Limitations of Liability

WARRANTY

OMRON's exclusive warranty is that the products are free from defects in materials and workmanship for a period of one year (or other period if specified) from date of sale by OMRON.

OMRON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, REGARDING NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR PARTICULAR PURPOSE OF THE PRODUCTS. ANY BUYER OR USER ACKNOWLEDGES THAT THE BUYER OR USER ALONE HAS DETERMINED THAT THE PRODUCTS WILL SUITABLY MEET THE REQUIREMENTS OF THEIR INTENDED USE. OMRON DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED.

LIMITATIONS OF LIABILITY

OMRON SHALL NOT BE RESPONSIBLE FOR SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCTS, WHETHER SUCH CLAIM IS BASED ON CONTRACT, WARRANTY, NEGLIGENCE, OR STRICT LIABILITY.

In no event shall the responsibility of OMRON for any act exceed the individual price of the product on which liability is asserted.

IN NO EVENT SHALL OMRON BE RESPONSIBLE FOR WARRANTY, REPAIR, OR OTHER CLAIMS REGARDING THE PRODUCTS UNLESS OMRON'S ANALYSIS CONFIRMS THAT THE PRODUCTS WERE PROPERLY HANDLED, STORED, INSTALLED, AND MAINTAINED AND NOT SUBJECT TO CONTAMINATION, ABUSE, MISUSE, OR INAPPROPRIATE MODIFICATION OR REPAIR.

Application Considerations

SUITABILITY FOR USE

OMRON shall not be responsible for conformity with any standards, codes, or regulations that apply to the combination of products in the customer's application or use of the products.

At the customer's request, OMRON will provide applicable third party certification documents identifying ratings and limitations of use that apply to the products. This information by itself is not sufficient for a complete determination of the suitability of the products in combination with the end product, machine, system, or other application or use.

The following are some examples of applications for which particular attention must be given. This is not intended to be an exhaustive list of all possible uses of the products, nor is it intended to imply that the uses listed may be suitable for the products:

- Outdoor use, uses involving potential chemical contamination or electrical interference, or conditions or uses not described in this manual.
- Nuclear energy control systems, combustion systems, railroad systems, aviation systems, medical equipment, amusement machines, vehicles, safety equipment, and installations subject to separate industry or government regulations.
- Systems, machines, and equipment that could present a risk to life or property.

Please know and observe all prohibitions of use applicable to the products.

NEVER USE THE PRODUCTS FOR AN APPLICATION INVOLVING SERIOUS RISK TO LIFE OR PROPERTY WITHOUT ENSURING THAT THE SYSTEM AS A WHOLE HAS BEEN DESIGNED TO ADDRESS THE RISKS, AND THAT THE OMRON PRODUCTS ARE PROPERLY RATED AND INSTALLED FOR THE INTENDED USE WITHIN THE OVERALL EQUIPMENT OR SYSTEM.

PROGRAMMABLE PRODUCTS

OMRON shall not be responsible for the user's programming of a programmable product, or any consequence thereof.

Disclaimers

CHANGE IN SPECIFICATIONS

Product specifications and accessories may be changed at any time based on improvements and other reasons.

It is our practice to change model numbers when published ratings or features are changed, or when significant construction changes are made. However, some specifications of the products may be changed without any notice. When in doubt, special model numbers may be assigned to fix or establish key specifications for your application on your request. Please consult with your OMRON representative at any time to confirm actual specifications of purchased products.

DIMENSIONS AND WEIGHTS

Dimensions and weights are nominal and are not to be used for manufacturing purposes, even when tolerances are shown.

PERFORMANCE DATA

Performance data given in this manual is provided as a guide for the user in determining suitability and does not constitute a warranty. It may represent the result of OMRON's test conditions, and the users must correlate it to actual application requirements. Actual performance is subject to the OMRON Warranty and Limitations of Liability.

ERRORS AND OMISSIONS

The information in this manual has been carefully checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical, or proofreading errors, or omissions.

Safety Precautions

Refer to the following manuals for safety precautions.

- NJ-series CPU Unit Hardware User's Manual (Cat No. W500)
- NJ-series CPU Unit Software User's Manual (Cat No. W501)

Precautions for Safe Use

Refer to the following manuals for precautions for safe use.

- NJ-series CPU Unit Hardware User's Manual (Cat No. W500)
- NJ-series CPU Unit Software User's Manual (Cat No. W501)

Precautions for Correct Use

Refer to the following manuals for precautions for correct use.

- NJ-series CPU Unit Hardware User's Manual (Cat No. W500)
- NJ-series CPU Unit Software User's Manual (Cat No. W501)

Regulations and Standards

Conformance to EC Directives

Applicable Directives

- EMC Directives
- Low Voltage Directive

Concepts

● EMC Directive

OMRON devices that comply with EC Directives also conform to the related EMC standards so that they can be more easily built into other devices or the overall machine. The actual products have been checked for conformity to EMC standards.*
Whether the products conform to the standards in the system used by the customer, however, must be checked by the customer. EMC-related performance of the OMRON devices that comply with EC Directives will vary depending on the configuration, wiring, and other conditions of the equipment or control panel on which the OMRON devices are installed. The customer must, therefore, perform the final check to confirm that devices and the overall machine conform to EMC standards.

* Applicable EMC (Electromagnetic Compatibility) standards are as follows:
EMS (Electromagnetic Susceptibility): EN 61131-2 and EN 61000-6-2
EMI (Electromagnetic Interference): EN 61131-2 and EN 61000-6-4 (Radiated emission: 10-m regulations)

● Low Voltage Directive

Always ensure that devices operating at voltages of 50 to 1,000 VAC and 75 to 1,500 VDC meet the required safety standards. The applicable directive is EN 61131-2.

● Conformance to EC Directives

The NJ-series Controllers comply with EC Directives. To ensure that the machine or device in which the NJ-series Controller is used complies with EC Directives, the Controller must be installed as follows:

- The NJ-series Controller must be installed within a control panel.
- You must use reinforced insulation or double insulation for the DC power supplies connected to DC Power Supply Units and I/O Units.
- NJ-series Controllers that comply with EC Directives also conform to the Common Emission Standard (EN 61000-6-4). Radiated emission characteristics (10-m regulations) may vary depending on the configuration of the control panel used, other devices connected to the control panel, wiring, and other conditions.
  You must therefore confirm that the overall machine or equipment complies with EC Directives.

Conformance to Shipbuilding Standards

The NJ-series Controllers comply with the following shipbuilding standards. Applicability to the shipbuilding standards is based on certain usage conditions. It may not be possible to use the product in some locations. Contact your OMRON representative before attempting to use a Controller on a ship.

Usage Conditions for NK and LR Shipbuilding Standards

- The NJ-series Controller must be installed within a control panel.
- Gaps in the door to the control panel must be completely filled or covered with gaskets or other material.
- The following noise filter must be connected to the power supply line.

Noise Filter

Manufacturer	Model
Cosel Co., Ltd.	TAH-06-683

Trademarks

- Sysmac and SYSMAC are trademarks or registered trademarks of OMRON Corporation in Japan and other countries for OMRON factory automation products.
- Windows, Windows 98, Windows XP, Windows Vista, and Windows 7 are registered trademarks of Microsoft Corporation in the USA and other countries.
- EtherCAT® is a registered trademark of Beckhoff Automation GmbH for their patented technology.
- The SD logo is a trademark of SD-3C, LLC.

Other company names and product names in this document are the trademarks or registered trademarks of their respective companies.

Software Licenses and Copyrights

This product incorporates certain third party software. The license and copyright information associated with this software is available at http://www.fa.omron.co.jp/nj_info_e/.

Unit Versions

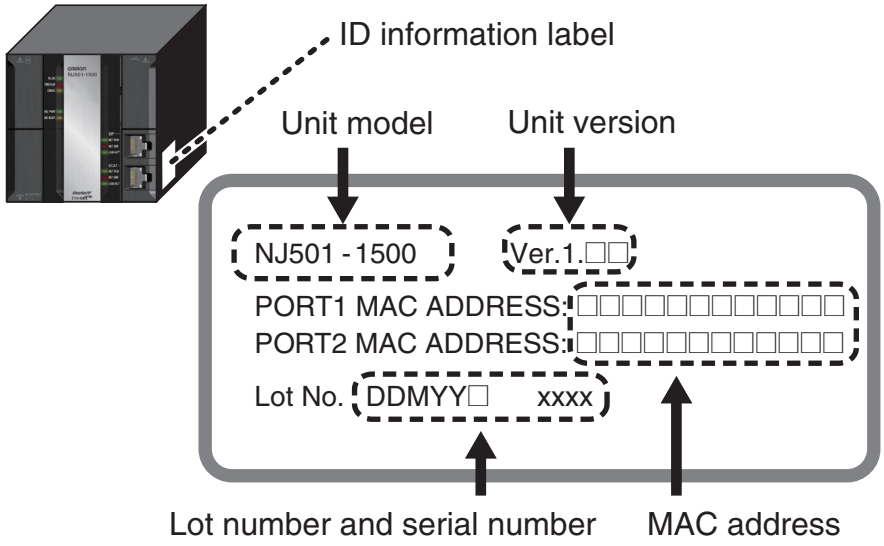
Unit Versions

A “unit version” has been introduced to manage CPU Units in the NJ Series according to differences in functionality accompanying Unit upgrades.

Notation of Unit Versions on Products

The unit version is given on the ID information label of the products for which unit versions are managed, as shown below.

Example for NJ-series NJ501-□□□□ CPU Unit:



The following information is provided on the ID information label.

Item	Description
Unit model	Gives the model of the Unit.
Unit version	Gives the unit version of the Unit.
Lot number and serial number	Gives the lot number and serial number of the Unit. DDMYY: Lot number, □: For use by OMRON, xxxx: Serial number “M” gives the month (1 to 9: January to September, X: October, Y: November, Z: December)
MAC address	Gives the MAC address of the built-in port on the Unit.

Confirming Unit Versions with Sysmac Studio

You can use the Unit Production Information on the Sysmac Studio to check the unit version of the CPU Unit, CJ-series Special I/O Units, CJ-series CPU Bus Units, and EtherCAT slaves. The unit versions of CJ-series Basic I/O Units cannot be checked from the Sysmac Studio.

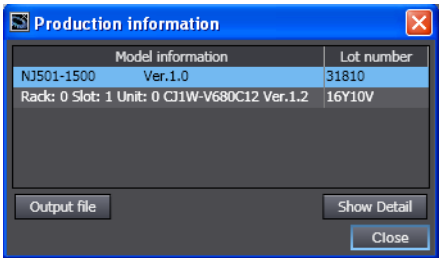
● CPU Unit and CJ-series Units

1 Double-click **CPU/Expansion Racks** under **Configurations and Setup** in the Multiview Explorer. Or, right-click **CPU/Expansion Racks** under **Configurations and Setup** and select ***Edit*** from the menu.

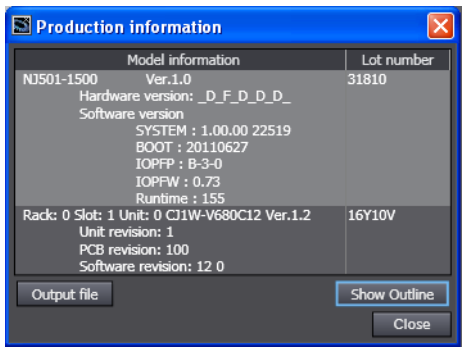
The Unit Editor is displayed for the Controller Configurations and Setup layer.

2 Right-click any open space in the Unit Editor and select ***Production Information***.

The Production Information Dialog Box is displayed.



Simple Display



Detailed Display

In this example, “Ver.1.0” is displayed next to the unit model.

The following items are displayed.

CPU Unit	CJ-series Units
Unit model	Unit model
Unit version	Unit version
Lot number	Lot number
	Rack number, slot number, and unit number

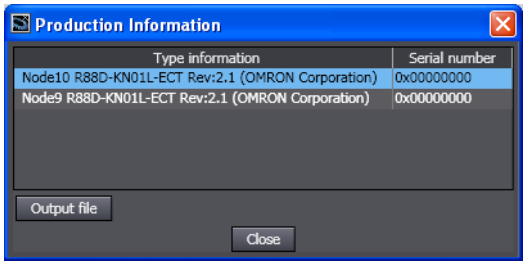
● EtherCAT Slaves

1 Double-click **EtherCAT** under **Configurations and Setup** in the Multiview Explorer. Or, right-click **EtherCAT** under **Configurations and Setup** and select ***Edit*** from the menu.

The EtherCAT Configuration Tab Page is displayed for the Controller Configurations and Setup layer.

2 Right-click the master in the EtherCAT Configurations Editing Pane and select **Display Production Information**.

The Production Information Dialog Box is displayed.



The following items are displayed.
Node address
Type information*
Serial number

* If the model number cannot be determined (such as when there is no ESI file), the vendor ID, product code, and revision number are displayed.

Unit Version Notation

In this manual, unit versions are specified as shown in the following table.

Product nameplate	Notation in this manual	Remarks
“Ver.1.0” or later to the right of the lot number	Unit version 1.0 or later	Unless unit versions are specified, the information in this manual applies to all unit versions.

Related Manuals

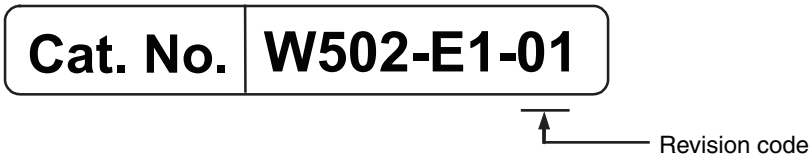
The following manuals are related to the NJ-series Controllers. Use these manuals for reference.

Manual name	Cat. No.	Model numbers	Application	Description
NJ-series CPU Unit Hardware User's Manual	W500	NJ501-□□□□	Learning the basic specifications of the NJ-series CPU Units, including introductory information, designing, installation, and maintenance. Mainly hardware information is provided.	An introduction to the entire NJ-series system is provided along with the following information on a Controller built with an NJ501 CPU Unit. • Features and system configuration • Introduction • Part names and functions • General specifications • Installation and wiring • Maintenance and inspection Use this manual together with the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501).
NJ-series CPU Unit Software User's Manual	W501	NJ501-□□□□	Learning how to program and set up an NJ-series CPU Unit. Mainly software information is provided.	The following information is provided on a Controller built with an NJ501 CPU Unit. • CPU Unit operation • CPU Unit features • Initial settings • Programming based on IEC 61131-3 language specifications Use this manual together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500).
NJ-series CPU Unit Motion Control User's Manual	W507	NJ501-□□□□	Learning about motion control settings and programming concepts.	The settings and operation of the CPU Unit and programming concepts for motion control are described. Use this manual together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500) and *NJ-series CPU Unit Software User's Manual* (Cat. No. W501).
NJ-series Instructions Reference Manual	W502	NJ501-□□□□	Learning about the specifications of the instruction set that is provided by OMRON.	The instructions in the instruction set (IEC 61131-3 specifications) are described. When programming, use this manual together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500) and *NJ-series CPU Unit Software User's Manual* (Cat. No. W501).
NJ-series Motion Control Instructions Reference Manual	W508	NJ501-□□□□	Learning about the specifications of the motion control instructions that are provided by OMRON.	The motion control instructions are described. When programming, use this manual together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500), NJ-series CPU Unit Software User's Manual (Cat. No. W501) and *NJ-series CPU Unit Motion Control User's Manual* (Cat. No. W507).
CJ-series Special Unit Manuals for NJ-series CPU Unit	W490 W498 W499 W491 Z310 W492 W494 W497	CJ1W-□□□□	Learning how to use CJ-series Units with an NJ-series CPU Unit.	The methods and precautions for using CJ-series Units with an NJ501 CPU Unit are described, including access methods and programming interfaces. Manuals are available for the following Units. Analog I/O Units, Insulated-type Analog I/O Units, Temperature Control Units, ID Sensor Units, High-speed Counter Units, Serial Communications Units, and DeviceNet Units. Use these manuals together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500) and *NJ-series CPU Unit Software User's Manual* (Cat. No. W501).

Manual name	Cat. No.	Model numbers	Application	Description
NJ-series CPU Unit Built-in EtherCAT Port User's Manual	W505	NJ501-□□□□	Using the built-in EtherCAT port on an NJ-series CPU Unit.	Information on the built-in EtherCAT port is provided. This manual provides an introduction and provides information on the configuration, features, and setup. Use this manual together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500) and *NJ-series CPU Unit Software User's Manual* (Cat. No. W501).
NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual	W506	NJ501-□□□□	Using the built-in EtherNet/IP port on an NJ-series CPU Unit.	Information on the built-in EtherNet/IP port is provided. Information is provided on the basic setup, tag data links, and other features. Use this manual together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500) and *NJ-series CPU Unit Software User's Manual* (Cat. No. W501).
NJ-series Troubleshooting Manual	W503	NJ501-□□□□	Learning about the errors that may be detected in an NJ-series Controller.	Concepts on managing errors that may be detected in an NJ-series Controller and information on individual errors are described. Use this manual together with the *NJ-series CPU Unit Hardware User's Manual* (Cat. No. W500) and *NJ-series CPU Unit Software User's Manual* (Cat. No. W501).
Sysmac Studio Version 1 Operation Manual	W504	SYSMAC-SE2□□□	Learning about the operating procedures and functions of the Sysmac Studio.	Describes the operating procedures of the Sysmac Studio.
CX-Integrator CS/CJ/CP/NSJ-series Network Configuration Tool Operation Manual	W464		Learning how to configure networks (data links, routing tables, Communications Unit settings, etc.).	Describes operating procedures for the CX-Integrator.
CX-Designer User's Manual	V099		Learning to create screen data for NS-series Programmable Terminals.	Describes operating procedures for the CX-Designer.
CX-Protocol Operation Manual	W344		Creating data transfer protocols for general-purpose devices connected to CJ-series Serial Communications Units.	Describes operating procedures for the CX-Protocol.

Revision History

A manual revision code appears as a suffix to the catalog number on the front and back covers of the manual.



Revision code	Date	Revised content
01	July 2011	Original production

1

Instruction Set

This section provides a table of the instructions that you can use with NJ-series Controllers.

Instruction Set **1-2**

Instruction Set

Type	Instruction	Name	Function	Page
Ladder Diagram Instructions	LD	Load	Reads the value of a BOOL variable.	2-14
	LDN	Load NOT	Reads the inverse of the value of a BOOL variable.	2-14
	AND	AND	Takes the logical AND of the value of a BOOL variable and the input value.	2-16
	ANDN	AND NOT	Takes the logical AND of the inverse of the value of a BOOL variable and the input value.	2-16
	OR	OR	Takes the logical OR of the value of a BOOL variable and the execution condition.	2-18
	ORN	OR NOT	Takes the logical OR of the inverse of the value of a BOOL variable and the execution condition.	2-18
	Out	Output	Takes the logical result from the previous instruction and outputs it to a BOOL variable.	2-20
	OutNot	Output NOT	Takes the inverse of the logical result from the previous instruction and outputs it to a BOOL variable.	2-20
ST Statement Instructions	IF	If	Uses the evaluation result of a specified condition expression to select one of two statements to execute.	2-24
	CASE	Case	Selects the statement to execute based on the value of a specified integer expression.	2-28
	WHILE	While	Repeatedly executes a statement as long as the evaluation result of a specified condition expression is TRUE.	2-32
	REPEAT	Repeat	Executes a statement once and then executes it repeatedly until a specified condition expression is TRUE.	2-34
	RETURN	Return	Ends a function or function block and returns processing to the calling instruction.	2-36
	FOR	Repeat Start	Marks the starting position for repeat processing of statements between the FOR and END_FOR statements and specifies the repeat condition.	2-37
	EXIT	Break Loop	Cancels repeat processing from the lowest level FOR statement to the END_FOR statement.	2-38
Sequence Input Instructions	R_TRIG (Up)	Up Trigger	Outputs TRUE for one task period only when the input signal changes to TRUE.	2-40
	F_TRIG (Down)	Down Trigger	Outputs TRUE for one task period only when the input signal changes to FALSE.	2-40
	TestABit	Test A Bit	Outputs the value of the specified bit in a bit string.	2-43
	TestABitN	Test A Bit NOT	Outputs the inverse of the value of the specified bit in a bit string.	2-43
Sequence Output Instructions	RS	Reset-Priority Keep	Retains the value of a BOOL variable. It gives priority to the *Reset* input if both the *Set* input and *Reset* input are TRUE.	2-46

Type	Instruction	Name	Function	Page
Sequence Output Instructions	SR	Set-Priority Keep	Retains the value of a BOOL variable. It gives priority to the *Set* input if both the *Set* input and *Reset* input are TRUE.	2-48
	Set	Set	Changes a BOOL variable to TRUE.	2-50
	Reset	Reset	Changes a BOOL variable to FALSE.	2-50
	SetBits	Set Bits	Changes consecutive bits in bit string data to TRUE.	2-53
	ResetBits	Reset Bits	Changes consecutive bits in bit string data to FALSE.	2-53
	SetABit	Set A Bit	Changes the specified bit in bit string data to TRUE.	2-55
	ResetABit	Reset A Bit	Changes the specified bit in bit string data to FALSE.	2-55
	OutABit	Output A Bit	Changes the specified bit in bit string data to TRUE or FALSE.	2-57
Sequence Control Instructions	End	End	Ends execution of a program in the current task period.	2-60
	RETURN	Return	Ends a function or function block and returns processing to the calling instruction.	2-61
	MC	Master Control Start	Marks the starting point of a master control region and resets the master control region.	2-62
	MCR	Master Control End	Marks the end point of a master control region.	2-62
	JMP	Jump	Moves processing to the specified jump destination.	2-74
	FOR	Repeat Start	Marks the starting position for repeat processing and specifies the repeat condition.	2-76
	NEXT	Repeat End	Marks the ending position for repeat processing.	2-76
	BREAK	Break Loop	Cancels repeat processing from the lowest level FOR instruction to the NEXT instruction.	2-81
Comparison Instructions	EQ (=)	Equal	Determines if two or more values or text strings are all equivalent.	2-84
	NE (<>)	Not Equal	Determines if two values or text strings are not equivalent.	2-86
	LT (<)	Less Than	Performs a less than comparison between values.	2-88
	LE (<=)	Less Than Or Equal	Performs a less than or equal comparison between values.	2-88
	GT (>)	Greater Than	Performs a greater than comparison between values.	2-88
	GE (>=)	Greater Than Or Equal	Performs a greater than or equal comparison between values.	2-88
	EQascii	Text String Comparison Equal	Determines if two or more text strings are all equivalent.	2-91
	NEascii	Text String Comparison Not Equal	Determines if two text strings are not equivalent.	2-93
	LTascii	Text String Comparison Less Than	Performs a less than comparison between text strings.	2-95
	LEascii	Text String Comparison Less Than or Equal	Performs a less than or equal comparison between text strings.	2-95

Type	Instruction	Name	Function	Page
Comparison Instructions	GTascii	Text String Comparison Greater Than	Performs a greater than comparison between text strings.	2-95
	GEascii	Text String Comparison Greater Than or Equal	Performs a greater than or equal comparison between text strings.	2-95
	Cmp	Compare	Compares two values.	2-98
	ZoneCmp	Zone Comparison	Determines if the comparison data is within the specified maximum and minimum values.	2-100
	TableCmp	Table Comparison	Compares the comparison data with multiple defined ranges in a comparison table.	2-102
	AryCmpEQ	Array Comparison Equal	Determines if the corresponding elements of two arrays are equal.	2-105
	AryCmpNE	Array Comparison Not Equal	Determines if the corresponding elements of two arrays are not equal.	2-105
	AryCmpLT	Array Comparison Less Than	Performs a less than comparison between the corresponding elements of two arrays.	2-107
	AryCmpLE	Array Comparison Less Than Or Equal	Performs a less than or equal comparison between the corresponding elements of two arrays.	2-107
	AryCmpGT	Array Comparison Greater Than	Performs a greater than comparison between the corresponding elements of two arrays.	2-107
	AryCmpGE	Array Comparison Greater Than Or Equal	Performs a greater than or equal comparison between the corresponding elements of two arrays.	2-107
	AryCmpEQV	Array Value Comparison Equal	Determines if the elements of an array are equal to a value.	2-110
	AryCmpNEV	Array Value Comparison Not Equal	Determines if the elements of an array are not equal to a value.	2-110
	AryCmpLTV	Array Value Comparison Less Than	Performs a less than comparison between a value and the elements of an array.	2-112
	AryCmpLEV	Array Value Comparison Less Than Or Equal	Performs a less than or equal comparison between a value and the elements of an array.	2-112
	AryCmpGTV	Array Value Comparison Greater Than	Performs a greater than comparison between a value and the elements of an array.	2-112
	AryCmpGEV	Array Value Comparison Greater Than Or Equal	Performs a greater than or equal comparison between a value and the elements of an array.	2-112
Timer Instructions	TON	On-Delay Timer	Outputs TRUE when the set time elapses after the timer starts.	2-116
	TOF	Off-Delay Timer	Outputs FALSE when the set time elapses after the timer starts.	2-120
	TP	Timer Pulse	Outputs TRUE while the set time elapses after the timer starts.	2-123
	AccumulationTimer	Accumulation Timer	Totals the time that the timer input is TRUE.	2-126
	Timer	Hundred-ms Timer	Outputs TRUE when the set time elapses after the timer starts. Set the time in increments of 100 ms. The timing accuracy is 100 ms.	2-129

Type	Instruction	Name	Function	Page
Counter Instructions	CTD	Down-counter	Decrements the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.	2-134
	CTD_**	Down-counter Group	Decrements the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.	2-136
	CTU	Up-counter	Increments the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.	2-138
	CTU_**	Up-counter Group	Increments the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.	2-140
	CTUD	Up-down Counter	Creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must have an INT data type.	2-142
	CTUD_**	Up-down Counter Group	Creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.	2-146
Math Instructions	ADD (+)	Addition	Adds integers and real numbers. Also joins text strings.	2-152
	AddOU (+OU)	Addition with Overflow/ Underflow Check	Adds integers and real numbers. Also performs an overflow/underflow check.	2-154
	SUB (-)	Subtraction	Subtracts integers and real numbers.	2-156
	SubOU (-OU)	Subtraction with Overflow/ Underflow Check	Subtracts integers or real numbers. Also performs an overflow/underflow check.	2-158
	MUL (*)	Multiplication	Multiplies integers and real numbers.	2-161
	MulOU (*OU)	Multiplication with Overflow/ Underflow Check	Multiplies integers and real numbers and outputs the result. It also performs an overflow/underflow check.	2-163
	DIV (/)	Division	Divides integers or real numbers.	2-166
	MOD	Modulo-division	Finds the remainder for division of integers.	2-168
	ABS	Absolute Value	Finds the absolute value of an integer or real number.	2-170
	RadToDeg	Radians to Degrees	Converts a real number from radians (rad) to degrees (°).	2-172
	DegToRad	Degrees to Radians	Converts a real number from degrees (°) to radians (rad).	2-172
	SIN	Sine in Radians	Calculates the sine of a real number.	2-174
	COS	Cosine in Radians	Calculates the cosine of a real number.	2-174
	TAN	Tangent in Radians	Calculates the tangent of a real number.	2-174
	ASIN	Principal Arc Sine	Calculates the arcsine of a real number.	2-177
	ACOS	Principal Arc Cosine	Calculates the arccosine of a real number.	2-177
	ATAN	Principal Arc Tangent	Calculates the arctangent of a real number.	2-177

Type	Instruction	Name	Function	Page
Math Instructions	SQRT	Square Root	Finds the square root of a number.	2-180
	LN	Natural Logarithm	Finds the natural logarithm of a real number.	2-182
	LOG	Logarithm Base 10	Finds the base-10 logarithm of a real number.	2-182
	EXP	Natural Exponential Operation	Performs calculations for the natural exponential function.	2-185
	EXPT (**)	Exponentiation	Raises one real number to the power of another real number.	2-187
	Inc	Increment	Increments an integer value.	2-189
	Dec	Decrement	Decrements an integer value.	2-189
	Rand	Random Number	Generates pseudorandom numbers.	2-191
	AryAdd	Array Addition	Adds corresponding elements of two arrays.	2-193
	AryAddV	Array Value Addition	Adds the same value to specified elements of an array.	2-195
	ArySub	Array Subtraction	Subtracts corresponding elements of two arrays.	2-197
	ArySubV	Array Value Subtraction	Subtracts the same value from specified elements of an array.	2-199
	AryMean	Array Mean	Calculates the average of the elements of an array.	2-201
	ArySD	Array Element Standard Deviation	Calculates standard deviation of the elements of an array.	2-203
	ModReal	Real Number Modulo-division	Calculates the remainder of real number division.	2-205
	Fraction	Real Number Fraction	Finds the fractional part of a real number.	2-207
	CheckReal	Real Number Check	Checks a real number to see if it is infinity or nonnumeric data.	2-209
BCD Conversion Instructions	**_BCD_TO_***	BCD-to-Unsigned Integer Conversion Group	Converts BCD bit strings into unsigned integers.	2-212
	_TO_BCD_*	Unsigned Integer-to-BCD Conversion Group	Converts unsigned integers to BCD bit strings.	2-215
	BCD_TO_**	BCD Data Type-to-Unsigned Integer Conversion Group	Converts BCD bit strings into unsigned integers.	2-218
	BCDsToBin	Signed BCD-to-Signed Integer Conversion	Converts signed BCD bit strings to signed integers.	2-221
	BinToBCDs_**	Signed Integer-to-BCD Conversion Group	Converts signed integers to signed BCD bit strings.	2-224
	AryToBCD	Array BCD Conversion	Converts the elements of an unsigned integer array to BCD bit strings.	2-227
	AryToBin	Array Unsigned Integer Conversion	Converts the elements of an array of BCD bit strings into unsigned integers.	2-229
Data Type Conversion Instructions	**_TO_*** (Integer-to-Integer Conversion Group)	Integer-to-Integer Conversion Group	Converts integers to integers with different data types.	2-232
	_TO_* (Integer-to-Bit String Conversion Group)	Integer-to-Bit String Conversion Group	Converts integers to bit strings.	2-235

Type	Instruction	Name	Function	Page
Data Type Conversion Instructions	**_TO_*** (Integer-to-Real Number Conversion Group)	Integer-to-Real Number Conversion Group	Converts integers to real numbers.	2-237
	_TO_* (Bit String-to-Integer Conversion Group)	Bit String-to-Integer Conversion Group	Converts bit strings to integers.	2-239
	_TO_* (Bit String-to-Bit String Conversion Group)	Bit String-to-Bit String Conversion Group	Converts bit strings to bit strings with different data types.	2-242
	_TO_* (Bit String-to-Real Number Conversion Group)	Bit String-to-Real Number Conversion Group	Converts bit strings to real numbers.	2-244
	_TO_* (Real Number-to-Integer Conversion Group)	Real Number-to-Integer Conversion Group	Converts real numbers to integers.	2-246
	_TO_* (Real Number-to-Bit String Conversion Group)	Real Number-to-Bit String Conversion Group	Converts real numbers to bit strings.	2-249
	_TO_* (Real Number-to-Real Number Conversion Group)	Real Number-to-Real Number Conversion Group	Converts real numbers to real numbers with different data types.	2-251
	**_TO_STRING (Integer-to-Text String Conversion Group)	Integer-to-Text String Conversion Group	Converts integers to text strings.	2-253
	**_TO_STRING (Bit String-to-Text String Conversion Group)	Bit String-to-Text String Conversion Group	Converts bit strings to text strings.	2-255
	**_TO_STRING (Real Number-to-Text String Conversion Group)	Real Number-to-Text String Conversion Group	Converts real numbers to text strings.	2-257
	RealToFormatString	REAL-to-Formatted Text String	Converts a REAL variable to a text string with the specified format.	2-259
	LrealToFormatString	LREAL-to-Formatted Text String	Converts a LREAL variable to a text string with the specified format.	2-264
	STRING_TO_** (Text String-to-Integer Conversion Group)	Text String-to-Integer Conversion Group	Converts text strings to integers.	2-270
	STRING_TO_** (Text String-to-Bit String Conversion Group)	Text String-to-Bit String Conversion Group	Converts text strings to bit strings.	2-272
	STRING_TO_** (Text String-to-Real Number Conversion Group)	Text String-to-Real Number Conversion Group	Converts text strings to real numbers.	2-274
	TO_** (Integer Conversion Group)	Integer Conversion Group	Converts integers, bit strings, real numbers, and text strings to integers.	2-277
	TO_** (Bit String Conversion Group)	Bit String Conversion Group	Converts integers, bit strings, real numbers, and text strings to bit strings.	2-279
	TO_** (Real Number Conversion Group)	Real Number Conversion Group	Converts integers, bit strings, real numbers, and text strings to real numbers.	2-281
	TRUNC	Truncate	Truncates a real number at the first decimal digit to make an integer.	2-283
	Round	Round Off Real Number	Rounds a real number at the first decimal digit to make an integer.	2-283

Type	Instruction	Name	Function	Page
Data Type Conversion Instructions	RoundUp	Round Up Real Number	Rounds up a real number at the first decimal digit to make an integer.	2-283
Bit String Processing Instructions	AND (&)	Logical AND	Performs a logical AND operation on Boolean variables or individual bits in bit stings.	2-286
	OR	Logical OR	Performs a logical OR operation on Boolean variables or individual bits in bit stings.	2-286
	XOR	Logical Exclusive OR	Performs a logical exclusive OR operation on Boolean variables or individual bits in bit stings.	2-286
	XORN	Logical Exclusive NOR	Performs a logical exclusive NOR operation on Boolean variables or individual bits in bit stings.	2-289
	NOT	Bit Reversal	Reverses the value of a Boolean variable or individual bits in a bit string.	2-291
	AryAnd	Array Logical AND	Performs a logical AND operation on Boolean variables or individual bits in bit stings between arrays.	2-293
	AryOr	Array Logical OR	Performs a logical OR operation on Boolean variables or individual bits in bit stings between arrays.	2-293
	AryXor	Array Logical Exclusive OR	Performs a logical exclusive OR operation on Boolean variables or individual bits in bit stings between arrays.	2-293
	AryXorN	Array Logical Exclusive NOR	Performs a logical exclusive NOR operation on Boolean variables or individual bits in bit stings between arrays.	2-293
Selection Instructions	SEL	Binary Selection	Selects one of two selections.	2-298
	MUX	Multiplexer	Selects one of two to five selections.	2-300
	LIMIT	Limiter	Limits the value of the input variable to the specified minimum and maximum values.	2-302
	Band	Deadband Control	Performs deadband control.	2-304
	Zone	Dead Zone Control	Adds a bias value to the input value.	2-307
	MAX	Maximum	Finds the largest of two to five values.	2-310
	MIN	Minimum	Finds the smallest of two to five values.	2-310
	AryMax	Array Maximum	Finds the elements with the largest value in a one-dimensional array.	2-312
	AryMin	Array Minimum	Finds the elements with the smallest value in a one-dimensional array.	2-312
	ArySearch	Array Search	Searches for the specified value in a one-dimensional array.	2-314
Data Movement Instructions	MOVE	Move	Moves the value of a constant or variable to another variable.	2-318
	MoveBit	Move Bit	Moves one bit in a bit string.	2-321
	MoveDigit	Move Digit	Moves digits (4 bits per digit) in a bit string.	2-323
	TransBits	Move Bits	Moves one or more bits in a bit string.	2-325
	MemCopy	Memory Copy	Moves one or more array elements. The move source and move destination must have the same data type.	2-327
	SetBlock	Block Set	Moves the value of a variable or constant to one or more array elements.	2-329
	Exchange	Data Exchange	Exchanges the values of two variables.	2-331

Type	Instruction	Name	Function	Page
Data Movement Instructions	AryExchange	Array Data Exchange	Exchanges the elements of two arrays.	2-333
	AryMove	Array Move	Moves one or more array elements. The data types of the move source and move destination can be different.	2-335
	Clear	Initialize	Initializes a variable.	2-337
	Copy**ToNum (Bit String to Signed Integer)	Bit Pattern Copy (Bit String to Signed Integer) Group	Copies the content of a bit string directly to a signed integer.	2-339
	Copy**To*** (Bit String to Real Number)	Bit Pattern Copy (Bit String to Real Number) Group	Copies the content of a bit string directly to a real number.	2-341
	CopyNumTo** (Signed Integer to Bit String)	Bit Pattern Copy (Signed Integer to Bit String) Group	Copies the content of a signed integer directly to a bit string.	2-343
	CopyNumTo** (Signed Integer to Real Number)	Bit Pattern Copy (Signed Integer to Real Number) Group	Copies the content of a signed integer directly to a real number.	2-345
	Copy**To*** (Real Number to Bit String)	Bit Pattern Copy (Real Number to Bit String) Group	Copies the content of a real number directly to a bit string.	2-347
	Copy**ToNum (Real Number to Signed Integer)	Bit Pattern Copy (Real Number to Signed Integer) Group	Copies the content of a real number directly to a signed integer.	2-349
Shift Instructions	AryShiftReg	Shift Register	Shifts a bit string one bit to the left and inserts the input value to the least-significant bit. The bit string consists of array elements.	2-352
	AryShiftRegLR	Reversible Shift Register	Shifts a bit string one bit to the left or right and inserts the input value to the least-significant or most-significant bit. The bit string consists of array elements.	2-354
	ArySHL	Array N-element Left Shift	Shifts array elements by one or more elements to the left (toward the higher elements).	2-357
	ArySHR	Array N-element Right Shift	Shifts array elements by one or more elements to the right (toward the lower elements).	2-357
	SHL	N-bit Left Shift	Shifts a bit string by one or more bits to the left (toward the higher bits).	2-360
	SHR	N-bit Right Shift	Shifts a bit string by one or more bits to the right (toward the lower bits).	2-360
	NSHLC	Shift N-bits Left with Carry	Shifts an array of bit strings that includes the Carry (CY) Flag by one or more bits to the left (toward the higher elements).	2-362
	NSHRC	Shift N-bits Right with Carry	Shifts an array of bit strings that includes the Carry (CY) Flag by one or more bits to the right (toward the lower elements).	2-362
	ROL	Rotate N-bits Left	Rotates a bit string by one or more bits to the left (toward the higher bits).	2-364
	ROR	Rotate N-bits Right	Rotates a bit string by one or more bits to the right (toward the lower bits).	2-364
Conversion Instructions	Swap	Swap Bytes	Swaps the upper byte and lower byte of a 16-bit value.	2-368
	Neg	Reverse Sign	Reverses the sign of a number.	2-369

Type	Instruction	Name	Function	Page
Conversion Instructions	Decoder	Bit Decoder	Sets the specified bit to TRUE and the other bits to FALSE in array elements that consist of a maximum of 256 bits.	2-371
	Encoder	Bit Encoder	Finds the position of the highest TRUE bit in array elements that consist of a maximum of 256 bits.	2-374
	BitCnt	Bit Counter	Counts the number of TRUE bits in a bit string.	2-376
	ColmToLine_**	Column to Line Conversion Group	Extracts bit values from the specified position of array elements and outputs them as a bit string.	2-377
	LineToColm	Line to Column Conversion	Takes the bits from a bit string and outputs them to the specified bit position in array elements.	2-379
	Gray	Gray Code Conversion	Converts a gray code into an angle.	2-381
	PWLApprox	Broken Line Approximation	Performs broken line approximations for integers or real numbers.	2-384
	MovingAverage	Moving Average	Calculates a moving average.	2-387
	PIDAT	PID with Autotuning	Performs PID control with autotuning (2-PID control with set point filter).	2-393
	DispartReal	Separate Mantissa and Exponent	Separates a real number into the signed mantissa and the exponent.	2-418
	UniteReal	Combine Real Number Mantissa and Exponent	Combines a signed mantissa and exponent to make a real number.	2-421
	NumToDecString	Fixed-length Decimal Text String Conversion	Converts an integer to a fixed-length decimal text string.	2-423
	NumToHexString	Fixed-length Hexadecimal Text String Conversion	Converts an integer to a fixed-length hexadecimal text string.	2-423
	HexStringToNum_**	Hexadecimal Text String-to-Number Conversion Group	Converts a hexadecimal text string to an integer.	2-426
	FixNumToString	Fixed-decimal Number-to-Text String Conversion	Converts a signed fixed-decimal number to a decimal text string.	2-428
	StringToFixNum	Text String-to-Fixed-decimal Conversion	Converts a decimal text string to a signed fixed-decimal number.	2-430
	DtToString	Date and Time-to-Text String Conversion	Converts a date and time to a text string.	2-433
	DateToString	Date-to-Text String Conversion	Converts a date to a text string.	2-435
	TodToString	Time of Day-to-Text String Conversion	Converts a time of day to a text string.	2-436
	GrayToBin_**	Gray Code-to-Binary Code Conversion Group	Converts a gray code to a bit string.	2-438
	BinToGray_**	Binary Code-to-Gray Code Conversion	Converts a bit string to a gray code.	2-438

Type	Instruction	Name	Function	Page
Conversion Instructions	StringToAry	Text String-to-Array Conversion	Converts a text string to a BYTE array.	2-441
	AryToString	Array-to-Text String Conversion	Converts a BYTE array to a text string.	2-443
	DispartDigit	Four-bit Separation	Separates a bit string into 4-bit units.	2-445
	UniteDigit_**	Four-bit Join Group	Joins 4-bit units of data into a bit string.	2-447
	Dispart8Bit	Byte Data Separation	Separates a bit string into individual bytes.	2-449
	Unite8Bit_**	Byte Data Join Group	Joins bytes of data into a bit string.	2-451
	ToAryByte	Conversion to Byte Array	Separates the value of a variable into bytes and stores them in a BYTE array.	2-453
	AryByteTo	Conversion from Byte Array	Joins BYTE array elements and stores the result in a variable.	2-458
	SizeOfAry	Get Number of Array Elements	Gets the number of elements in an array.	2-463
Stack and Table Instructions	StackPush	Push onto Stack	Stores a value in a stack.	2-466
	StackFIFO	First In First Out	Removes the bottom value from a stack.	2-475
	StackLIFO	Last In First Out	Removes the top value from a stack.	2-475
	StackIns	Insert into Stack	Inserts a value at a specified position in a stack.	2-478
	StackDel	Delete from Stack	Deletes a value from a specified position in a stack.	2-480
	RecSearch	Record Search	Searches an array of structures for elements that match the search key with the specified method.	2-482
	RecRangeSearch	Range Record Search	Searches an array of structures for elements that match the search condition range with the specified method.	2-487
	RecSort	Record Sort	Sorts the elements of an array of structures.	2-492
	RecNum	Get Number of Records	Finds the number of records in an array of structures to the end data.	2-497
	RecMax	Maximum Record Search	Searches the specified member in the structures of an array of structures for the maximum value.	2-499
	RecMin	Minimum Record Search	Searches the specified member in the structures of an array of structures for the minimum value.	2-499
FCS Instructions	StringSum	Checksum Calculation	Calculates the checksum for a text string.	2-504
	StringLRC	Calculate Text String LRC	Calculates the LRC value (horizontal parity).	2-506
	StringCRCCCITT	Calculate Text String CRC-CCITT	Calculates the CRC-CCITT value using the XMODEM method.	2-508
	StringCRC16	Calculate Text String CRC-16	Calculates the CRC-16 value using the MODBUS method.	2-510
	AryLRC_**	Calculate Array LRC Group	Calculates the LRC value for an array	2-512
	AryCRCCCITT	Calculate Array CRC-CCITT	Calculates the CRC-CCITT value using the XMODEM method.	2-514
	AryCRC16	Calculate Array CRC-16	Calculates the CRC-16 value using the MODBUS method.	2-516

Type	Instruction	Name	Function	Page
Text String Instructions	CONCAT	Concatenate String	Joins two to five text strings.	2-520
	LEFT	Get String Left	Extracts a text string with the specified number of characters from the start (left) of a text string.	2-522
	RIGHT	Get String Right	Extracts a text string with the specified number of characters from the end (right) of a text string.	2-522
	MID	Get String Any	Extracts a text string with the specified number of characters from the specified character position.	2-524
	FIND	Find String	Searches a specified text string for the position of a specified text string.	2-526
	LEN	String Length	Finds the number of characters in a text string.	2-528
	REPLACE	Replace String	Replaces part of a text string with another text string	2-529
	DELETE	Delete String	Deletes all or part of a text string.	2-531
	INSERT	Insert String	Inserts a text string into another text string.	2-533
	GetByteLen	Get Byte Length	Counts the number of bytes in a text string.	2-535
	ClearString	Clear String	Clears a text string.	2-537
	ToUCase	Convert to Uppercase	Converts all single-byte letters in a text string to uppercase.	2-538
	ToLCase	Convert to Lowercase	Converts all single-byte letters in a text string to lowercase.	2-538
	TrimL	Trim String Left	Removes blank space from the beginning of a text string.	2-540
	TrimR	Trim String Right	Removes blank space from the end of a text string.	2-540
Time and Time of Day Instructions	ADD_TIME	Add Time	Adds two times.	2-544
	ADD_TOD_TIME	Add Time to Time of Day	Adds a time to a time of day.	2-546
	ADD_DT_TIME	Add Time to Date and Time	Adds a time to a date and time.	2-548
	SUB_TIME	Subtract Time	Subtracts one time from another.	2-550
	SUB_TOD_TIME	Subtract Time from Time of Day	Subtracts a time from a time of day.	2-552
	SUB_TOD_TOD	Subtract Time of Day	Subtracts a time of day from another time of day.	2-554
	SUB_DATE_DATE	Subtract Date	Subtracts another date from another date.	2-555
	SUB_DT_DT	Subtract Date and Time	Subtracts another date and time from another date and time.	2-556
	SUB_DT_TIME	Subtract Time from Date and Time	Subtracts a time from a date and time.	2-558
	MULTIME	Multiply Time	Multiplies a time by a specified number.	2-560
	DIVTIME	Divide Time	Divides a time by a specified number.	2-562
	CONCAT_DATE_TOD	Concatenate Date and Time of Day	Combines a date and a time of day.	2-564
	DT_TO_TOD	Extract Time of Day from Date and Time	Extracts the time of day from a date and time.	2-566
	DT_TO_DATE	Extract Date from Date and Time	Extracts the date from a date and time.	2-568
	SetTime	Set Time	Sets the system time.	2-570

Type	Instruction	Name	Function	Page
Time and Time of Day Instructions	GetTime	Get Time of Day	Reads the current time.	2-572
	DtToSec	Convert Date and Time to Seconds	Converts a date and time to the number of seconds from 00:00:00 on January 1, 1970.	2-574
	DateToSec	Convert Date to Seconds	Converts a date to the number of seconds from 00:00:00 on January 1, 1970.	2-576
	TodToSec	Convert Time of Day to Seconds	Converts a time of day to the number of seconds from 00:00:00.	2-577
	SecToDt	Convert Seconds to Date and Time	Converts the number of seconds from 00:00:00 on January 1, 1970 to a date and time.	2-578
	SecToDate	Convert Seconds to Date	Converts the number of seconds from 00:00:00 on January 1, 1970 to a date.	2-580
	SecToTod	Convert Seconds to Time of Day	Converts the number of seconds from 00:00:00 to a time of day.	2-582
	TimeToNanoSec	Convert Time to Nanoseconds	Converts a time to nanoseconds.	2-583
	TimeToSec	Convert Time to Seconds	Converts a time to seconds.	2-584
	NanoSecToTime	Convert Nanoseconds to Time	Converts nanoseconds to a time.	2-585
	SecToTime	Convert Seconds to Time	Converts seconds to a time.	2-586
	ChkLeapYear	Check for Leap Year	Checks for a leap year.	2-588
	GetDaysOfMonth	Get Days in Month	Gets the number of days in the specified month.	2-589
	DaysToMonth	Convert Days to Month	Calculates the month based on the number of days from January 1.	2-591
	GetDayOfWeek	Get Day of Week	Gets the day of the week for the specified year, month, and day of month.	2-593
	GetWeekOfYear	Get Week Number	Gets the week number for the specified year, month, and day of month.	2-595
	DtToDateStruct	Break Down Date and Time	Converts a date and time to the year, month, day, hour, minutes, seconds, and nanoseconds.	2-597
	DateStructToDt	Join Time	Joins a year, month, day, hour, minutes, seconds, and nanoseconds into a date and time.	2-599
System Control Instructions	TraceSamp	Data Trace Sampling	Performs sampling for a data trace.	2-602
	TraceTrig	Data Trace Trigger	Generates a trigger for data tracing.	2-605
	GetTraceStatus	Read Data Trace Status	Reads the execution status of a data trace.	2-607
	SetAlarm	Create User-defined Error	Creates a user-defined error.	2-610
	ResetAlarm	Reset User-defined Error	Resets a user-defined error.	2-615
	GetAlarm	Get User-defined Error Status	Gets the highest event level (of user-defined error levels 1 to 8) and the highest level event code of the current user-defined errors.	2-617
	ResetPLCError	Reset PLC Controller Error	Resets errors in the PLC Function Module.	2-619

Type	Instruction	Name	Function	Page
System Control Instructions	GetPLCError	Get PLC Controller Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the PLC Function Module.	2-622
	ResetCJBError	Reset CJ Bus Controller Error	Resets a Controller Error in the I/O bus.	2-624
	GetCJBError	Get I/O Bus Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the I/O bus.	2-626
	GetEIPError	Get EtherNet/IP Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the EtherNet/IP Function Module.	2-628
	ResetMCError	Reset Motion Control Error	Resets a Controller Error in the Motion Control Function Module.	2-630
	GetMCError	Get Motion Control Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the Motion Control Function Module.	2-634
	ResetECError	Reset EtherCAT Error	Resets a Controller Error in the EtherCAT Master Function Module.	2-636
	GetECError	Get EtherCAT Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current communications port errors or master errors in the EtherCAT Master Function Module.	2-637
	SetInfo	Create User-defined Information	Creates user-defined information.	2-639
	ResetUnit	Restart Unit	Restarts a CPU Bus Unit or Special I/O Unit.	2-641
	GetNTPStatus	Read NTP Status	Reads the NTP status.	2-645
Communications Instructions	ExecPMCR	Protocol Macro	Requests execution of a communications sequence (protocol data) registered in a Serial Communications Unit (unit version 2.2 or later).	2-648
	SerialSend	SCU Send Serial	Sends data in No-protocol Mode from a serial port on a Serial Communications Unit.	2-658
	SerialRcv	SCU Receive Serial	Receives data in No-protocol Mode from a serial port on a Serial Communications Unit.	2-665
	SendCmd	Send Command	Uses a serial gateway and sends a command to a Serial Communications Unit. Or, sends an explicit command to a DeviceNet Unit.	2-674
	CIPOpen	Open CIP Class 3 Connection	Opens a CIP class 3 connection with the specified remote node.	2-684
	CIPRead	Read Variable Class 3 Explicit	Uses a class 3 explicit message to read the value of a variable in another Controller on a CIP network.	2-692
	CIPWrite	Write Variable Class 3 Explicit	Uses a class 3 explicit message to write the value of a variable in another Controller on a CIP network.	2-696
	CIPSend	Send Explicit Message Class 3	Sends a class 3 CIP message to a specified device on a CIP network.	2-701
	CIPClose	Close CIP Class 3 Connection	Closes the CIP class 3 connection to the specified handle.	2-704

Type	Instruction	Name	Function	Page
Communications Instructions	CIPUCMMRead	Read Variable UCMM Explicit	Uses a UCMM explicit message to read the value of a variable in another Controller on the specified CIP network.	2-706
	CIPUCMMWrite	Write Variable UCMM Explicit	Uses a UCMM explicit message to write the value of a variable in another Controller on a CIP network.	2-710
	CIPUCMMSend	Send Explicit Message UCMM	Sends a UCMM CIP message to a specified device on a CIP network.	2-716
	EC_CoESDOWrite	Write EtherCAT CoE SDO	Writes a value to a CoE object of a specified slave on the EtherCAT network.	2-726
	EC_CoESDORead	Read EtherCAT CoE SDO	Reads a value from a CoE object of a specified slave.	2-729
	EC_StartMon	Start EtherCAT Packet Monitor	Starts packet monitoring of EtherCAT communications.	2-734
	EC_StopMon	Stop EtherCAT Packet Monitor	Stops execution of packet monitoring.	2-740
	EC_SaveMon	Save EtherCAT Packets	Saves EtherCAT communications packet data to an internal file in the main memory of the CPU Unit.	2-742
	EC_CopyMon	Transfer EtherCAT Packets	Transfers packet data in an internal file in the main memory of the CPU Unit to a SD Memory Card.	2-744
	EC_DisconnectSlave	Disconnect EtherCAT Slave	Disconnects the specified slave from the network.	2-746
	EC_ConnectSlave	Connect EtherCAT Slave	Connects the specified slave to the EtherCAT network.	2-752
	SktUDPCreate	Create UDP Socket	Creates a UDP socket request to open a servo port for the built-in EtherNet/IP.	2-754
	SktUDPRcv	UDP Socket Receive	Reads the data from the receive buffer for a UDP socket for the built-in EtherNet/IP.	2-761
	SktUDPSend	UDP Socket Send	Sends data from a UDP port for the built-in EtherNet/IP.	2-764
	SktTCPAccept	Accept TCP Socket	Requests accepting a TCP socket for the built-in EtherNet/IP.	2-767
	SktTCPConnect	Connect TCP Socket	Connects to a remote TCP port from the built-in EtherNet/IP.	2-770
	SktTCPRcv	TCP Socket Receive	Reads the data from the receive buffer for a TCP socket for the built-in EtherNet/IP.	2-777
	SktTCPSend	TCP Socket Send	Sends data from a TCP port for the built-in EtherNet/IP.	2-780
	SktGetTCPStatus	Read TCP Socket Status	Reads the status of a TCP socket.	2-783
	SktClose	Close TCP/UDP Socket	Closes the specified TCP or UDP socket for the built-in EtherNet/IP.	2-786
	SktClearBuf	Clear TCP/UDP Socket Receive Buffer	Clears the receive buffer for the specified TCP or UDP socket for the built-in EtherNet/IP.	2-789

Type	Instruction	Name	Function	Page
SD Memory Card Instructions	FileWriteVar	Write Variable to File	Writes the value of a variable to the specified file in the SD Memory Card. The value is written in binary format.	2-794
	FileReadVar	Read Variable from File	Reads the contents of the specified file on the SD Memory Card as binary data and writes it to a variable.	2-799
	FileOpen	Open File	Opens the specified file in the SD Memory Card.	2-803
	FileClose	Close File	Closes the specified file in the SD Memory Card.	2-806
	FileSeek	Seek File	Sets a file position indicator in the specified file in the SD Memory Card.	2-809
	FileRead	Read File	Reads the data from the specified file in the SD Memory Card.	2-812
	FileWrite	Write File	Writes data to the specified file in the SD Memory Card.	2-819
	FileGets	Get Text String	Reads a text string of one line from the specified file in the SD Memory Card.	2-826
	FilePuts	Put Text String	Writes a text string to the specified file in the SD Memory Card.	2-833
	FileCopy	Copy File	Copies the specified file in the SD Memory Card.	2-840
	FileRemove	Delete File	Deletes the specified file from the SD Memory Card.	2-848
	FileRename	Change File Name	Changes the name of the specified file or directory in the SD Memory Card.	2-852
	DirCreate	Create Directory	Creates a directory with the specified name in the SD Memory Card.	2-857
	DirRemove	Delete Directory	Deletes the specified directory from the SD Memory Card.	2-860
Other Instructions	ReadNbit_**	N-bit Read Group	Reads zero or more bits from a bit string.	2-864
	WriteNbit_**	N-bit Write Group	Writes zero or more bits to a bit string.	2-866
	ChkRange	Check Subrange Variable	Determines if the value of a variable is within the valid range of the range type specification.	2-868
	GetMyTaskStatus	Read Current Task Status	Reads the status of the current task.	2-870
	Task_IsActive	Determine Task Status	Determines if the specified task is currently in execution.	2-873
	Lock	Lock Tasks	Starts an exclusive lock between tasks. Execution of any other task with a lock region with the same lock number is disabled.	2-875
	Unlock	Unlock Tasks	Stops an exclusive lock between tasks.	2-875
	Get**Clk	Get Clock Pulse Group	Outputs a clock pulse at the specified cycle.	2-880
	Get**Cnt	Get Incrementing Free-running Counter Group	Gets the values of free-running counters of the specified cycle.	2-881

- Refer to the *NJ-series Motion Control Instructions Reference Manual* (Cat. No. W508) for the specifications of the motion control instructions.
- Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504) for the specifications of the simulation instructions.

2

Instruction Descriptions

This section describes the specifications of the instructions that you can use with NJ-series Controllers.

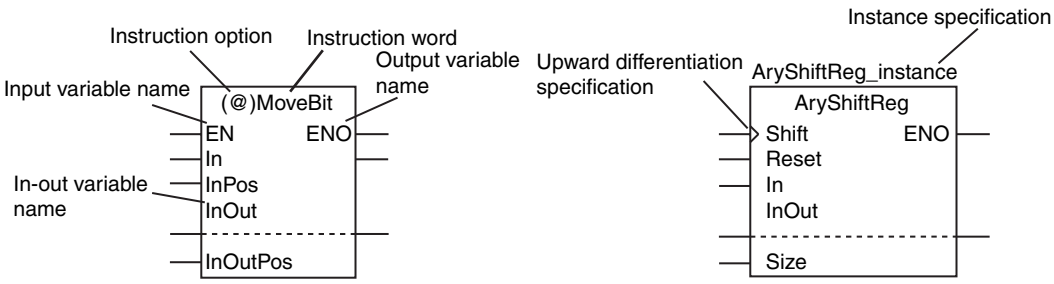
Using this Section 2-2
Ladder Diagram Instructions 2-13
ST Statement Instructions 2-23
Sequence Input Instructions 2-39
Sequence Output Instructions 2-45
Sequence Control Instructions 2-59
Comparison Instructions 2-83
Timer Instructions 2-115
Counter Instructions 2-133
Math Instructions 2-151
BCD Conversion Instructions 2-211
Data Type Conversion Instructions 2-231
Bit String Processing Instructions 2-285
Selection Instructions 2-297
Data Movement Instructions 2-317
Shift Instructions 2-351
Conversion Instructions 2-367
Stack and Table Instructions 2-465
FCS Instructions 2-503
Text String Instructions 2-519
Time and Time of Day Instructions 2-543
System Control Instructions 2-601
Communications Instructions 2-647
SD Memory Card Instructions 2-793
Other Instructions 2-863

Using this Section

The notation used to describe instructions in this section is explained below.

Items

The following items are provided.

Item	Description
Instruction	The instruction word is given. Example: MoveBit
Name	The name of the instruction is given. Example: Move Bit
FB/FUN	Whether the instruction is a function block (FB) instruction or a function (FUN) instruction is given. You can call FB instructions only from programs and function blocks. You can call FUN instructions from programs, function blocks, and functions.
Graphic expression	The figure that represents the instruction in a ladder diagram is given. **Example for a FUN Instruction** **Example for a FB Instruction**  The instruction option, upward differentiation specification, and instance specification are described below. Instruction option: Support for the instruction option is indicated by "(@)" before the FUN instruction. If support for the instruction option is indicated, you can place "@" before the instruction word to specify upward differentiation. Also, you can place "%" before the instruction word to specify downward differentiation. An instruction for which upward differentiation is specified is executed when the value of the EN input variable was FALSE in the previous task period and is TRUE in the current task period. An instruction for which downward differentiation is specified is executed when the value of the EN input variable was TRUE in the previous task period and is FALSE in the current task period. Upward differentiation specification: This is indicated by the arrow pointing into the instruction at the entry point of the input variable. Instructions with this specification operate as upwardly differentiated instructions. Instance specification: An instance of an instruction is indicated by "XX_instance" above an FB instruction. You must assign an instance name to any instance of an instruction that you specify.

Item	Description
ST expression	The notation that represents the instruction in ST is given. There are two ways that you can use to code an instruction in ST. These are described below. 1. Directly Specifying the Correspondence between the Parameters and the Input, Output, and In-Out Variables Example: MoveBit(In:=abc, InPos:=def, InOut:=ghi, InOutPos:=jkl); 2. Specifying Only the Parameters and Omitting the Input, Output, and In-Out Variables Example: MoveBit(In, InPos, InOut, InOutPos); Method 2 is used in this section. You must assign an instance name to any instruction that is given as "XX_instance(*variable_name*)." Example: TON_instance (In, PT, Q, ET);
Variables	• Name The input variables, output variables, and in-out variables are given. Example: *In1* However, variables that are used by many instructions are not given on the pages that describe individual instructions. The following eight variables are commonly used. The specifications of these variables are given later. (*EN*, *ENO*, *Execute*, *Done*, *Busy*, *Error*, *ErrorID*, and *ErrorIDEx*) • Meaning The name of the variable is given. Example: Up-counter • I/O Whether the variable is an input variable, output variable, or in-out variable is given. • Description The meaning of the variable and any restrictions are given. • Valid range The range that the variable can take is given. "Depends on data type" indicates that the valid range of the variable depends on the data type that you use. The valid ranges of the data types are given later in this section. • Unit The unit of the value that is specified with the variable is given. "---" indicates that there is no unit. Example: Bytes • Default The specified default value is automatically used for the variable if you do not assign a parameter to the instruction before it is executed. "---" indicates the following: Input variables: The default value of the data type of the input variable is assigned. The default values of the data types are given later in this section. If the input variable is a structure, the default value is given in the specifications of the structure in the description of the function of the instruction. Output variables: Default values are not set. In-out variables: Default values are not set. • Data type The data type of the variable is given. The use of enumerations, arrays, structures, and unions is also given.
Function	The function of the instruction is described. Variable names are given in italic text. Example: *In1* Array names are followed by "[]". Example: *InOut[]*
Related System-defined Variables	The system-defined variables that are related to the instruction are given. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details on system-defined variables.
Related Semi-user-defined Variables	The semi-user-defined variables and variable names that are related to the instruction are given. Refer to the specified manuals for details on semi-user-defined variables.

Item	Description
Additional Information	Additional information on the function of the instruction is provided. This includes related instructions and helpful information for application of the instruction.
Precautions for Correct Use	Precautions for application of the instruction are given. The conditions under which errors occur for the instruction are also given here.
Sample Programming	Short samples of how to use the instruction in an application program are provided. The ladder diagram and ST for the same process are shown.

Common Variables

The specifications of variables that are used for many instructions (*EN, ENO, Execute, Done, Busy, Error, ErrorID,* and *ErrorIDEx*) are described below. These variables are not described in the tables of variables for individual instructions. Check the graphic or ST expression for the instruction to see if an instruction uses these variables.

EN

EN is an input variable that gives the execution condition for a FUN instruction.
When you use a FUN instruction in a ladder diagram, connect the execution condition to *EN*.

Name	Meaning	I/O	Description	Data type	Valid range	Default
EN	Enable (Execution Condition)	Input	TRUE: Instruction is executed.* FALSE: Instruction is not executed.	BOOL	TRUE or FALSE	TRUE

* If upward differentiation (@) is specified as an instruction option, the execution condition is when the value of *EN* changes from FALSE to TRUE. If downward differentiation (%) is specified as an instruction option, the execution condition is when the value of *EN* changes from TRUE to FALSE.

- FB instructions do not have an *EN* input variable.
- When you call a FUN instruction from structured text, omit the *EN* input variable. The *EN* input variable is not required in structured text because the execution condition for the instruction is determined by the operation sequence.

ENO

The *ENO* output variable passes the execution to the next instruction in a ladder diagram. Normally, when instruction execution is completed, the value of *ENO* changes to TRUE. Execution of the next instruction is then started.

Name	Meaning	I/O	Description	Data type	Valid range	Default
ENO	Enable Output	Output	TRUE: Normal end.* FALSE: Error end, execution in progress, or execution condition not met.	BOOL	TRUE or FALSE	---

* *ENO* is TRUE only while the execution condition is met. The value of *ENO* changes to FALSE when the execution condition is no longer met after a normal end.

- Most FUN instructions and FB instructions have *ENO* output variables. There are, however, some instructions that do not have an *ENO* output variable.
- Omit the *ENO* output variable in structured text. The *ENO* output variable is not required in structured text because the execution condition for the next instruction is determined by the operation sequence.

Execute, *Done*, and *Busy*

Execute is an input variable that gives the execution condition for some FB instructions.
Instruction execution starts when *Execute* changes to TRUE. After *Execute* changes to TRUE, execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the instruction execution time exceeds the task period.

Done is an output variable that shows the completion of execution for some FB instructions.
Busy is an output variable that shows that instruction execution is in progress for some FB instructions.

Name	Meaning	I/O	Description	Data type	Valid range	Initial value
Execute	Execute	Input	TRUE: Instruction is executed.*1 FALSE: Instruction is not executed.*2	BOOL	TRUE or FALSE	FALSE
Done	Done	Output	TRUE: Normal end.*3*4 FALSE: Error end, execution in progress, or execution condition not met.	BOOL	TRUE or FALSE	---
Busy	Busy		TRUE: Execution processing is in progress. FALSE: Execution processing is not in progress.			

*1 If the value of *Execute* is already TRUE when Controller operation starts, the instruction is not executed. To execute the instruction in that case, first change the value of *Execute* to FALSE.

*2 Processing is completed to the end even if *Execute* changes to FALSE during execution.

*3 The value of *Done* changes to FALSE when the execution condition is no longer met after a normal end.

*4 If the execution condition is no longer met when a normal end occurs, the value of *Done* is TRUE for one task period and it then changes to FALSE.

Error, ErrorID, and *ErrorIDEx*

Error, *ErrorID*, and *ErrorIDEx* are output variables that show that an error occurred in the execution of some FB instructions.

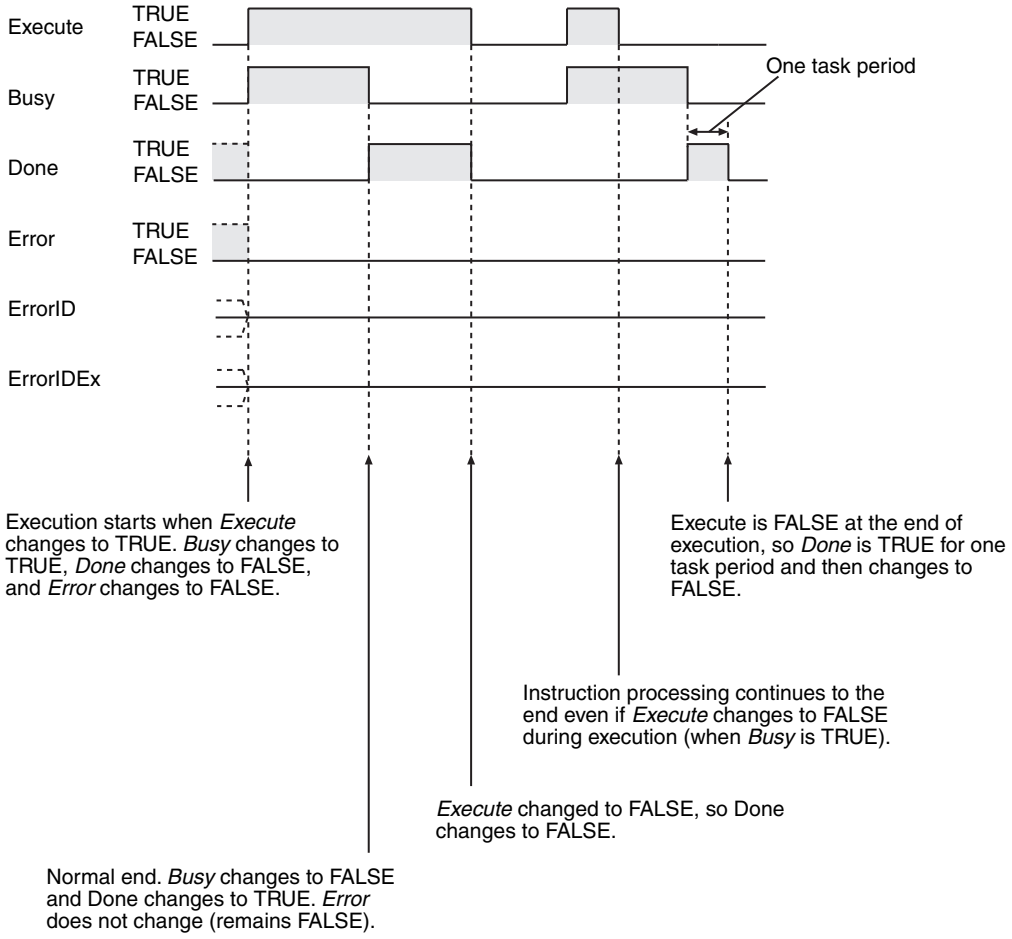
Name	Meaning	I/O	Description	Data type	Valid range	Initial value
Error	Error	Output	TRUE: Error end.*1*2 FALSE: Normal end, execution in progress, or execution condition not met.	BOOL	TRUE or FALSE	---
ErrorID	Error code		This is the error ID for an error end. The value is WORD#16#0 for a normal end.	WORD	Depends on the instruction.	
ErrorIDEx	Expansion error code		This is the error ID for an Expansion Unit Hardware Error. The value is DWORD#16#0 for a normal end.	DWORD		

*1 The value of *Error* changes to FALSE when the execution condition is no longer met after an error end.

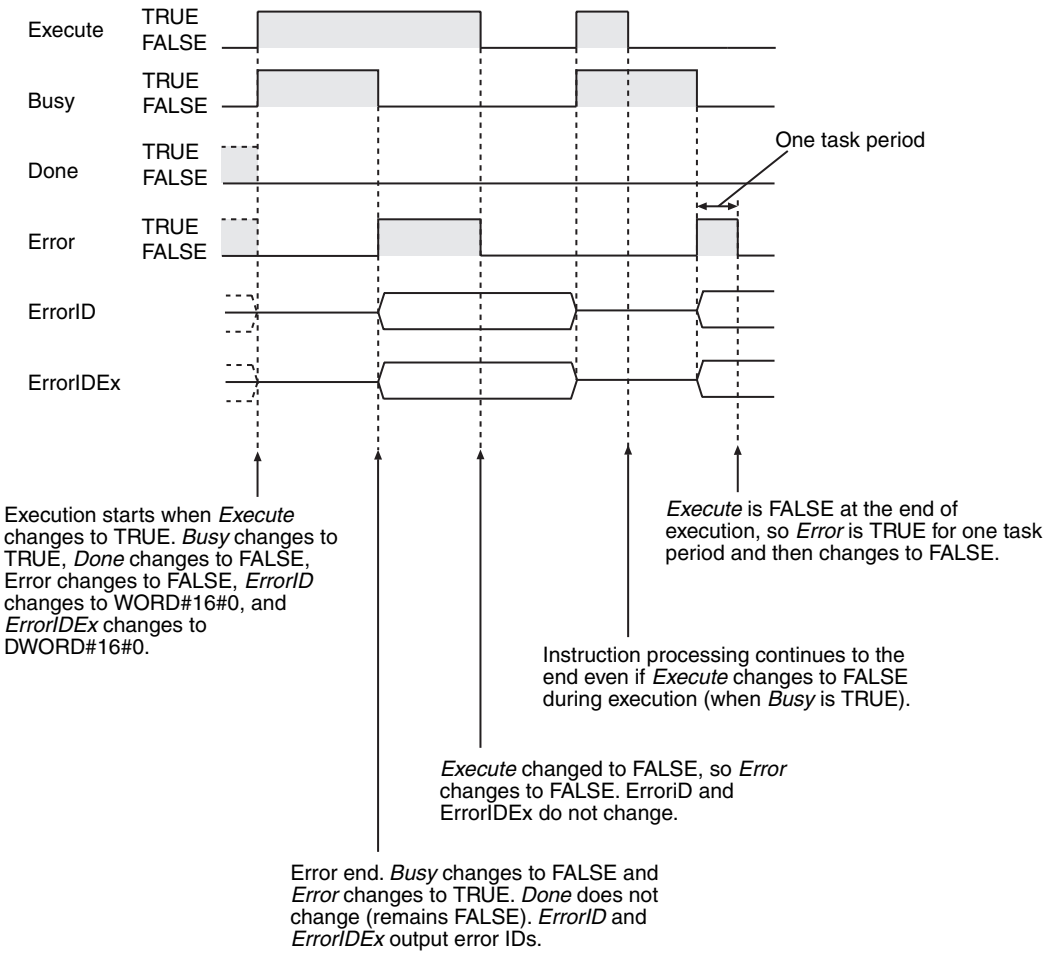
*2 If the execution condition is no longer met when an error end occurs, the value of *Error* is TRUE for one task period and it then changes to FALSE.

Timing charts are provided below for *Execute*, *Done*, *Busy*, *Error*, *ErrorID*, and *ErrorIDEx*.

● Normal End



● Error End



Valid Ranges and Default Values of Variables

The valid range of a variable indicates the range of values that variable can take. The default value of a variable indicates the value that is assigned to an input variable when the instruction is executed without a parameter assigned to the input variable. These values are defined for each data type. If specific values are not given for an instruction, then the valid ranges and default values of the data types are applied. These variables are indicated by “depends on data type” in the valid range column and by “---” in the input variable default column. The valid ranges and default values of the data types are given in the following tables.

Classification	Data type	Valid range	Default
Boolean	BOOL	TRUE or FALSE	FALSE
Bit string	BYTE	BYTE#16#00 to FF	BYTE#16#00
	WORD	WORD#16#0000 to FFFF	WORD#16#0000
	DWORD	DWORD#16#00000000 to FFFFFFFF	DWORD#16#0000_0000
	LWORD	LWORD#16#0000000000000000 to FFFFFFFFFFFFFFFF	LWORD#16#0000_0000_0000_0000

Classification	Data type	Valid range	Default
Integers	USINT	USINT#0 to +255	USINT#0
	UINT	UINT#0 to +65535	UINT#0
	UDINT	UDINT#0 to +4294967295	UDINT#0
	ULINT	ULINT#0 to +18446744073709551615	ULINT#0
	SINT	SINT#−128 to +127	SINT#0
	INT	INT#−32768 to +32767	INT#0
	DINT	DINT#−2147483648 to +2147483647	DINT#0
	LINT	LINT#−9223372036854775808 to +9223372036854775807	LINT#0
Real numbers	REAL	REAL#−3.402823e+38 to −1.175494e-38, 0, +1.175494e-38 to +3.402823e+38, +∞/−∞	REAL#0
	LREAL	LREAL#−1.79769313486231e+308 to −2.22507385850720e-308, 0, +2.22507385850720e−308 to +1.79769313486231e+308, +∞/−∞	LREAL#0
Times, durations, dates, and text strings	TIME	T#−9223372036854.775808ms (T#−106751d_23h_47m_16s_854.775808ms) to T#9223372036854.775807ms (T#+106751d_23h_47m_16s_854.775807ms)	T#0s
	DATE	D#1970-01-01 to D#2106-02-06 (January 1, 1970 to February 6, 2106)	D#1970-01-01
	TOD	TOD#00:00:00.000000000 to TOD#23:59:59.999999999 (0:00 and 0.000000000 to 23:59 and 59.999999999 seconds)	TOD#00:00:00.000000000
	DT	DT#1970-01-01-00:00:00.000000000 to DT#2106-02-06-23:59:59.999999999 (0:00 and 0.000000000 on January 1, 1970 to 23:59 and 59.999999999 seconds on February 6, 2106)	DT#1970-01-01-00:00:00.000000000
	STRING	Character code: UTF-8 0 to 1,986 bytes (1,985 single-byte alphanumeric characters plus the final NULL character)	''

Derivative Data Types (Enumerations, Structures, and Unions)

Variables that use derivative data types (enumerations, structures, and unions) are specified as such in the tables of variable data types. The notation is described below.

Enumerations

The data type for an enumerated variable is given within the table. The following is an example. Here, the data type of the *Out* variable is enumerated type _eDAYOFWEEK. The enumerators are described in the description of the function of the instruction.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK		OK	
Out	Refer to *Function* for the enumerators of the enumerated type _eDAYOFWEEK.																			

Structures and Unions

The data type for a structure or union variable is given within the table. The following is an example. Here, the data type of the *In1* variable is structure _sPORT. Details on the members of a structure or union are given in the description of the function of the instruction.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	Refer to *Function* for details on the structure _sPORT.																			

The tables also indicate any variables for which you can specify a structure, a structure member, a union, or a union member as the parameter.
In the following example, you can specify a parameter with a basic data type, or you can specify a structure, a structure member, a union, or a union member for the *In1* variable. To specify a structure or union, specify only the structure or the union as the parameter. To specify a structure member or a union member, specify the member as the parameter.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	A structure, structure member, union, or union member can also be specified.																			

Array Specifications

Array variable names are followed by "[]" and "(array)" is specified. For these variables, specify an element of the array (i.e., specify the subscript) as the parameter.
An example is shown below. Here, the table shows that *In1[]* is a BYTE array.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)		OK																		

The data type table indicates the arrays for which structures and unions can be used as elements, as shown in the following example. For these variables, specify an element of the array (i.e., specify the subscript) as the parameter.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	Arrays of structures or unions can also be specified.																			

The table indicates any variables for which you can specify either an array or an array element as the parameter.
In the following example, you can specify a parameter with a basic data type, or you can specify an array or an array element. To specify an array, specify only the array as the parameter. To specify an array element, specify an element of the array (i.e., specify the subscript) as the parameter.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	An array or array element can also be specified.																			

Others

Errors Detected for All Instructions

The errors that can occur for an instruction are given in the *Precautions for Correct Use* section. The following three errors, however, can be detected for any instruction. They are not listed in the *Precautions for Correct Use* sections.

- Reading or writing elements that exceed the range of an array variable.
  Example: Setting *a[4]* for an input variable for the array variable *a[0..3]*.
- Passing parameters that are not variables to instructions for which array variables are defined for input, output, or in-out variables.
- Assigning a text string that is longer than the defined number of bytes to a STRING variable.
- Assigning a text string that does not end in a NULL character to a STRING variable.

- Assigning a text string that has character code error to a STRING variable.
- Dividing an integer variable by 0.

Precautions for All Instructions

The amount of processing that is required for some instructions depends on the parameters that you connect. If there is too much processing, the instruction execution time increases and the task period may be exceeded. This will result in a Task Period Exceeded error. Adjust the amount of processing to a suitable amount.

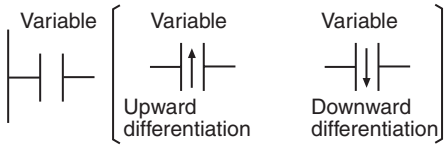
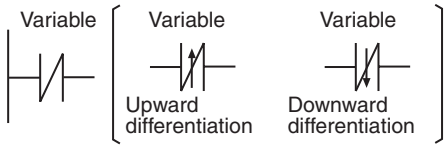
Ladder Diagram Instructions

Instruction	Name	Page
LD and LDN	Load/ Load NOT	2-14
AND and ANDN	AND/ AND NOT	2-16
OR and ORN	OR/ OR NOT	2-18
Out and OutNot	Output/ Output NOT	2-20

LD and LDN

LD: Reads the value of a BOOL variable.

LDN: Reads the inverse of the value of a BOOL variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LD	Load	---		None
LDN	Load NOT	---		None

Variables

None

Function

● LD

The LD instruction reads the value of the specified BOOL variable and outputs it to the next instruction. If the value of the specified variable is TRUE, then TRUE is output. If the value is FALSE, then FALSE is output. Use the LD instruction for the first NO bit from the bus bar or for the first NO bit of a logic block.

● LDN

The LD instruction reads the inverse of the value of the specified BOOL variable and outputs it to the next instruction. If the value of the specified variable is TRUE, then FALSE is output. If the value is FALSE, then TRUE is output. Use the LDN instruction for the first NC bit from the bus bar or for the first NC bit of a logic block.

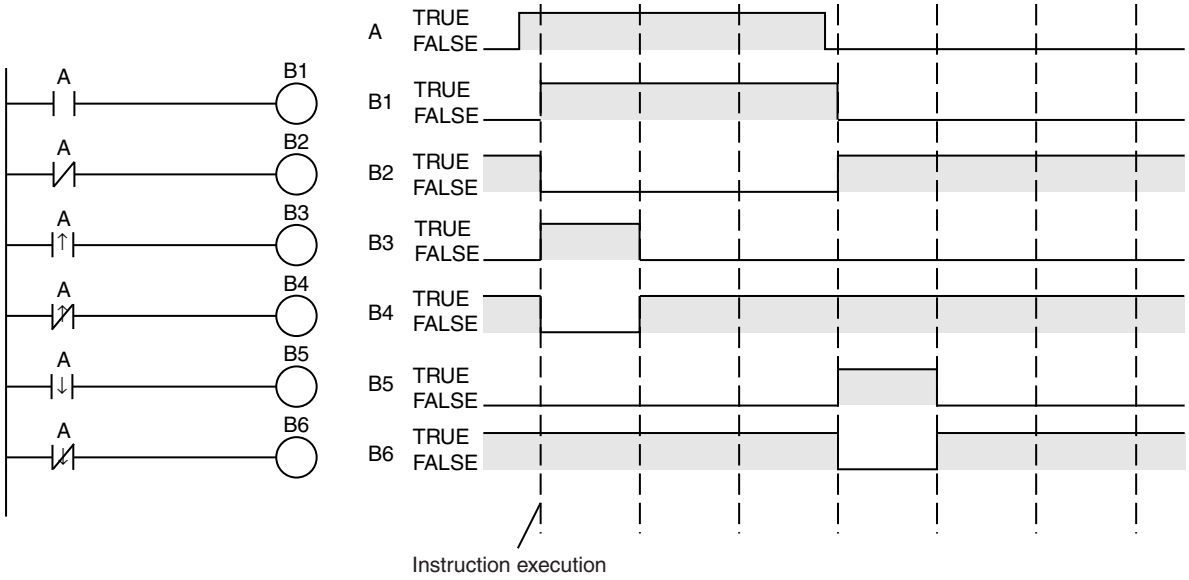
The operation is as shown below if you do not specify upward or downward differentiation.

Instruction	Value of variable	Output value
LD	TRUE	TRUE
	FALSE	FALSE
LDN	TRUE	FALSE
	FALSE	TRUE

If you specify upward or downward differentiation, the operation depends on the following: the value of the variable the last time the instruction was executed and the current value of the variable. This is shown below.

Instruction	Differentiation specification	Value of variable at last execution and current value of variable	Output value
LD	Upward differentiation	FALSE at the last execution → Currently TRUE	TRUE
		Other than the above.	FALSE
	Downward differentiation	TRUE at the last execution → Currently FALSE	TRUE
		Other than the above.	FALSE
LDN	Upward differentiation	FALSE at the last execution → Currently TRUE	FALSE
		Other than the above.	TRUE
	Downward differentiation	TRUE at the last execution → Currently FALSE	FALSE
		Other than the above.	TRUE

The following figure shows a programming example and timing chart.



Precautions for Correct Use

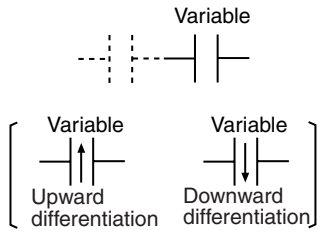
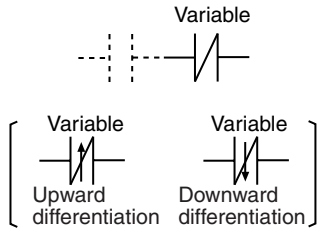
- An error occurs in the following case and the output value from the last execution is retained.
  - You specify an array element for the variable value and the element does not exist.

    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- Do not use these instructions as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.

AND and ANDN

AND: Takes the logical AND of the value of a BOOL variable and the execution condition.

ANDN: Takes the logical AND of the inverse of the value of a BOOL variable and the execution condition.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AND	AND	---		result:=vBool1 AND vBOOL2; result:=vBool1 & vBool2;
ANDN	AND NOT	---		result:=vBool1 AND NOT vBool2;

Variables

None

Function

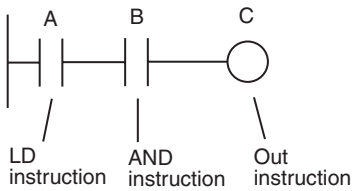
- **AND**

The AND instruction takes the logical AND of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the AND instruction for a NO bit connected in series with the previous instruction.

- **ANDN**

The ANDN instruction takes the logical AND of the inverse of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the ANDN instruction for a NC bit connected in series with the previous instruction.

The following figure shows a programming example of the AND instruction. It takes the logical AND of variable *A* and variable *B* and outputs it to variable *C*.



The operation is as shown below if you do not specify upward or downward differentiation.

Instruction	Combination of variable value and execution condition	Output value
AND	Variable value: TRUE Execution condition: TRUE	TRUE
	Other than the above.	FALSE
ANDN	Variable value: FALSE Execution condition: TRUE	TRUE
	Other than the above.	FALSE

If you specify upward or downward differentiation, the operation depends on the following: the value of the variable the last time the instruction was executed, the current value of the variable, and the execution condition. This is shown below.

Instruction	Differentiation specification	Combination of value of variable at last execution, current value of variable, and execution condition	Output value
AND	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: TRUE	TRUE
		Other than the above.	FALSE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: TRUE	TRUE
		Other than the above.	FALSE
ANDN	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: TRUE	FALSE
		Variable value: Ignored Execution condition: FALSE	
		Other than the above.	TRUE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: TRUE	FALSE
		Variable value: Ignored Execution condition: FALSE	
		Other than the above.	TRUE

Precautions for Correct Use

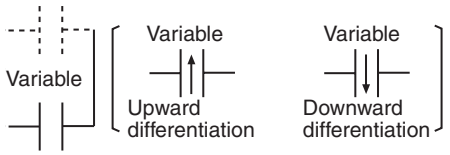
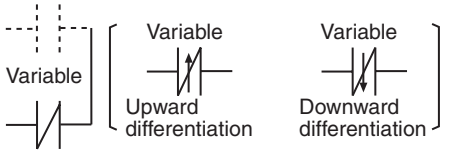
- An error occurs in the following case and the output value from the last execution is retained.
  - You specify an array element for the variable value and the element does not exist.

    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- Do not use these instructions as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- You cannot connect these instructions directly to the bus bar.

OR and ORN

OR: Takes the logical OR of the value of a BOOL variable and the execution condition.

ORN: Takes the logical OR of the inverse of the value of a BOOL variable and the execution condition.

Instruction	Name	FB/FUN	Graphic expression	ST expression
OR	OR	---		result:=vBool1 OR vBool2;
ORN	OR NOT	---		result:=vBool1 OR NOT vBool2;

Variables

None

Function

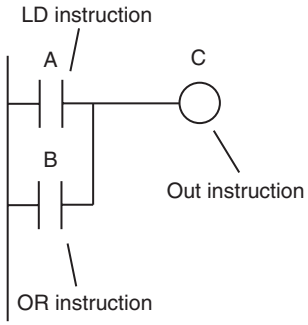
● OR

The OR instruction takes the logical OR of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the OR instruction for a NO bit connected in parallel with the previous instruction. Use the OR instruction to configure a logical OR between an NO bit and one of the following: a LD or LDN instruction connected directly to the bus bar, or the logic block starting with a LD or LDN instruction and ending with the instruction immediately before the OR instruction.

● ORN

The ORN instruction takes the logical OR of the inverse of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the ORN instruction for a NC bit connected in parallel with the previous instruction. Use the ORN instruction to configure a logical OR between an NC bit and one of the following: a LD or LDN instruction connected directly to the bus bar, or the logic block starting with a LD or LDN instruction and ending with the instruction immediately before the ORN instruction.

The following figure shows a programming example of the OR instruction. It takes the logical OR of variable *A* and variable *B* and outputs it to variable *C*.



The operation is as shown below if you do not specify upward or downward differentiation.

Instruction	Combination of variable value and execution condition	Output value
OR	Variable value: FALSE Execution condition: FALSE	FALSE
	Other than the above.	TRUE
ORN	Variable value: TRUE Execution condition: FALSE	FALSE
	Other than the above.	TRUE

If you specify upward or downward differentiation, the operation depends on the following: the value of the variable the last time the instruction was executed, the current value of the variable, and the execution condition. This is shown below.

Instruction	Differentiation specification	Combination of value of variable at last execution, current value of variable, and execution condition	Output value
OR	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: Ignored.	TRUE
		Variable value: Ignored Execution condition: TRUE	
		Other than the above.	FALSE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: Ignored.	TRUE
		Variable value: Ignored Execution condition: TRUE	
		Other than the above.	FALSE
ORN	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: FALSE	FALSE
		Other than the above.	TRUE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: FALSE	FALSE
		Other than the above.	TRUE

Precautions for Correct Use

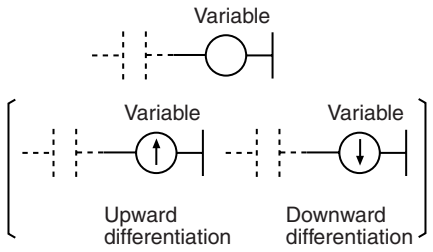
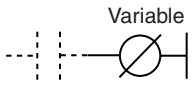
- An error occurs in the following case and the output value from the last execution is retained.
  - You specify an array element for the variable value and the element does not exist.

    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- Do not use these instructions as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.

Out and OutNot

Out: Takes the logical result from the previous instruction and outputs it to a BOOL variable.

OutNot: Takes the inverse of the logical result from the previous instruction and outputs it to a BOOL variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Out	Output	---		Variable:=(Logic expression up to previous instruction);
OutNot	Output NOT	---		Variable:=NOT(Logic expression up to previous instruction);

Variables

None

Function

● Out

The Out instruction takes the logical result from the previous instruction and outputs it to a specified BOOL variable.

The operation is as shown below if you do not specify upward or downward differentiation.

Logic processing result from previous instruction	Output
TRUE	TRUE
FALSE	FALSE

You can specify upward or downward differentiation for the Out instruction. If upward or downward differentiation is specified, the output value is determined by changes in the result of logic processing from the previous instruction between the last execution of the instruction and the current execution. The operation is according to the current logical result from the previous instruction, as shown in the following table.

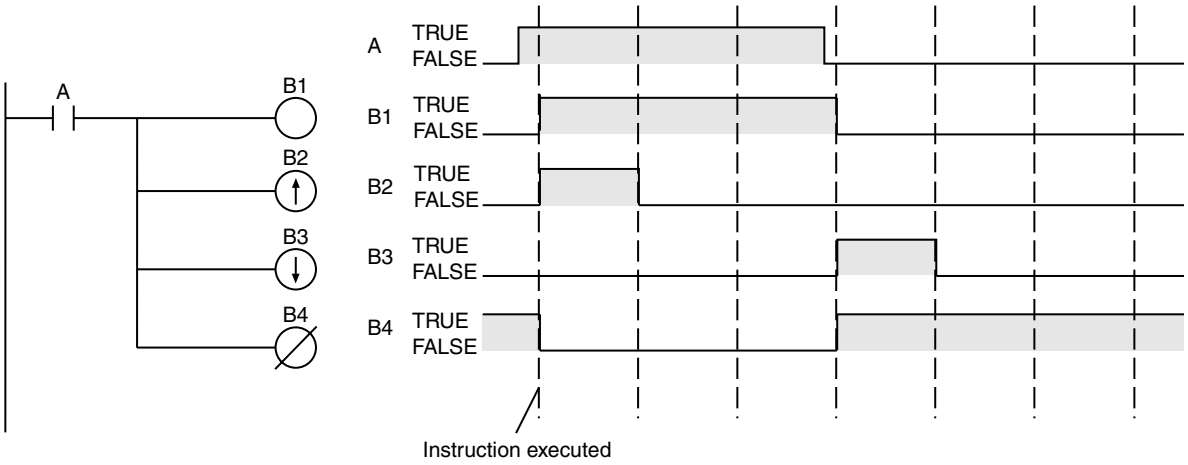
Differentiation specification	Results of logic processing from the previous execution and current execution	Output
Upward differentiation	FALSE at the last execution → Currently TRUE	TRUE
	Other than the above.	FALSE
Downward differentiation	TRUE at the last execution → Currently FALSE	TRUE
	Other than the above.	FALSE

● OutNot

The OutNot instruction takes the inverse of the logical result from the previous instruction and outputs it to a specified BOOL variable.

Logic processing result from previous instruction	Output
TRUE	FALSE
FALSE	TRUE

The following figure shows a programming example and timing chart.



Additional Information

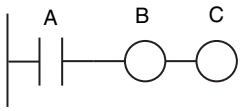
Differences between the Set and Reset Instructions and the Out and OutNot Instructions

- The Set and Reset instructions operate only when the input value changes to TRUE. They do not operate when the input value is FALSE. When the input value is FALSE, the output does not change.
- The Out and OutNot instructions affect the output whether the logical result of the previous instruction is TRUE or FALSE.

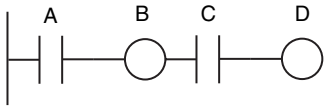
Precautions for Correct Use

- In the following case, an error occurs and nothing is output.
  - You specify an array element for the variable value and the element does not exist.

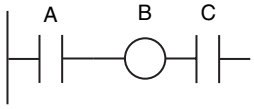
    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- The following connections are possible.
  - You can connect another Out instruction after an Out instruction.



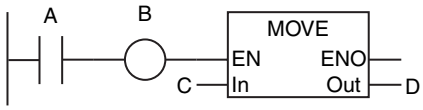
  - You can connect an LD instruction and Out instruction after an Out instruction.



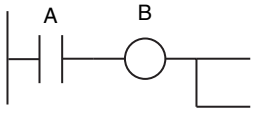
- The following connections are not possible.
  - You cannot connect only an LD instruction after an Out instruction.

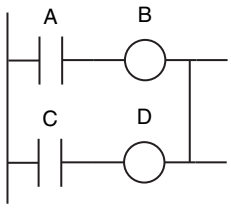


  - Functions and function blocks cannot be connected after an Out instruction.



  - Branches and joins cannot be used after Out instructions.





ST Statement Instructions

Instruction	Name	Page
IF	If	2-24
CASE	Case	2-28
WHILE	While	2-32
REPEAT	Repeat	2-34
RETURN	Return	2-36
FOR	Repeat Start	2-37
EXIT	Break Loop	2-38

IF

The IF construct uses the evaluation result of a specified condition expression to select one of two statements to execute.

Instruction	Name	FB/FUN	Graphic expression	ST expression

IF	If	---	None	IF *condition expression*
				THEN
				  statement;
				ELSIF *condition expression*
				THEN
				  statement;
				ELSE
				  statement;
				END_IF;

Variables

None

Function

The IF construct uses the evaluation result of a specified condition expression to select one of two statements to execute. Use a condition expression that evaluates to TRUE or FALSE.

Item used for condition expression	Example	Evaluation result
Logic expression	a>3	If the value of variable *a* is greater than 3, the result is TRUE. Otherwise, the result is FALSE.
	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
BOOL variable	abc	If the value of variable *abc* is TRUE, the result is TRUE. If it is FALSE, the result is FALSE.
BOOL constant	TRUE	TRUE
Function with a BOOL return value	FUN name	If the function returns TRUE, the result is TRUE. If it returns FALSE, the result is FALSE.

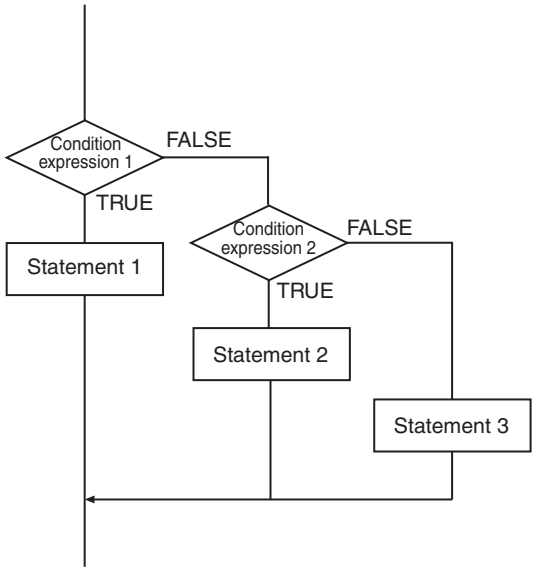
You can use the following operators in the logic expression.

Operator	Meaning	Example	Evaluation result
=	Equals	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
<>	Not equals	a<>b	If the values of variables *a* and *b* are not equal, the result is TRUE. Otherwise, the result is FALSE.
<	Comparison	a<b	If the value of variable *a* is less than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
<=		a<=b	If the value of variable *a* is less than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>		a>b	If the value of variable *a* is greater than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>=		a>=b	If the value of variable *a* is greater than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
AND (&)	Logical AND	a AND b a & b	The result is the logical AND of BOOL variables *a* and *b*.

Operator	Meaning	Example	Evaluation result
OR	Logical OR	a OR b	The result is the logical OR of BOOL variables *a* and *b*.
XOR	Exclusive OR	a XOR b	The result is the logical exclusive OR of BOOL variables *a* and *b*.
NOT	NOT	NOT a	The result is the NOT of BOOL variable *a*.

The flowchart in the following example shows the evaluation results for condition expressions 1 and 2. You can use more than one statement for each of statements 1 to 3.

IF *condition expression 1* THEN
    statement 1;
ELSIF *condition expression 2* THEN
    statement 2;
ELSE
    statement 3;
END_IF;



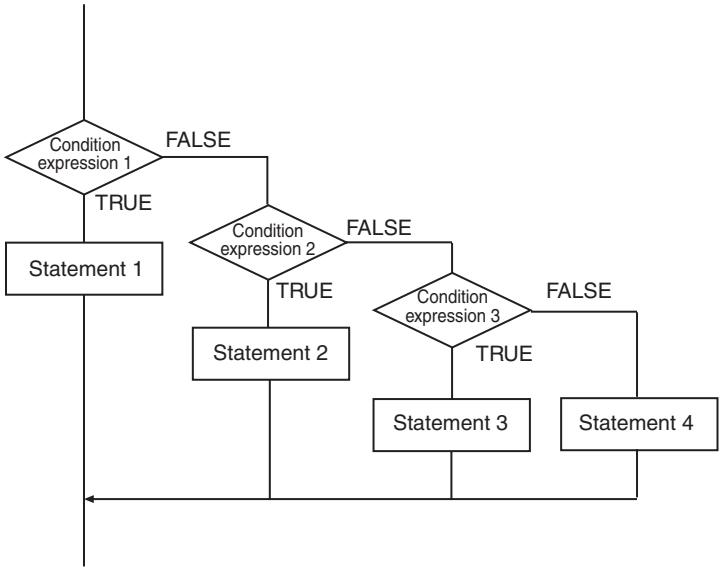
Additional Information

- You can use the IF construct to build a hierarchy. The following example executes statement 11 if the evaluation results of both condition expression 1 and condition expression 11 are TRUE.

IF *condition expression 1* THEN
    IF *condition expression 11* THEN
        statement 11;
    ELSIF *condition expression 12* THEN
        statement 12;
    ELSE
        statement 13;
    END_IF;
ELSIF *condition expression 2* THEN
    statement 2;
ELSE
    statement 3;
END_IF;

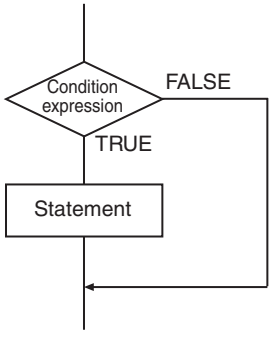
You can use ELSIF more than once. The following processing flow is for this example.

IF *condition expression 1* THEN
    statement 1;
ELSIF *condition expression 2* THEN
    statement 2;
ELSIF *condition expression 3* THEN
    statement 3;
ELSE
    statement 4;
END_IF;



- You do not use ELSIF if there is only one condition expression. You do not use ELSE if no processing is performed when none of the condition expressions are TRUE. The following processing flow is for this example.

IF condition expression THEN
    statement;
END_IF;



- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the IF construct as you do for the statements outside the IF construct. For example, you can use function block calls and FOR constructs.

Precautions for Correct Use

- You must always use IF and END_IF. They must be paired.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example assigns INT#0 to variable *def* if the value of variable *abc* is less than INT#0. It assigns INT#1 to variable *def* and INT#2 to variable *ghi* if the value of variable *abc* is INT#0. It assigns INT#3 to variable *def* if the value of variable *abc* is none of the above.

Variable	Data type	Initial value
abc	INT	0
def	INT	0
ghi	INT	0

```
IF (abc<INT#0) THEN
    def:=INT#0;
ELSIF (abc=INT#0) THEN
    def:=INT#1;
    ghi:=INT#2;
ELSE
    def:=INT#3;
END_IF;
```

CASE

You use the CASE construct to select the statement to execute based on the value of a specified integer expression.

Instruction	Name	FB/FUN	Graphic expression	ST expression

CASE	Case	---	None	CASE *integer expression*
				OF
				  value:
				  statement;
				  value:
				  statement;
				  .
				  .
				  .
				  ELSE
				  statement;
				END_CASE;

Variables

None

Function

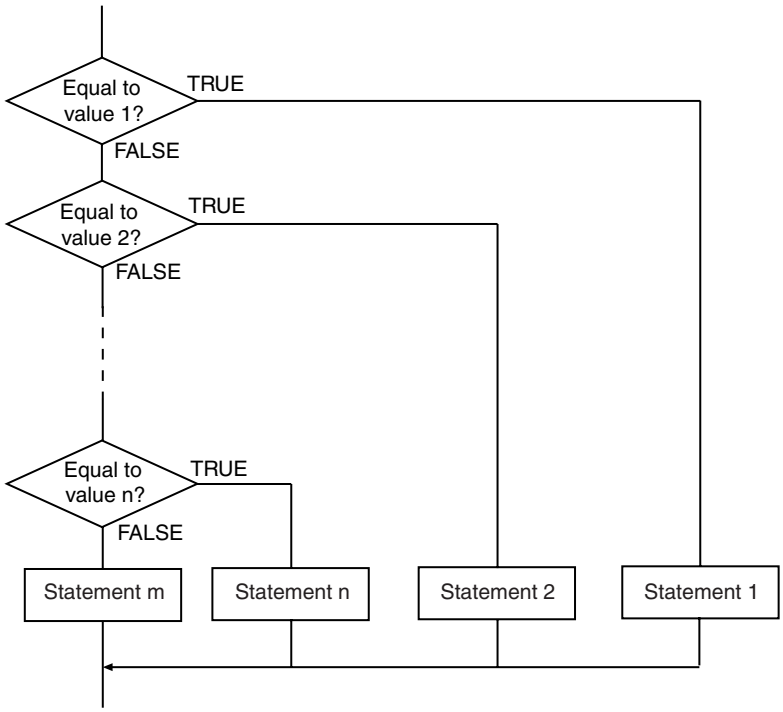
You use the CASE construct to select the statement to execute based on the value of a specified integer expression.

You can use any of the following as the integer expression and values.

	Allowed notation
Integer expression	Integer variable, integer constant, integer expression, or a function that returns an integer return value, enumeration variable, enumeration expression, or enumerator
Values	Integer constants

The flowchart in the following example shows the processing flow for an integer expression. You can use more than one statement for each of the statements.

CASE *integer expression* OF
  1 :
    statement 1;
  2 :
    statement 2;
  .
  .
  .
  n :
    statement n;
  ELSE
    statement m;
END_CASE;



Additional Information

- You can use the CASE construct to build a hierarchy. The following example executes statement 12 if the value of integer expression 1 is 1 and the value of integer expression 11 is 2.

```
CASE integer expression 1 OF
    1 :
        CASE integer expression 1 OF
        1 :
            statement 11;
        2 :
            statement 12;
        ELSE
            statement 1m;
        END_CASE;
    2 :
        statement 2;
    3 :
        statement 3;
    ELSE
        statement m;
END_CASE;
```

- You can use more than one value at the same time. Separate values with commas. The following example executes statement 1 if the value of the integer expression is either 1 or 2.

```
CASE integer expression 1 OF
    1,2 :
        statement 1;
    3 :
        statement 2;
    4 :
        statement 3;
    ELSE
        statement m;
END_CASE;
```

- You can use a range of consecutive values. Place two periods between the numbers to indicate consecutive values. The following example executes statement 1 if the value of the integer expression is between 10 and 15, inclusive.

```
CASE integer expression 1 OF
    10..15:
        statement 1;
    16:
        statement 2;
    17:
        statement 3;
    ELSE
        statement m;
END_CASE;
```

- You can omit ELSE. If you do, none of the statements is executed if none of the values is equal to the value of the integer expression.
- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the CASE construct as you do for the statements outside the CASE construct. For example, you can use function block calls and FOR constructs.
- The following is different in comparison to a C language *switch* statement. With a C language *switch* statement, all statements after a value that equals the integer expression are executed unless a *break* statement is used. With the CASE statement, only the statements that correspond directly to the value that equals the integer expression are executed. For example, in the following example, statements 1 to 3 are executed for the C language *switch* statement. Here, only statement 1 is executed for the CASE instruction.

C Language switch Statement

```
val=1;
switch val
{
        case 1:
    statement 1;
        case 2:
    statement 2;
        case 3:
    statement 3;
}
```

CASE Instruction

```
val:=1;
CASE val OF
    1:
        statement 1;
    2:
        statement 2;
    3:
        statement 3;
END_CASE;
```

Precautions for Correct Use

- You must always use CASE and END_CASE. They must be paired.
- The data types of the integer expression and values can be different.
- Each value can be given only once.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example assigns INT#10 to variable *def* if the value of variable *abc* is INT#1, INT#20 if the value of variable *abc* is INT#2, and INT#30 if the value of variable *abc* is INT#3. Otherwise, it assigns the value of variable *ghi* to variable *def*.

Variable	Data type	Initial value
abc	INT	0
def	INT	0
ghi	INT	0

```
CASE abc OF
    INT#1:
        def:=INT#10;
    INT#2:
        def:=INT#20;
    INT#3:
        def:=INT#30;
    ELSE
        def:=ghi;
END_CASE;
```

This example assigns INT#10 to variable *def* if the value of variable *abc* is INT#1, INT#20 if the value of variable *abc* is INT#2 or INT#5, and INT#30 if the value of variable *abc* is between INT#6 and INT#10, inclusive. Otherwise, it does nothing.

Variable	Data type	Initial value
abc	INT	0
def	INT	0

```
CASE abc OF
    INT#1:
        def:=INT#10;
    INT#2,INT#5:
        def:=INT#20;
    INT#6..INT#10:
        def:=INT#30;
END_CASE;
```

WHILE

The WHILE construct repeatedly executes a statement as long as the evaluation result of a specified condition expression is TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
WHILE	While	---	None	WHILE *condition expression* DO *statement*; END_WHILE;

Variables

None

Function

The WHILE construct repeatedly executes a statement as long as the evaluation result of a specified condition expression is TRUE. Use a condition expression that evaluates to TRUE or FALSE.

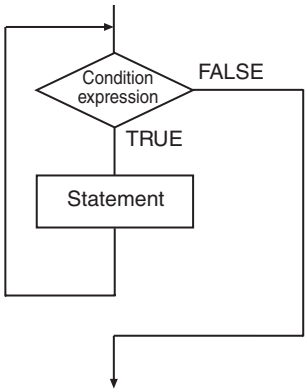
Item used for condition expression	Example	Evaluation result
Logic expression	a>3	If the value of variable *a* is greater than 3, the result is TRUE. Otherwise, the result is FALSE.
	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
BOOL variable	abc	If the value of variable *abc* is TRUE, the result is TRUE. If it is FALSE, the result is FALSE.
BOOL constant	TRUE	TRUE
Function with a BOOL return value	FUN name	If the function returns TRUE, the result is TRUE. If it returns FALSE, the result is FALSE.

You can use the following operators in the logic expression.

Operator	Meaning	Example	Evaluation result
=	Equals	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
<>	Not equals	a<>b	If the values of variables *a* and *b* are not equal, the result is TRUE. Otherwise, the result is FALSE.
<	Comparison	a<b	If the value of variable *a* is less than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
<=		a<=b	If the value of variable *a* is less than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>		a>b	If the value of variable *a* is greater than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>=		a>=b	If the value of variable *a* is greater than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
AND (&)	Logical AND	a AND b a & b	The result is the logical AND of BOOL variables *a* and *b*.
OR	Logical OR	a OR b	The result is the logical OR of BOOL variables *a* and *b*.
XOR	Exclusive OR	a XOR b	The result is the logical exclusive OR of BOOL variables *a* and *b*.
NOT	NOT	NOT a	The result is the NOT of BOOL variable *a*.

The following processing flow is for this example. You can use more than one statement.

WHILE *condition expression* DO
    statement;
END_WHILE;



Additional Information

- The statement is not executed even once if the condition expression is FALSE the first time it is evaluated.
- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the WHILE construct as you do for the statements outside the WHILE construct. For example, you can use function block calls and FOR constructs.

Precautions for Correct Use

- You must always use WHILE and END_WHILE. They must be paired.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example adds INT#7 to variable *abc* as long as the value of variable *abc* is less than or equal to INT#1000.

Variable	Data type	Initial value
abc	INT	0

```
abc:=INT#0;
WHILE abc<=INT#1000 DO
    abc:=abc+INT#7;
END_WHILE;
```

REPEAT

The REPEAT construct executes a statement once and then executes it repeatedly until a specified condition expression is TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
REPEAT	Repeat	---	None	REPEAT *statement*; UNTIL condition expression END_REPEAT;

Variables

None

Function

The REPEAT construct executes a statement once and then executes it repeatedly until a specified condition expression is TRUE. Use a condition expression that evaluates to TRUE or FALSE.

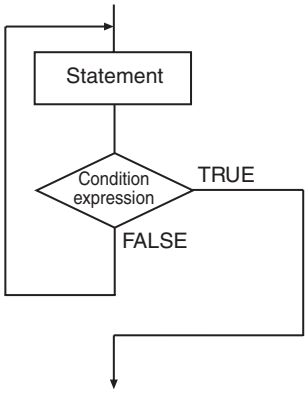
Item used for condition expression	Example	Evaluation result
Logic expression	a>3	If the value of variable *a* is greater than 3, the result is TRUE. Otherwise, the result is FALSE.
	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
BOOL variable	abc	If the value of variable *abc* is TRUE, the result is TRUE. If it is FALSE, the result is FALSE.
BOOL constant	TRUE	TRUE
Function with a BOOL return value	FUN name	If the function returns TRUE, the result is TRUE. If it returns FALSE, the result is FALSE.

You can use the following operators in the logic expression.

Operator	Meaning	Example	Evaluation result
=	Equals	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
<>	Not equals	a<>b	If the values of variables *a* and *b* are not equal, the result is TRUE. Otherwise, the result is FALSE.
<	Comparison	a<b	If the value of variable *a* is less than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
<=		a<=b	If the value of variable *a* is less than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>		a>b	If the value of variable *a* is greater than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>=		a>=b	If the value of variable *a* is greater than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
AND (&)	Logical AND	a AND b a & b	The result is the logical AND of BOOL variables *a* and *b*.
OR	Logical OR	a OR b	The result is the logical OR of BOOL variables *a* and *b*.
XOR	Exclusive OR	a XOR b	The result is the logical exclusive OR of BOOL variables *a* and *b*.
NOT	NOT	NOT a	The result is the NOT of BOOL variable *a*.

The following processing flow is for this example. You can use more than one statement.

```
REPEAT
    statement;
UNTIL condition expression
END_REPEAT;
```



Additional Information

- The statement is executed once before the condition expression is evaluated. Therefore, the statement is always executed at least once.
- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the REPEAT construct as you do for the statements outside the REPEAT construct. For example, you can use function block calls and FOR constructs.

Precautions for Correct Use

- You must always use REPEAT, UNTIL, and END_REPEAT. They must be used as a set.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example adds INT#1 to variable *abc* until the value of variable *abc* exceeds INT#10.

Variable	Data type	Initial value
abc	INT	0

```
abc:=INT#0;
REPEAT
    abc:=abc+INT#1;
UNTIL abc>INT#10
END_REPEAT;
```

RETURN

Refer to *RETURN* on page *2-61* in the Sequence Control Instructions for a description of this instruction.

FOR

Refer to *FOR and NEXT* on page *2-76* in the Sequence Control Instructions for a description of this instruction.

EXIT

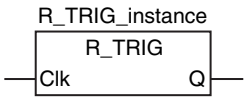
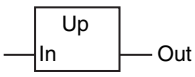
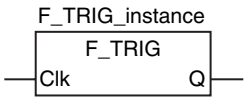
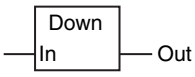
Refer to *BREAK* on page *2-81* in the Sequence Control Instructions for a description of this instruction. The BREAK ladder diagram instruction and the EXIT structured text instruction function in the same way.

Sequence Input Instructions

Instruction	Name	Page
R_TRIG (Up) and F_TRIG (Down)	Up Trigger/ Down Trigger	2-40
TestABit and TestABitN	Test A Bit/ Test A Bit NOT	2-43

R_TRIG (Up) and F_TRIG (Down)

R_TRIG (Up): Outputs TRUE for one task period only when the input signal changes to TRUE.
F_TRIG (Down): Outputs TRUE for one task period only when the input signal changes to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
R_TRIG	Up Trigger	FB		R_TRIG_instance(Clk, Q);
Up		FUN		None
F_TRIG	Down Trigger	FB		F_TRIG_instance(Clk, Q);
Down		FUN		None

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Clk, In	Input signal	Input	Input signal	Depends on data type.	---	---
Q, Out	Output signal	Output	Output signal	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Clk, In	OK																			
Q, Out	OK																			

Function

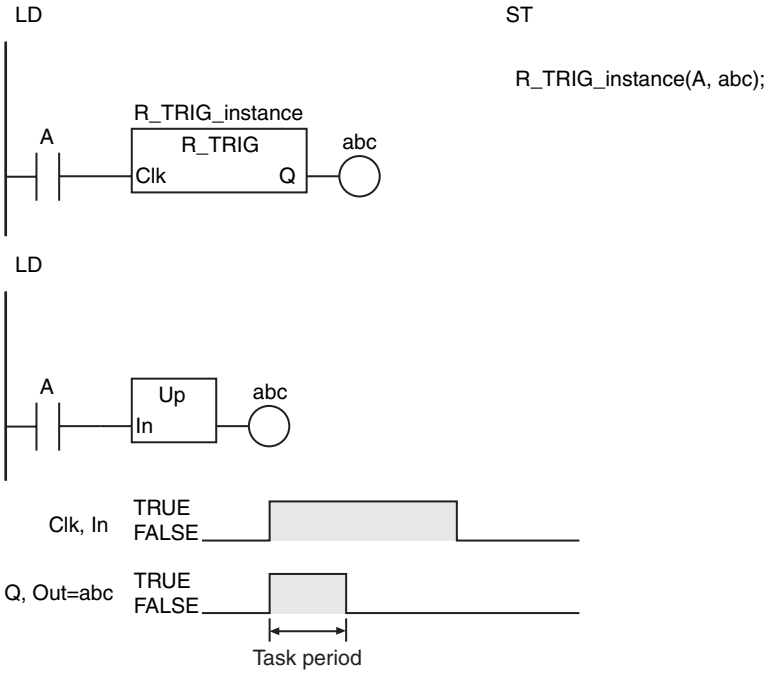
● R_TRIG

R_TRIG assigns TRUE to output signal *Q* for one task period only when input signal *Clk* changes to TRUE. Otherwise, the value of *Q* is FALSE. In the first task period in which this instruction is executed, the value of *Q* is FALSE regardless of the value of *Clk*. If the value of *Clk* is TRUE when the power supply is turned ON, the value of *Q* remains FALSE until the value of *Clk* changes to FALSE and then back to TRUE.

● Up

The functions of the R_TRIG instruction and the Up instruction are exactly the same. The *Clk* variable of the R_TRIG instruction corresponds to the *In* variable of the Up instruction. The *Q* variable corresponds to the *Out* variable.

The following figure shows a programming example and timing chart.



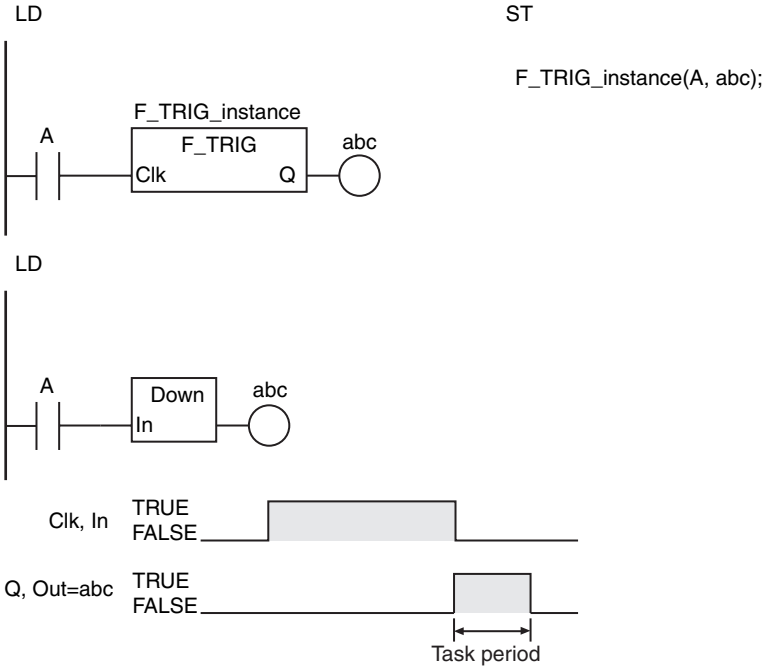
● F_TRIG

F_TRIG assigns TRUE to output signal *Q* for one task period only when input signal *Clk* changes to FALSE. Otherwise, the value of *Q* is FALSE. In the first task period in which this instruction is executed, the value of *Q* is FALSE regardless of the value of *Clk*. If the value of *Clk* is FALSE when the power supply is turned ON, the value of *Q* remains FALSE until the value of *Clk* changes to TRUE and then back to FALSE.

● Down

The functions of the F_TRIG instruction and the Down instruction are exactly the same. The *Clk* variable of the F_TRIG instruction corresponds to the *In* variable of the Down instruction. The *Q* variable corresponds to the *Out* variable.

The following figure shows a programming example and timing chart.



Precautions for Correct Use

- Detection of upward or downward differentiation depends on differences between the current value of *Clk* or *In* and the value the last time the instruction was executed. Caution is required when using the JMP instruction or other times that the instruction is not executed every task period.
- If power is interrupted, the value of *Clk* or *In* is not detected as FALSE. The value of *Clk* or *In* is detected as FALSE only if the instruction evaluates the value of *Clk* or *In* while *Clk* or *In* is FALSE.

TestABit and TestABitN

TestABit: Outputs the value of the specified bit in a bit string.

TestABitN: Outputs the inverse of the value of the specified bit in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TestABit	Test A Bit	FUN	(@)TestABit EN In Pos Out	Out:=TestABit (In, Pos);
TestABitN	Test A Bit NOT	FUN	(@)TestABitN EN In Pos Out	Out:=TestABitN (In, Pos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Bit string	Input	Bit string	Depends on data type.	---	*
Pos	Bit position		Specified bit position	0 to No. of bits in *In* – 1		0
Out	Bit value	Output	TestABit Value of specified bit TestABitN Inverse of value of specified bit	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Pos						OK														
Out	OK																			

Function

● TestABit

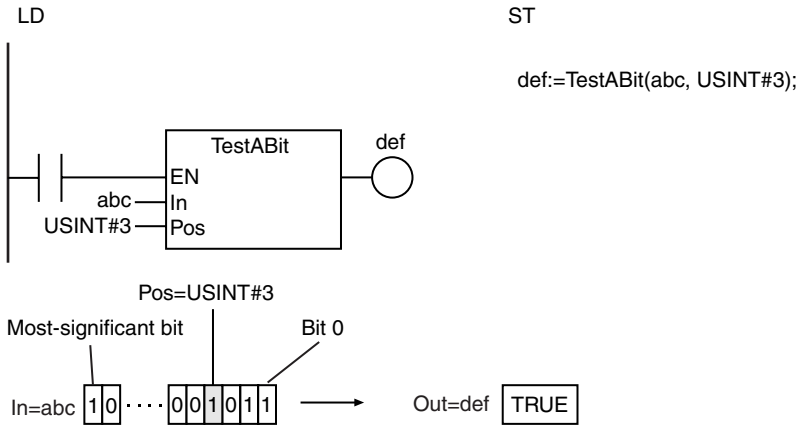
The TestABit instruction assigns the value of the bit at bit position *Pos* in the bit string *In* to the bit value *Out* when *EN* is TRUE.
When *EN* is FALSE, the value of *Out* is FALSE.

● TestABitN

The TestABitN instruction assigns the inverse of the value of the bit at bit position *Pos* in the bit string *In* to the bit value *Out* when *EN* is TRUE.

When *EN* is FALSE, the value of *Out* is FALSE.

The following example shows the TestABit instruction when *Pos* is USINT#3.



Precautions for Correct Use

- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following case. *Out* will be FALSE.
  - The value of *Pos* is greater than No. of bits in *In* − 1.

Sequence Output Instructions

Instruction	Name	Page
RS	Reset-Priority Keep	2-46
SR	Set-Priority Keep	2-48
Set and Reset	Set/Reset	2-50
SetBits and ResetBits	Set Bits/Reset Bits	2-53
SetABit and ResetABit	Set A Bit/Reset A Bit	2-55
OutABit	Output A Bit	2-57

RS

The RS instruction retains the value of a BOOL variable. It gives priority to the *Reset* input if both the *Set* input and *Reset* input are TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RS	Reset-Priority Keep	FB	RS_instance RS Set Q1 Reset1	RS_instance(Set, Reset1, Q1);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Set	Set	Input	Set input	Depends on data type.	---	0
Reset1	Reset		Reset input			
Q1	Keep	Output	Keep output	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Set	OK																			
Reset1	OK																			
Q1	OK																			

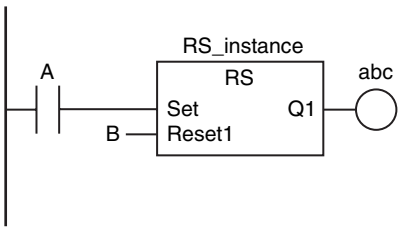
Function

The RS instruction forms a self-holding output that gives priority to resetting. The following table shows the relationship between the inputs are outputs.

Value of *Set*	Value of *Reset1*	Value of *Q1*
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	Not changed.

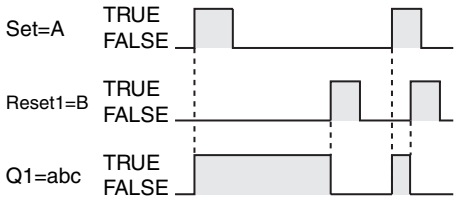
The following figure shows a programming example and timing chart.

LD



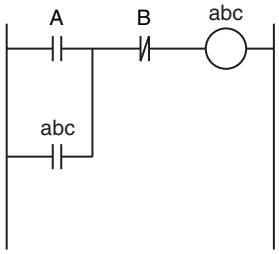
ST

```
RS_instance(A, B, abc);
```



Additional Information

- The RS instruction behaves like the following self-holding rung.



- However, if the RS instruction is in a master control region and the master control region is reset, the behavior will not be the same as the above self-holding rung.

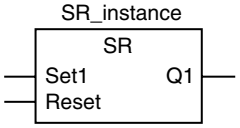
Instruction/rung	Value of *B*	Value of *abc*
RS instruction	TRUE	Not changed.
	FALSE	FALSE
Self-holding rung	TRUE	FALSE
	FALSE	

Precautions for Correct Use

- Never use an NC bit directly from an external device for the *Reset1* input. The internal power supply in the Controller will not turn OFF immediately when the AC power is interrupted (even for momentary interruptions), and the input from the Input Unit may change to ON first. This could cause the *Reset1* input to change to TRUE.
- If this instruction is used in a ladder diagram, the value of *Q1* is retained if an error occurs in the previous instruction on the rung.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), *Q1* retains the value from the last execution.
- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - If the value of *Reset1* is TRUE, the value of *Q1* is retained. If the value of *Reset1* is FALSE, the value of *Q1* changes to FALSE.
  - FALSE is input to the instruction that is connected to *Q1* even if the value of *Q1* is TRUE.

SR

The SR instruction retains the value of a BOOL variable. It gives priority to the *Set* input if both the *Set* input and *Reset* input are TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SR	Set-Priority Keep	FB		SR_instance(Set1, Reset, Q1);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Set1	Set	Input	Set input	Depends on data type.	---	0
Reset	Reset		Reset input			
Q1	Keep	Output	Keep output	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Set1	OK																			
Reset	OK																			
Q1	OK																			

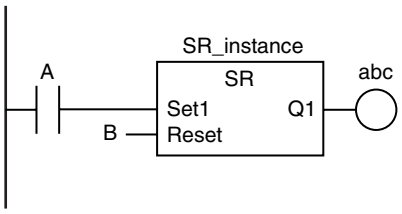
Function

The SR instruction forms a self-holding output that gives priority to setting. The following table shows the relationship between the inputs are outputs.

Value of *Set1*	Value of *Reset*	Value of *Q1*
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	Not changed.

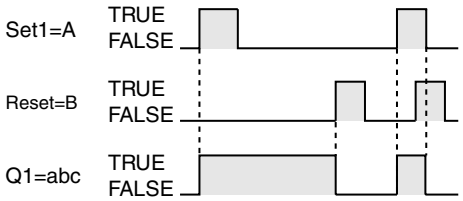
The following figure shows a programming example and timing chart.

LD



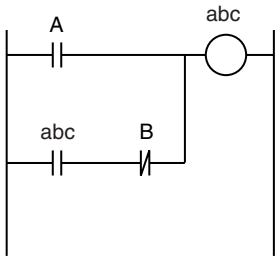
ST

```
SR_instance(A, B, abc);
```



Additional Information

- The SR instruction behaves like the following self-holding rung.



- However, if the SR instruction is in a master control region and the master control region is reset, the behavior will not be the same as the above self-holding rung.

Instruction/rung	Value of *B*	Value of *abc*
SR instruction	TRUE	Not changed.
	FALSE	FALSE
Self-holding rung	TRUE	FALSE
	FALSE	

Precautions for Correct Use

- Never use an NC bit directly from an external device for the *Reset* input. The internal power supply in the Controller will not turn OFF immediately when the AC power is interrupted (even for momentary interruptions), and the input from the Input Unit may change to ON first. This could cause the *Reset* input to change to TRUE.
- If this instruction is used in a ladder diagram, the value of *Q1* is retained if an error occurs in the previous instruction on the rung.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), *Q1* retains the value from the last execution.
- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - If the value of *Reset* is TRUE, the value of *Q1* is retained. If the value of *Reset* is FALSE, the value of *Q1* changes to FALSE.
  - FALSE is input to the instruction that is connected to *Q1* even if the value of *Q1* is TRUE.

Set and Reset

Set: Changes a BOOL variable to TRUE.

Reset: Changes a BOOL variable to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Set	Set	---		None
Reset	Reset	---		None

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Output	Output	Output	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			

Function

● Set

The Set instruction changes *Out* to TRUE if the input is TRUE. If *Out* is TRUE, the Set instruction will not change it to FALSE even if the input changes to FALSE. Use the Reset instruction to change *Out* to FALSE.

● Reset

The Reset instruction changes *Out* to FALSE if the input is TRUE. If *Out* is FALSE, the Reset instruction will not change it to TRUE even if the input changes to FALSE. Use the Set instruction to change *Out* to TRUE.

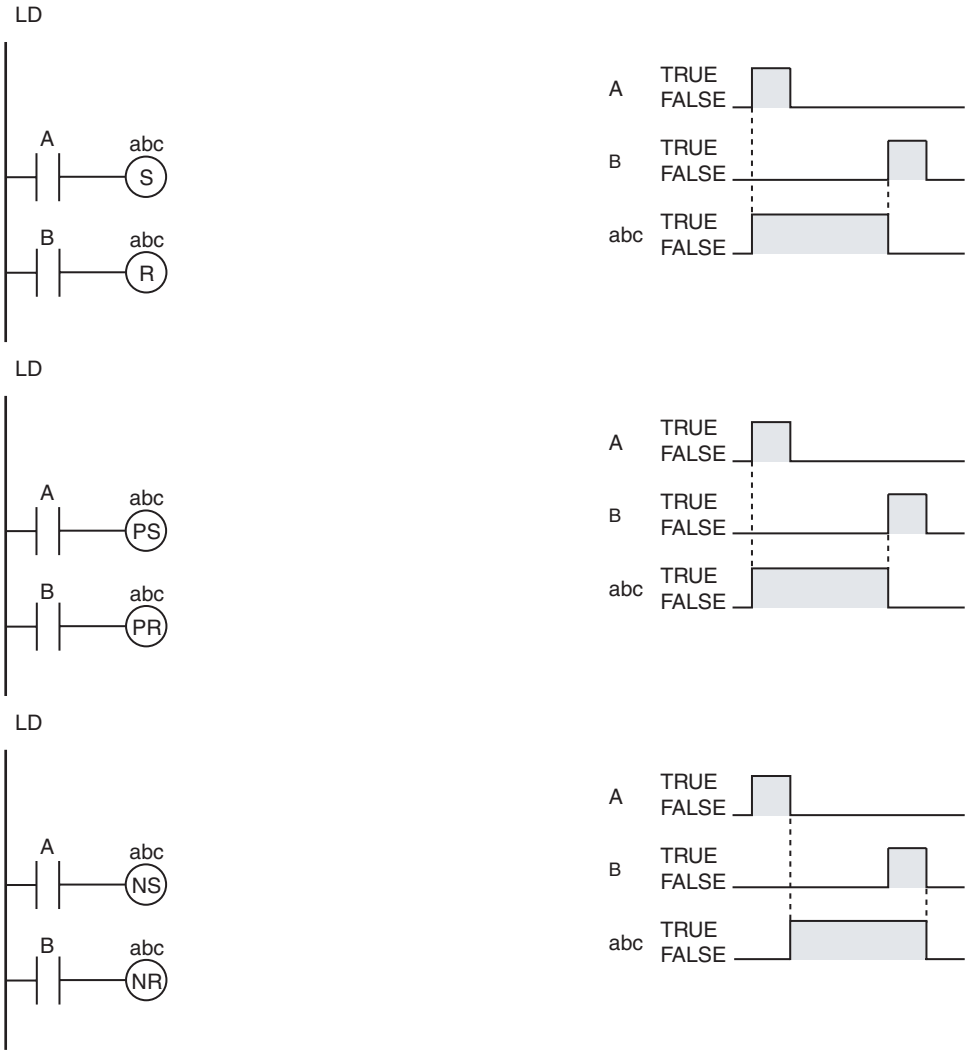
The operation is as shown below if you do not specify upward or downward differentiation.

Instruction	Input	Output value
Set	TRUE	TRUE
	FALSE	Not changed.
Reset	TRUE	FALSE
	FALSE	Not changed.

If you specify upward or downward differentiation, the operation depends on the following: the value of the input for the last execution and the current value of the input. This is shown below.

Instruction	Differentiation specification	Value of input at last execution and current value	Output value
Set	Upward differentiation	FALSE at the last execution → Currently TRUE	TRUE
		Other than the above.	Not changed.
	Downward differentiation	TRUE at the last execution → Currently FALSE	TRUE
		Other than the above.	Not changed.
Reset	Upward differentiation	FALSE at the last execution → Currently TRUE	FALSE
		Other than the above.	Not changed.
	Downward differentiation	TRUE at the last execution → Currently FALSE	FALSE
		Other than the above.	Not changed.

The following figure shows a programming example and timing chart.



Additional Information

Differences between the Set and Reset Instructions and the Out Instruction

- The Set and Reset instructions operate only when the input value changes to TRUE. They do not operate when the input value is FALSE. When the input value is FALSE, the output does not change.
- The Out instruction changes the specified variable to TRUE when the result from the previous instruction is TRUE and to FALSE when the result from the previous instruction is FALSE. It operates both when the input is TRUE and when it is FALSE.

Differences between the Set and Reset Instructions and the SR and RS Instructions

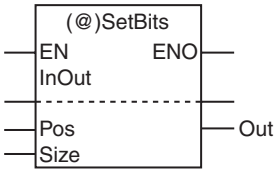
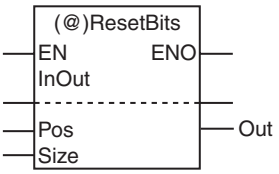
- The SR and RS instructions require that the *Set* input and *Reset* input are in the same place in the program. You can place the Set and Reset instructions in different places.

Precautions for Correct Use

- If this instruction is in a master control region and the master control region is reset, the value of *Out* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *Out* is retained.
- These instructions will not change the value of *Out* if you specify upward differentiation and the input is TRUE immediately after the power turns ON. The input must first change to FALSE and then to TRUE before the value of *Out* changes.
- These instructions will change the value of *Out* if you do not specify upward differentiation and the input is TRUE immediately after the power turns ON. In this case it is not necessary for the input to change to FALSE first.

SetBits and ResetBits

SetBits: Changes consecutive bits in bit string data to TRUE.
ResetBits: Changes consecutive bits in bit string data to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetBits	Set Bits	FUN		SetBits(InOut, Pos, Size);
ResetBits	Reset Bits	FUN		ResetBits(InOut, Pos, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Bit string	In-out	Bit string	Depends on data type.	---	---
Pos	Bit position	Input	Specified bit position	0 to No. of bits in *InOut* – 1	---	0
Size	Number of bits		Number of bits	0 to No. of bits in *InOut*		1
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		OK	OK	OK	OK															
Pos						OK														
Size						OK														
Out	OK																			

Function

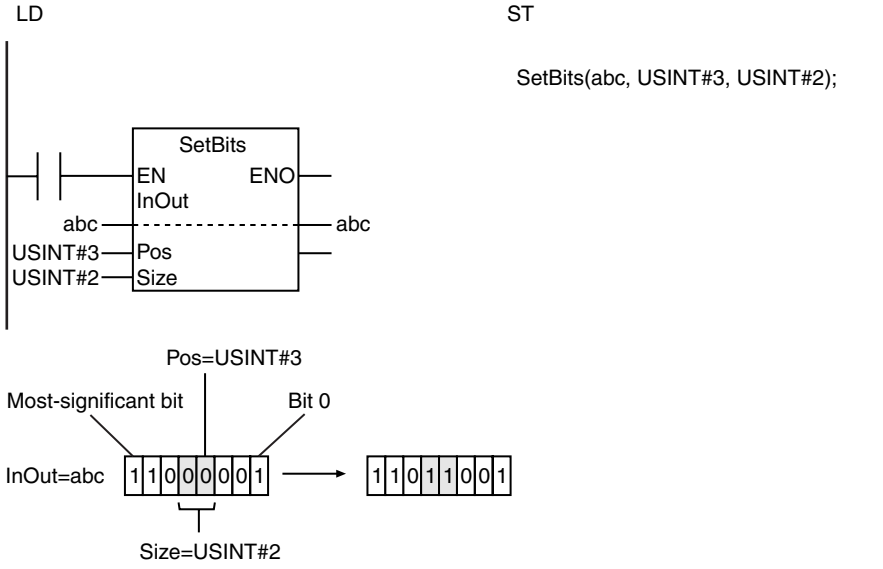
● SetBits

The SetBits instruction changes the value of *Size* bits from the bit position *Pos* in the bit string *InOut* to TRUE. The status of the other bits will not change.

● ResetBits

The ResetBits instruction changes the value of *Size* bits from the bit position *Pos* in the bit string *InOut* to FALSE. The status of the other bits will not change.

The following example shows the SetBits instruction when *Pos* is USINT#3 and *Size* is USINT#2.



Additional Information

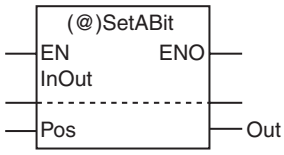
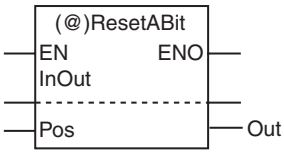
Use these instructions to globally set variables with AT specification in memory areas that handle data by word (e.g., the DM Area) to TRUE or FALSE.

Precautions for Correct Use

- If this instruction is in a master control region and the master control region is reset, the value of *InOut* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- The value of *InOut* does not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - The value of *Pos* is greater than No. of bits in *InOut* – 1.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* or *Size* exceeds the number of bits in *InOut*.

SetABit and ResetABit

SetABit: Changes the specified bit in bit string data to TRUE.
ResetABit: Changes the specified bit in bit string data to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetABit	Set A Bit	FUN		SetABit (InOut, Pos);
ResetABit	Reset A Bit	FUN		ResetABit (InOut, Pos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Bit string	In-out	Bit string	Depends on data type.	---	---
Pos	Bit position	Input	Specified bit position	0 to No. of bits in *InOut* – 1	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		OK	OK	OK	OK															
Pos						OK														
Out	OK																			

Function

● SetABit

The SetABit instruction changes the value of the bit at bit position *Pos* in the bit string *InOut* to TRUE.

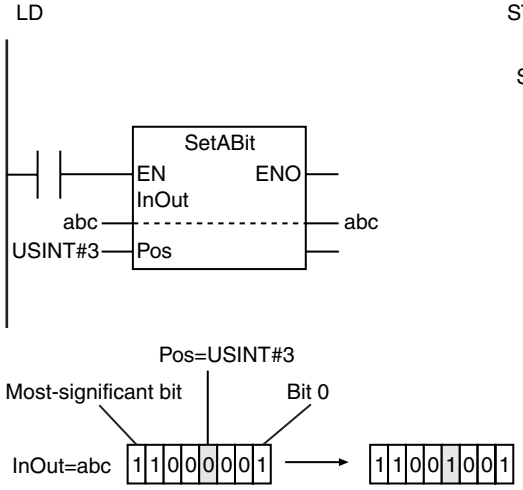
The bits that are not specified do not change.
Even if EN changes to FALSE after execution, the *Pos* bit in *InOut* will not change.

● ResetABit

The ResetABit instruction changes the value of the bit at bit position *Pos* in the bit string *InOut* to FALSE.

The bits that are not specified do not change.
Even if EN changes to FALSE after execution, the *Pos* bit in *InOut* will not change.

The following example shows the SetABit instruction when *Pos* is USINT#3.



Additional Information

Differences between the SetABit and ResetABit Instructions and the OutABit Instruction

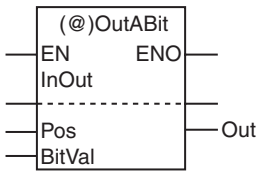
- The SetABit and ResetABit instructions change the value of the specified bit to either TRUE or FALSE.
- With the OutABit instruction, however, you can dynamically change the value to which the specified bit is set.

Precautions for Correct Use

- If this instruction is in a master control region and the master control region is reset, the value of *InOut* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - The value of *Pos* is greater than No. of bits in *In* – 1.

OutABit

The OutABit instruction changes the specified bit in bit string data to TRUE or FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
OutABit	Output A Bit	FUN		OutABit (InOut, Pos, BitVal);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Bit string	In-out	Bit string	Depends on data type.	---	---
Pos	Bit position	Input	Specified bit position	0 to No. of bits in *InOut* – 1	---	0
BitVal	Set value		Value to set	Depends on data type.		TRUE
Out	Return value	Output	Always TRUE	TRUE only	---	---

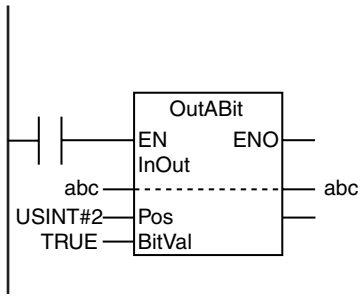
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		OK	OK	OK	OK															
Pos						OK														
BitVal	OK																			
Out	OK																			

Function

The OutABit instruction stores the value of set value *BitVal* at bit position *Pos* in the bit string *InOut*. Only the bit at *Pos* changes.

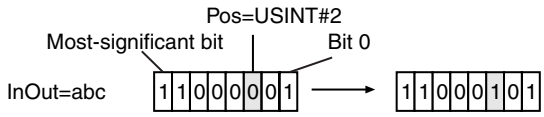
The following example is for when *Pos* is USINT#2 and *BitVal* is TRUE.

LD



ST

```
OutABit(abc, USINT#2, TRUE);
```



Additional Information

Differences between the SetABit and ResetABit Instructions and the OutABit Instruction

- The SetABit and ResetABit instructions change the value of the specified bit to either TRUE or FALSE.
- With the OutABit instruction, however, you can dynamically change the value to which the specified bit is set if you change the value of *BitVal*.

Precautions for Correct Use

- If this instruction is in a master control region and the master control region is reset, the value of *InOut* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- Return value *Out* is not used when the instruction is used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - The value of *Pos* is greater than No. of bits in *InOut* − 1.

Sequence Control Instructions

Instruction	Name	Page
End	End	2-60
RETURN	Return	2-61
MC and MCR	Master Control Start/ Master Control End	2-62
JMP	Jump	2-74
FOR and NEXT	Repeat Start/ Repeat End	2-76
BREAK	Break Loop	2-81

End

The End instruction ends execution of a program in the current task period.

Instruction	Name	FB/FUN	Graphic expression	ST expression
End	End	FUN	End EN ENO	None

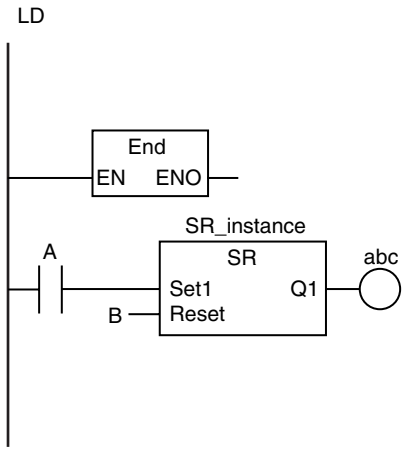
Variables

None

Function

The End instruction ends execution of a program in the current task period.

The following figure shows a programming example. When the End instruction is executed in the example, the SR instruction that follows it is not executed.



Precautions for Correct Use

- This instruction must be used only in a program.
- If this instruction is used in a function, function block, or inline ST, a building error will occur.
- You must connect this instruction to the left bus bar.

RETURN

The RETURN instruction ends a function or function block and returns processing to the calling instruction.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RETURN	Return	FUN	RETURN EN ENO	RETURN;

Variables

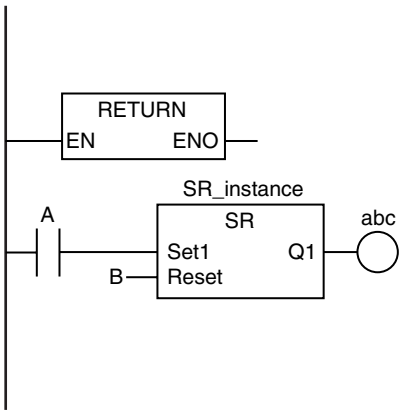
None

Function

The RETURN instruction ends a function or function block and returns processing to the calling instruction.

The following figure shows a programming example. When the RETURN instruction is executed in the example, the SR instruction that follows it is not executed.

LD



ST

```
RETURN;
SR_instance(A, B, abc);
```

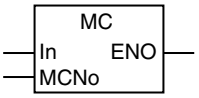
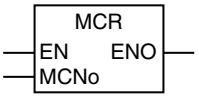
Precautions for Correct Use

- Observe the following precautions if you use this instruction in a ladder diagram.
  - Use this instruction only in functions and function blocks. If you use it in a program, a building error will occur.
  - Always connect this instruction directly to the left bus bar.
- Before you execute this instruction set the return values, output variables, and ENO value of the POU.
- If you use this instruction too often, the flow of processing will be difficult to understand. Use it with caution.

MC and MCR

MC: Marks the starting point of a master control region and resets the master control region.

MCR: Marks the end point of a master control region.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MC	Master Control Start	---		None
MCR	Master Control End	---		None

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In (MC instruction only)	Master control input	Input	FALSE: Resets the master control region.	Depends on data type.	---	TRUE
MCNo	Master control number		Master control number	0 to 14*		1

* The number is automatically registered by the Sysmac Studio. You do not need to set it.

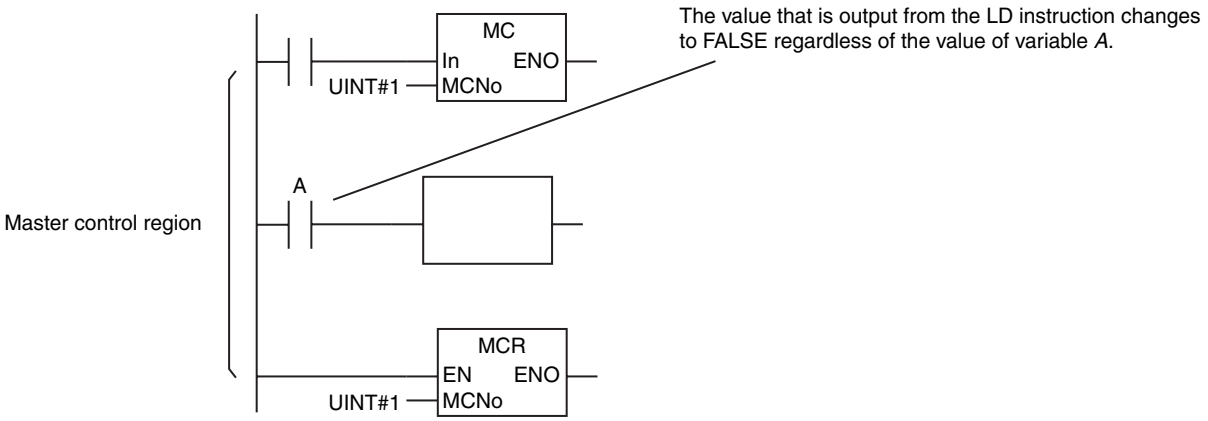
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In (MC instruction only)	OK																			
MCNo							OK													

Function

Master control is used to stop processing or place in an equivalent status all POUs in a specified region of a program. You can use master control to easily control the execution conditions for a relatively long segment of processing.

The region in the program for which master control is applied is called the master control region. You place the MC instruction at the start of the master control region and the MCR instruction at the end. When the value of the master control input *In* changes to FALSE, the outputs for all LD instructions that are connected to the left bus bar in the master control region are forced to change to FALSE. This is called a master control reset.

When master control is reset, the POUs that follow the LD instructions, as a rule, operate as if the execution condition is FALSE. There are, however, some POUs that operate differently. This is explained later.



If the value of *In* is TRUE, then master control is not reset. The POUs in the master control region operate normally.

POU Operation during a Master Control Reset

The operation of the POUs when master control is reset depends on the POU as described in the following table.

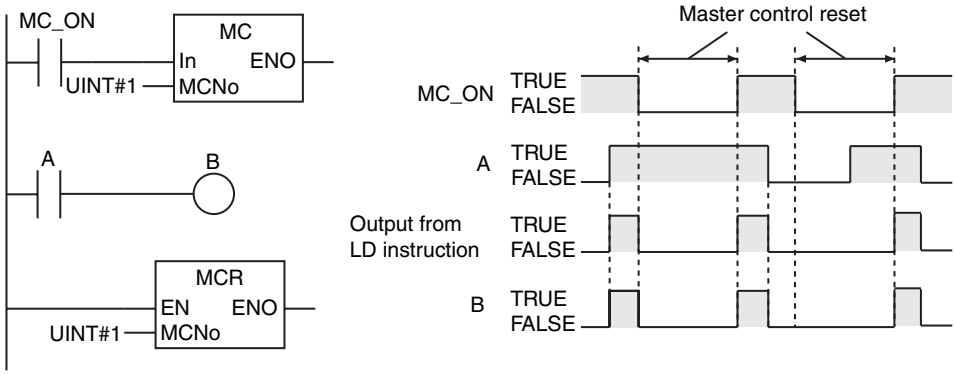
POU	Operation
Out and OutABit instructions	FALSE is output to the specified variable.
OutNot instruction	FALSE is output to the specified variable.
Set and Reset instructions	The output from before the master control reset is retained.
TON instruction	The instruction operates with a FALSE value for timer input *In*. That means that the timer is reset. The value of elapsed time *ET* changes to 0 and the value of timer output *Q* changes to FALSE.
TOF instruction	The instruction operates with a TRUE value for timer input *In*. That means that the timer is reset. The value of elapsed time *ET* changes to 0 and the value of timer output *Q* changes to TRUE. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE.
TP instruction	The instruction operates with a FALSE value for timer input *In*. That means that the timer is reset. Timing active: The value of elapsed time *ET* is incremented to the end and then returns to 0. The value of timer output *Q* is TRUE until the end of timing, and then it changes to FALSE. Timing not active: The value of *ET* changes to 0 and the value of *Q* changes to FALSE. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even while timing is active.
AccumulationTimer instruction	The instruction operates with a FALSE value for timer input *In*. That means that the timer stops. The values of elapsed time *ET* and timer output *Q* are retained. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even if the value of *Q* is TRUE. However, reset *Reset* is enabled.
Timer instructions	The instruction operates with a FALSE value for timer input *In*. That means that the timer is reset. Remaining time *ET* is set to the value of set time *PT*, and the value of timer output *Q* changes to FALSE.
CTU, CTD, and CTUD instructions	These instructions are not executed. If one of these instructions was operating before the master control reset, the count value from before the reset will be held and the Counter Completion Flag *Q* will be FALSE.
JMP instruction	This instruction is not executed.
FOR and NEXT instructions	These instructions are not executed.

POU	Operation
BREAK instruction	This instruction is not executed.
Function blocks that are executed over more than one task period (i.e., instructions with *Done*, *Busy*, and *Error* output variables)	The power flow from the left bus bar changes to FALSE. If the instruction was operating before the master control reset, execution of the instruction is continued until processing is completed. *Busy, Done,* and *Error* outputs will be made, but FALSE will always be output if the next instruction is an output instruction. If a variable is directly connected to *Busy, Done*, or *Error*, the proper value for the instruction specifications will be assigned to that variable. You can also get the value of *Busy, Done*, or *Error* in the form *instance_name.output_variable.*
Other functions	These are not executed.
Other function blocks	The power flow from the left bus bar changes to FALSE.

The operation of some typical instructions is described below.

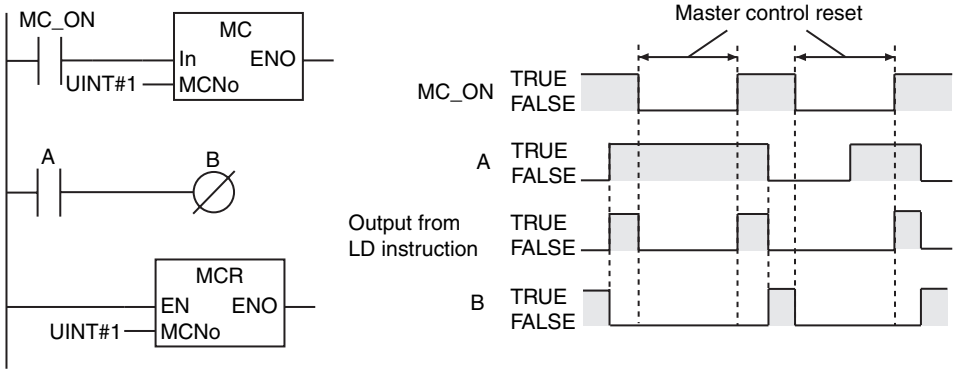
● Out

FALSE is output while the master control is reset.



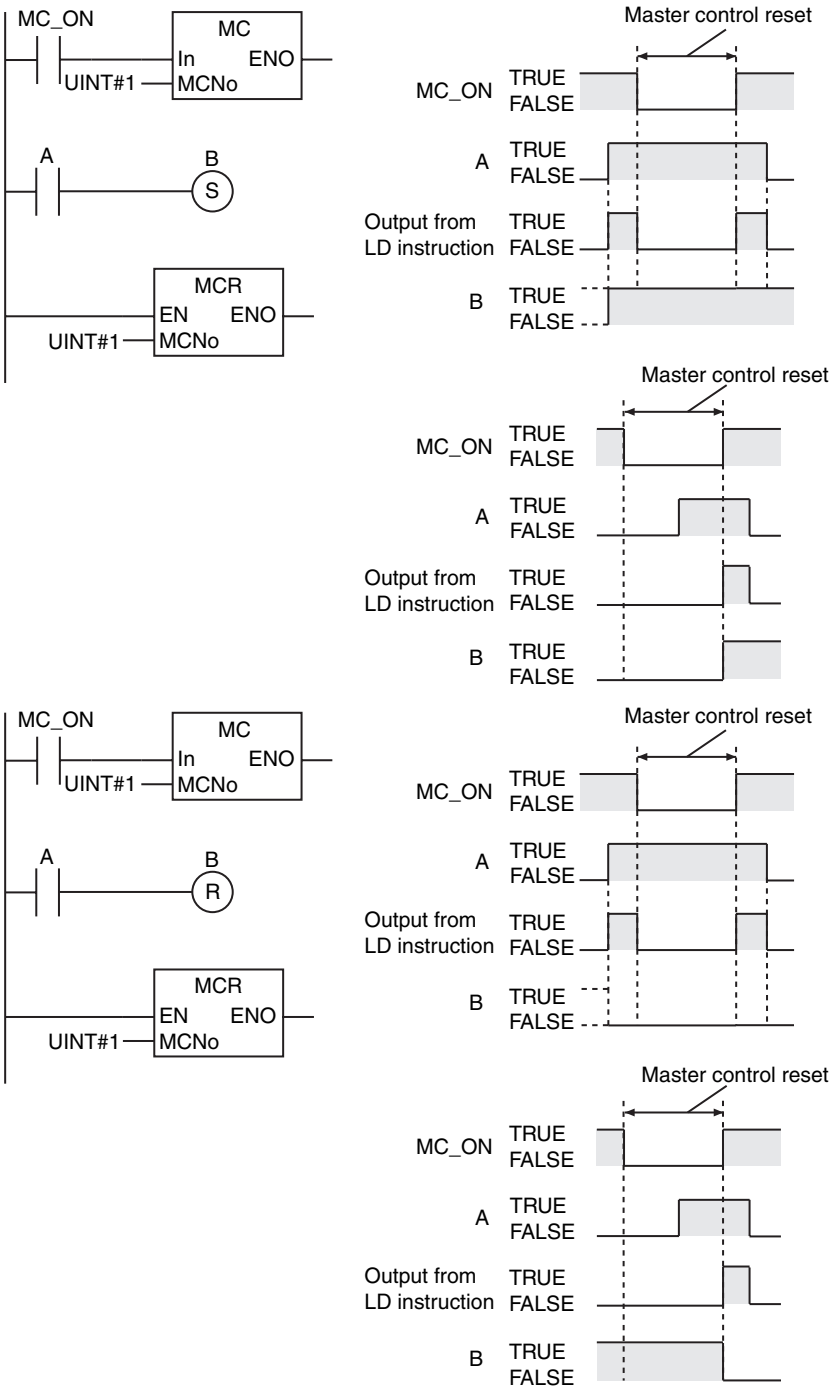
● OutNot

FALSE is output while the master control is reset. Caution is required because this operation of the OutNot instruction is different from when the output of the previous LD instruction is FALSE.



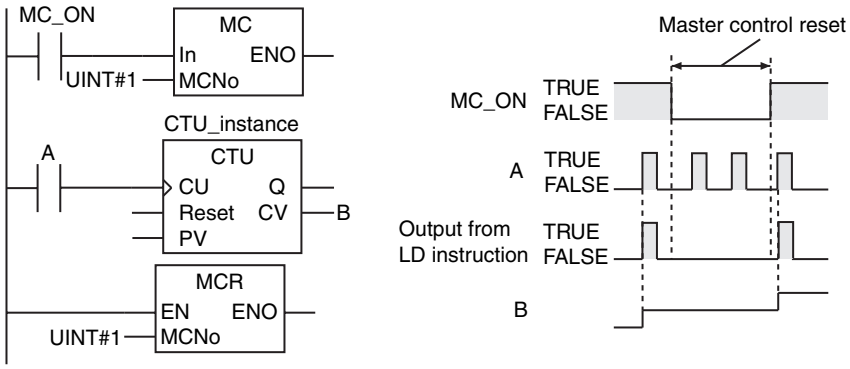
● Set and Reset

The previous value of the output is retained while the master control is reset.



● CTU, CTD, and CTUD

The previous counter value is retained while the master control is reset. When the master control reset is cleared, counting continues from the counter value that was retained.



Operation of POUs with Input Upward Differentiation or Input Downward Differentiation

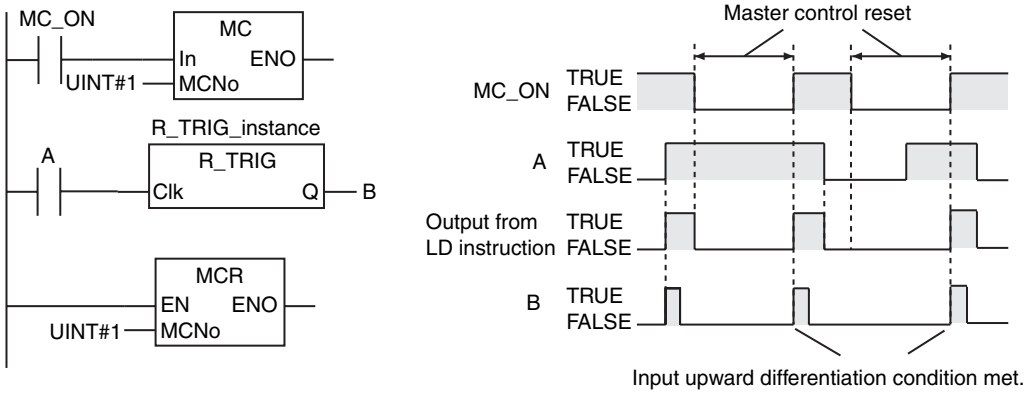
The POUs that are given in the following table have upward or downward differentiation specifications.

Differentiation	Instructions
Input upward differentiation	• LD, LDN, AND, ANDN, OR, ORN, and OUT with upward differentiation specifications • R_TRIG (Up) • Functions with an @ input upward differentiation option • Functions blocks (e.g., counter instructions) with input upward differentiation specifications
Input downward differentiation	• LD, LDN, AND, ANDN, OR, ORN, and OUT with downward differentiation specifications • F_TRIG (Down) • Functions with a % input downward differentiation option

When the master control is reset or the reset is cleared, the execution conditions for these POUs change. That means that the upward or downward differentiation conditions for these POUs may be met. If the upward or downward differentiation conditions are met, then the instructions are executed accordingly. The operation of some typical instructions is described below.

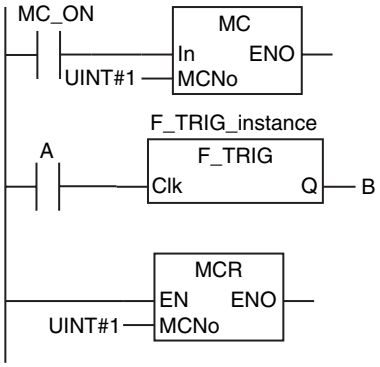
● R_TRIG (Up)

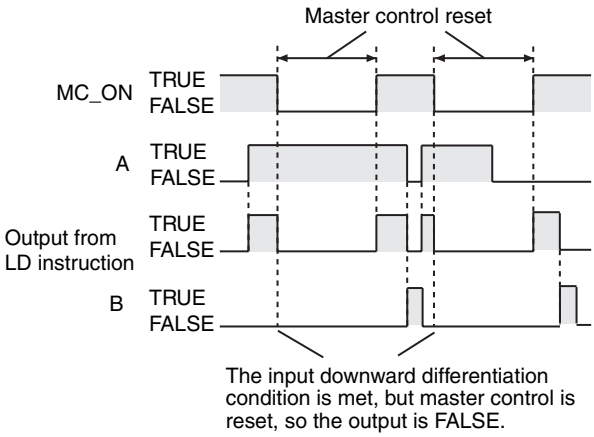
When the master control is reset, the execution condition changes to FALSE. If the execution condition is TRUE when the master control reset is cleared, the input upward differentiation condition is met and the instruction operates accordingly.



● F_TRIG (Down)

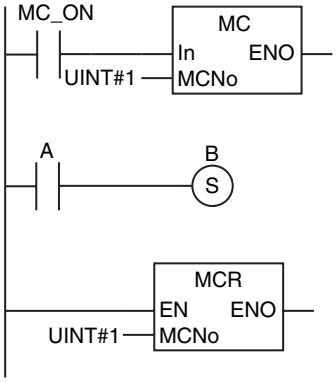
When the master control is reset, the execution condition changes to FALSE. If the previous execution condition was TRUE, then the input downward differentiation condition is met. However, the value of the output from the F_TRIG (Down) instruction during the master control reset is forced to change to FALSE, so the output value changes to FALSE.

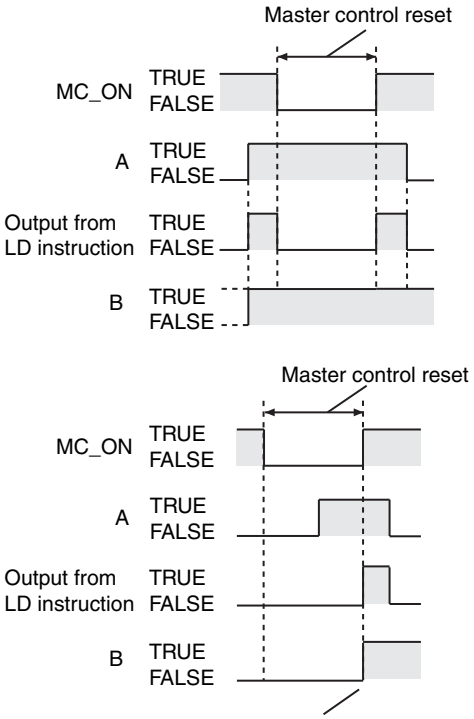


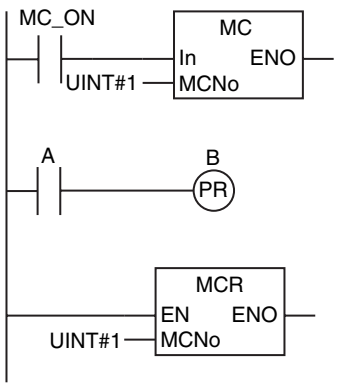


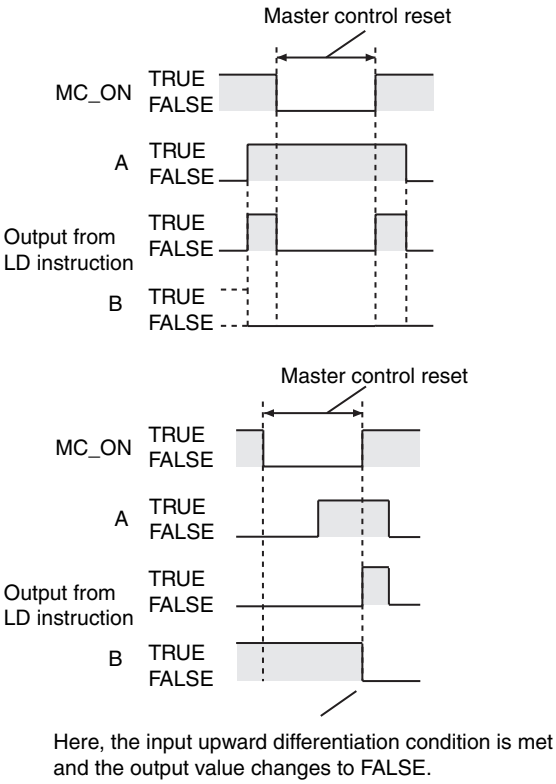
● Set and Reset with Input Upward Differentiation Specification

The previous value of the output is retained while the master control is reset. When the master control reset is cleared, the execution condition changes to TRUE and the instruction operates.



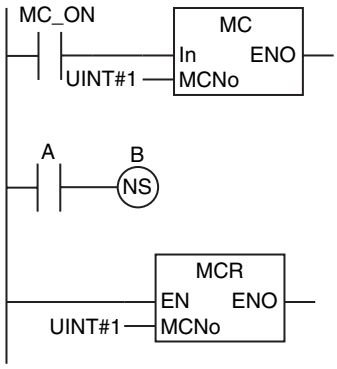


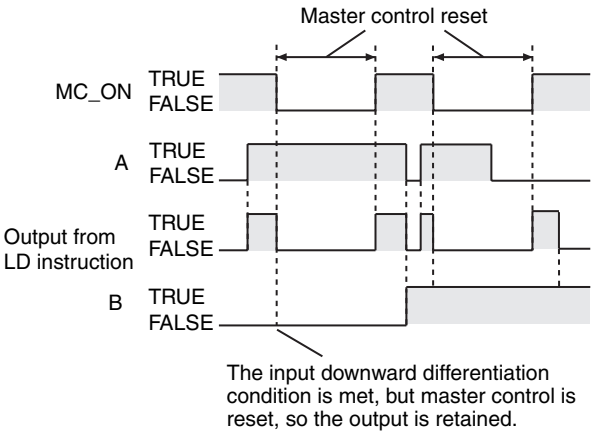


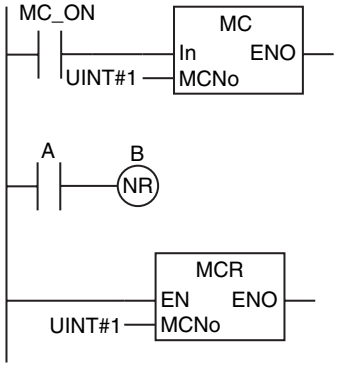


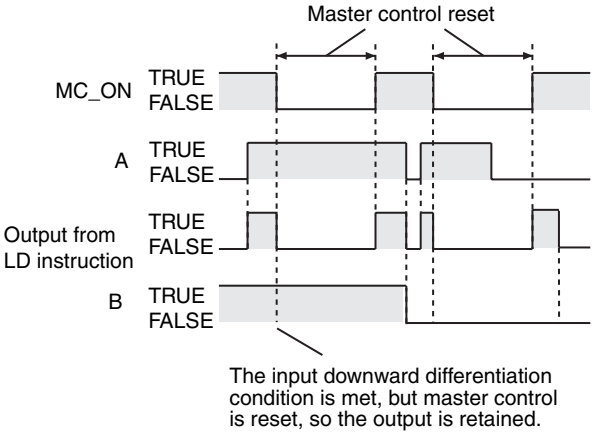
● Set and Reset with Input Downward Differentiation Specification

When the master control is reset, the execution condition changes to FALSE. If the previous execution condition was TRUE, then the input downward differentiation condition is met. However, during the master control reset, the previous output value is retained, so as a result the value of the output is retained.



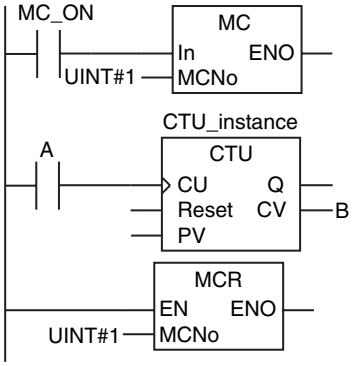


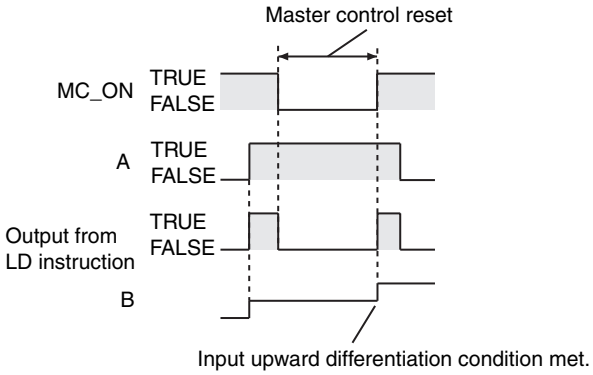




● CTU, CTD, and CTUD

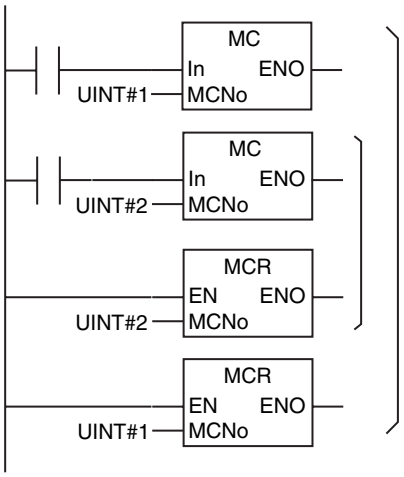
When the master control is reset, the value of the counter input changes to FALSE. If the value of the counter input is TRUE when the master control reset is cleared, the input upward differentiation condition is met and the instruction counts.





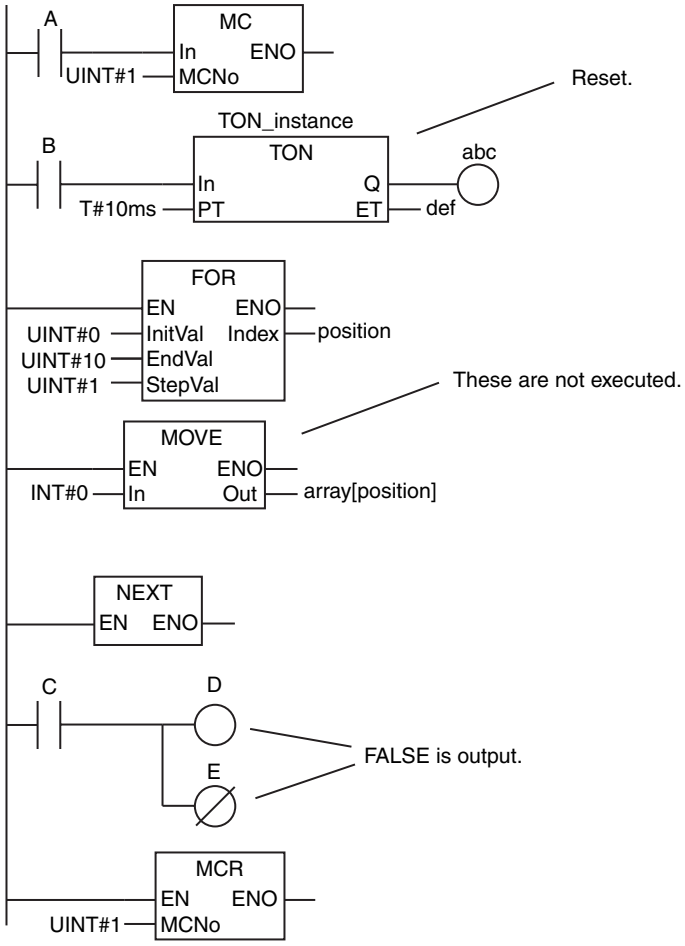
Always use the MC and MCR instructions as a pair in the same POU. The same value is used for master control number *MCNo* for both of the paired MC and MCR instructions. The user does not set the value of *MCNo*. It is automatically registered by the Sysmac Studio.

The MC and MCR instructions can be nested to up to 15 levels.



The following figure shows a programming example.

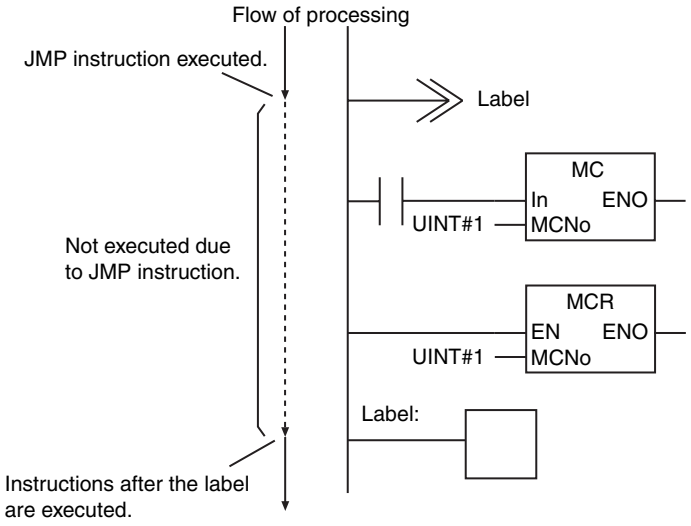
If the value of bit A is FALSE, the master control region is reset. While the master control region is in a reset state, the TON and MOVE instructions are not executed. Also the Out instruction and OutNot instruction will output FALSE to bits D and E.



Precautions for Correct Use

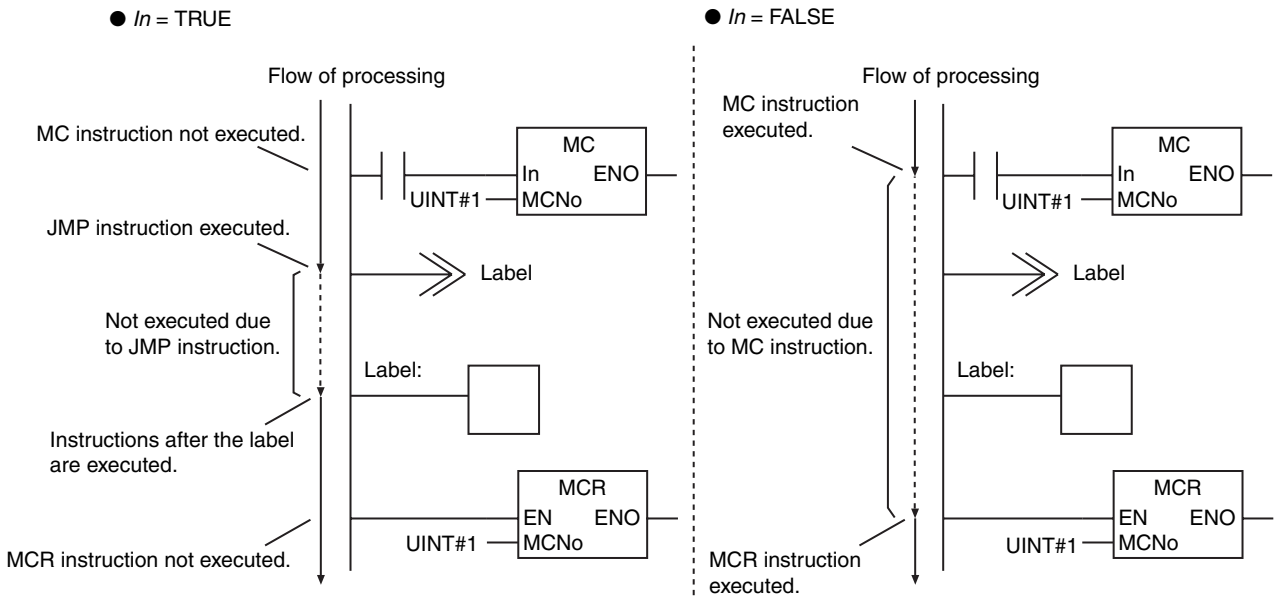
- These instructions must be used in a ladder diagram. They cannot be used in ST. They also cannot be used in inline ST in a ladder diagram.
- Always use the MC and MCR instructions as a pair in the same POU.
- Always place the MCR instruction after the MC instruction.
- Do not nest the MC and MCR instructions to more than 15 levels.
- If there is inline ST in the master control region, the inline ST is not executed when the master control region is reset.

- If you use the MC and MCR instructions and the JMP instruction together, the operation is as follows:
  - The following figure shows an MC-MCR pair inside a JMP-Label pair. Here, the jump is executed regardless of the value of *In*.

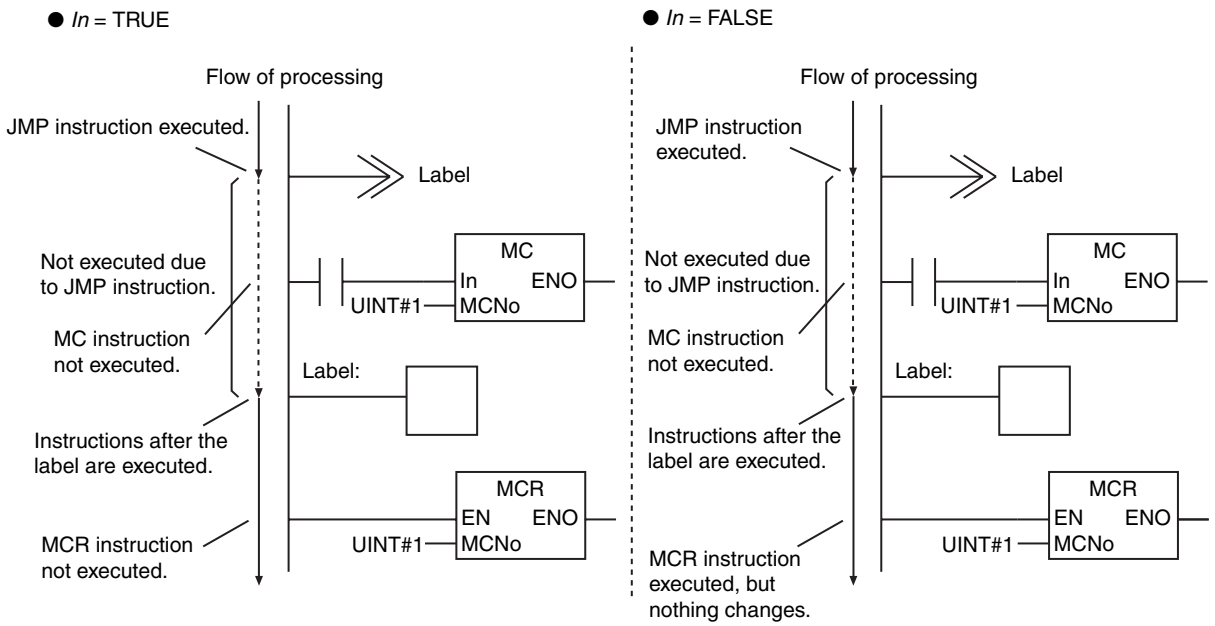


  - The following figure shows a JMP-Label pair inside an MC-MCR pair. Here, operation is as given in the following table.

Value of *In*	Operation
TRUE	Master control region is not reset. The jump is made.
FALSE	Master control region is reset. The jump is not made.

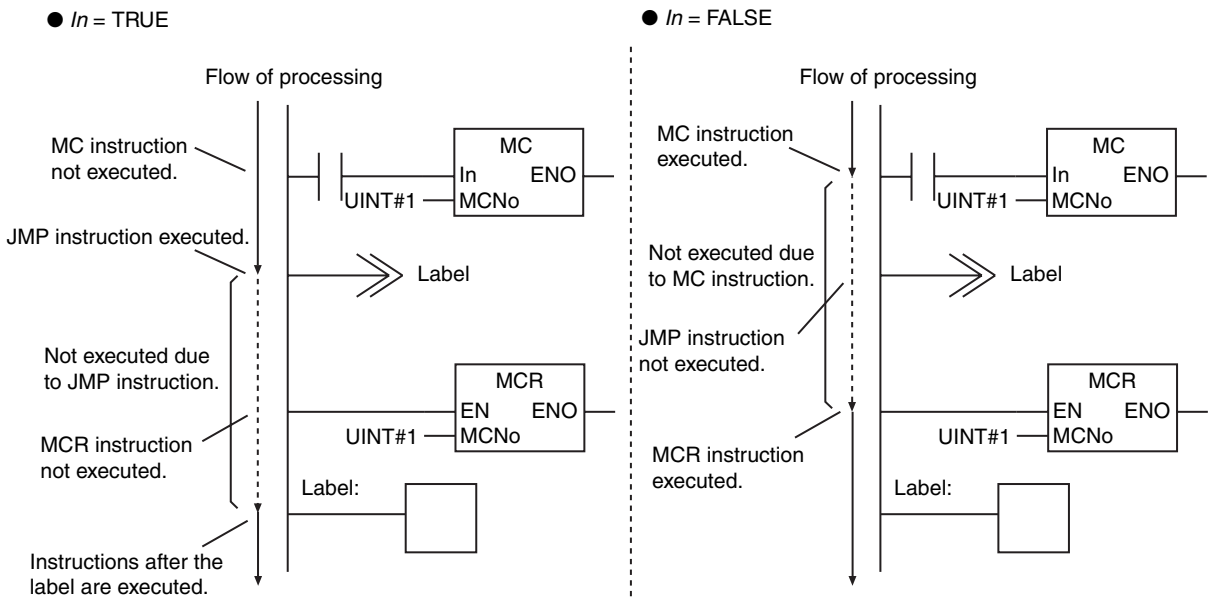


- The instructions are in the following order in the following figure: JMP instruction, MC instruction, Label, and MCR instruction. First, the jump is made. As a result, the MC instruction is not executed. Therefore, the instructions after the Label instruction are executed. If the value of *In* is FALSE, the MCR instruction is executed, but nothing changes.



- The instructions are in the following order in the following figure: MC instruction, JMP instruction, MCR instruction, and Label. Here, operation is as given in the following table.

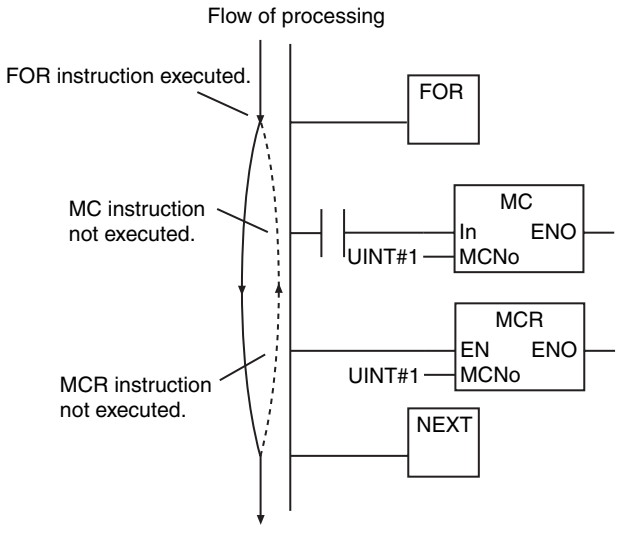
Value of *In*	Operation
TRUE	Master control region is not reset. The jump is made.
FALSE	Master control region is reset. The jump is not made.



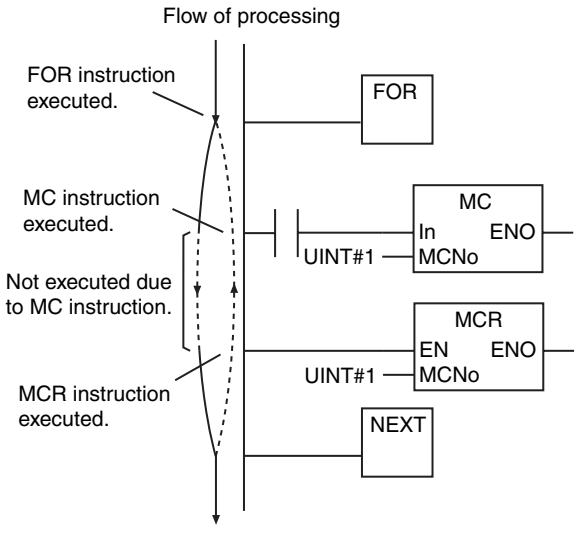
- If you use the MC and MCR instructions and the FOR and NEXT instructions together, the operation is as follows:
  - The following figure shows an MC-MCR pair inside a FOR-NEXT pair. Here, operation is as given in the following table.

Value of *In*	Operation
TRUE	Master control region is not reset. The FOR loop is executed.
FALSE	Master control region is reset. The FOR loop is executed, but the instructions between the MC and MCR instructions are not executed.

● *In* = TRUE



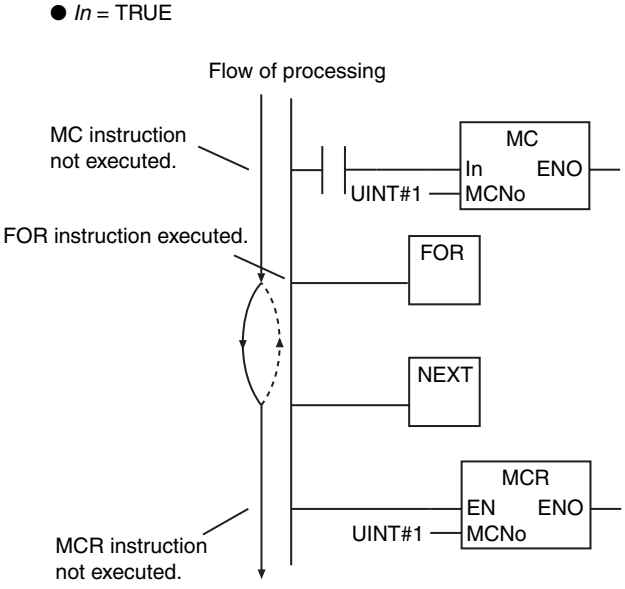
● *In* = FALSE



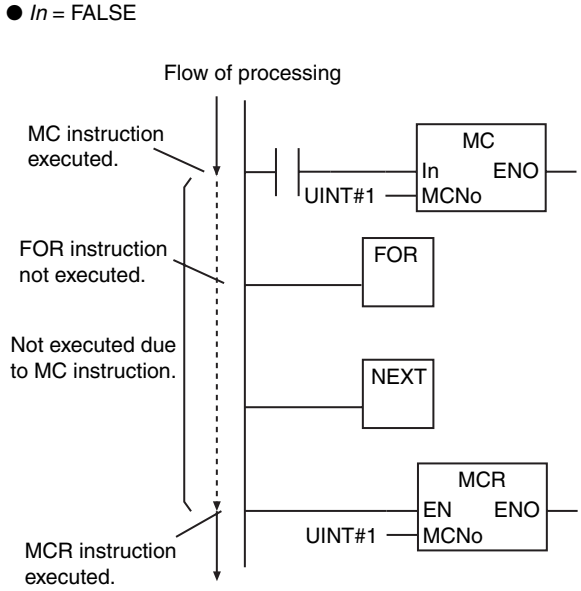
  - The following figure shows a FOR-NEXT pair inside an MC-MCR pair. Here, operation is as given in the following table.

Value of *In*	Operation
TRUE	Master control region is not reset. The FOR loop is executed.
FALSE	Master control region is reset. The FOR loop is not executed.

● *In* = TRUE



● *In* = FALSE



- A building error occurs if the FOR, NEXT, MC, and MCR instructions are used in either of the following orders.

  FOR, MC, NEXT, MCR, or MC, FOR, MCR, NEXT

JMP

The JMP instruction moves processing to the specified jump destination.

Instruction	Name	FB/FUN	Graphic expression	ST expression
JMP	Jump	FUN	Label	None

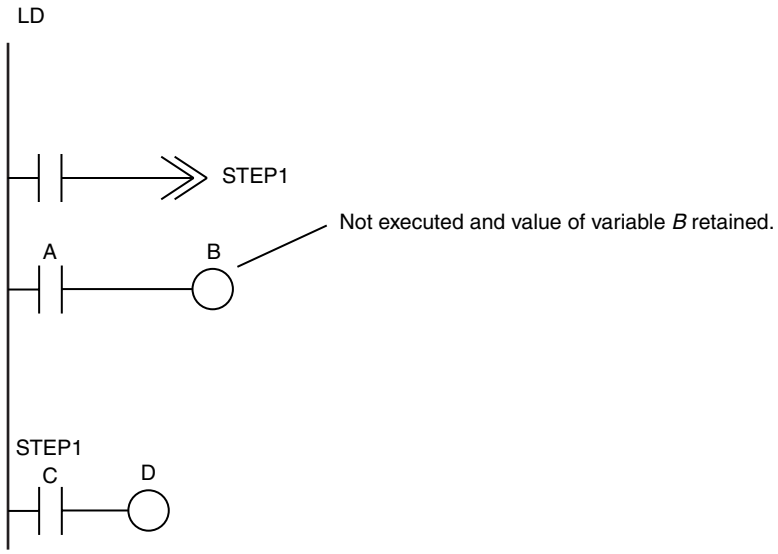
Variables

None

Function

When the execution condition is TRUE, the JMP instruction moves processing to the jump destination specified by a Label in a ladder diagram. The label can be any text string.

The following figure shows a programming example. This example uses the text string *STEP1* as the label. When the JMP instruction is executed, processing moves to the location marked *STEP1*. In this example, the Out instruction between the JMP instruction and the Label is not executed, and the value of variable *B* is retained.



Additional Information

- You can also jump to a Label instruction above the JMP instruction in the section.
- You can use the same Label instruction as the jump destination for more than one JMP instruction.

Precautions for Correct Use

- You cannot omit labels. If you omit a label, a building error will occur.
- Place the JMP and Label instructions in the same POU and in the same section.
- Do not set the same Label instruction more than once in the same section.
- You cannot jump into a FOR-NEXT loop from outside the loop.

- The following restrictions apply to the characters that can be used as labels.

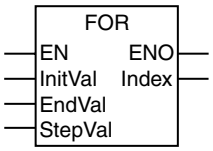
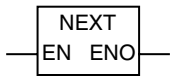
Item	Specification
Maximum number of bytes	127 bytes 127 characters when converted to ACSII 31 characters when converted to Japanese characters (including single-byte kana)
Character code	UTF-8
Applicable characters	Not case sensitive. English alphanumeric characters and other language characters. Symbols: _ (underbar) and ~ (tilda)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with ‘P_’
Prohibited characters	Blank space ! " # $ & ' ( ) * + , - . / : ; < = > ? @ [ ] ^ ` %

- Variable names cannot be used as labels.

FOR and NEXT

FOR: Marks the starting position for repeat processing and specifies the repeat condition.

NEXT: Marks the ending position for repeat processing.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FOR	Repeat Start	FUN		FOR Index:=InitVal TO EndVal BY StepVal DO expression END_FOR*; * In ST, do not use NEXT to mark the ending position of repeat processing. Use END_FOR instead.
NEXT	Repeat End	FUN		

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InitVal	Initial value	Input	Value to set the *Index* to when repetition is started.	Depends on data type.	---	*1
EndVal	End value		Value of Index where repetition is stopped			
StepVal	Increment		Value to add to *Index* each time processing is repeated	Depends on the data type (but 0 is not allowed)		*2
Index	Control variable	Output	Loop index	Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 If you omit the input parameter in a ladder diagram, the default value is not applied. A building error will occur. If you omit the input parameter in ST, a default value of 1 is applied.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InitVal						OK	OK	OK	OK	OK	OK	OK	OK							
EndVal	Must be the same data type as *InitVal*.																			
StepVal	Must be the same data type as *InitVal*.																			
Index	Must be the same data type as *InitVal*.																			

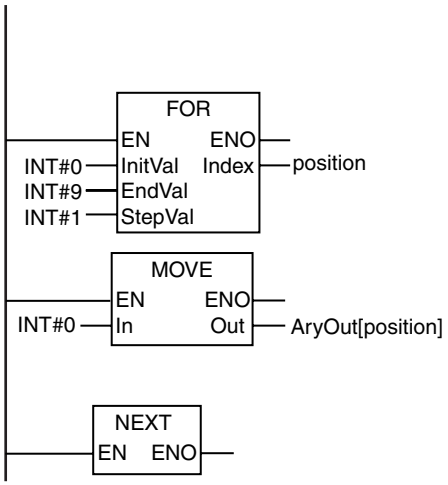
Function

The FOR and NEXT instructions repeat the processing that you place between them. (FOR and END_FOR are used in ST.) The processing procedure for a FOR-NEXT loop is as follows:

1. The value of *InitVal* is set in control variable *Index*.
2. The value of *Index* is checked to see if it is equal to or greater than *InitVal* and equal to or less than *EndVal* (or equal to or greater than *EndVal* and equal to or less than *InitVal).* If it is, the process moves to step 3. If it is not, repeat processing is ended and the next instruction after the NEXT instruction (or the END_FOR instruction in ST) is moved to.
3. The processing between the FOR instruction and the NEXT instruction (or the END_FOR instruction in ST) is executed once.
4. The value of *StepVal* is added to *Index*.
5. The process returns to step 2.

The following example is for when *InitVal* is INT#0, *EndVal* is INT#9, and *StepVal* is INT#1. The MOVE instruction is executed 10 times and INT#0 is assigned to array variables *AryOut[0] to AryOut[9]*.

LD



ST

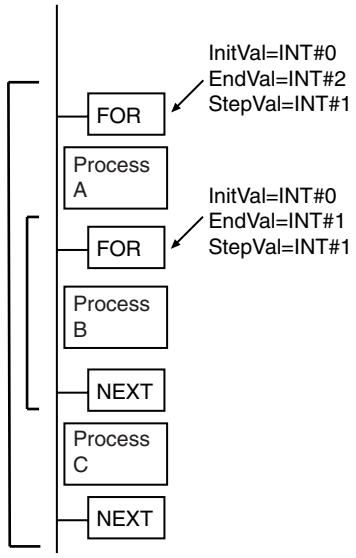
```
FOR position:=INT#0 TO INT#9 BY INT#1 DO
        AryOut[position]:=INT#0;
END_FOR;
```

Additional Information

- Execute a BREAK instruction (or an EXIT instruction in ST) to cancel repeat processing. The processing between the BREAK instruction and the NEXT instruction will not be executed.
- The value of *StepVal* can be negative. The value of *InitVal* can be larger than the value of *EndVal*.

- FOR-NEXT loops (or FOR-END_FOR loops in ST) can be nested. In the following figure, the processes are performed in the following order.

Process A → Process B → Process B → Process C → Process A → Process B → Process B → Process C → Process A → Process B → Process B → Process C



Precautions for Correct Use

- In a ladder diagram, connect the FOR and NEXT instructions directly to the left bus bar.
- If you use this instruction in ST, you can use a function or expression that returns an integer for *InitVal*. You cannot use a function or expression for *EndVal* or *StepVal*.
- Always use the FOR and NEXT instructions (FOR and END_FOR statements in ST) as a pair. A programming error will occur if there is not the same number of both instructions.
- Always use the FOR-NEXT pair (the FOR-END_FOR pair in ST) in the same program section.
- If the value of *InitVal* is less than the value of *EndVal*, use a positive number for the value of *StepVal*. If the value of *InitVal* is greater than the value of *EndVal*, use a negative number for the value of *StepVal*.
- Set the condition to end repetition carefully so that you do not create an infinite loop.

  Example: If the values that are given in the following table are used for the input parameters to the variables, the value of *Index* will never be greater than the value of *EndVal* because the maximum value of SINT data is 255. Therefore, an infinite loop is created.

Variable	Value of input parameter
InitVal	SINT#0
EndVal	SINT#255
StepVal	SINT#1
Index	---

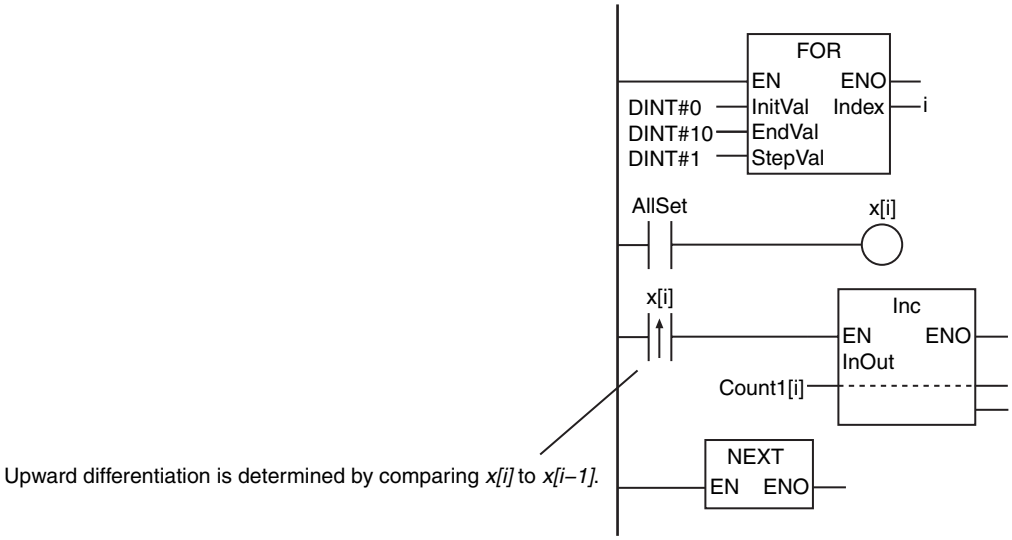
- The FOR-NEXT loops can be nested up to 15 levels, but count all nesting levels for the following instructions: IF, CASE, FOR, WHILE, and REPEAT.
- If loops are nested, you will need one BREAK instruction (or one EXIT instruction in ST) for each nesting level to cancel all repeat processing.
- Do not use Jump Instructions (e.g., the JMP instruction) to interrupt repeat processing. Always use a BREAK instruction (or an EXIT instruction in ST) to cancel repeat processing.
- You can change the values of *StepVal* and *EndVal* during repeat processing. You cannot change the value of *InitVal* during repeat processing.
- If the value of *StepVal* is 0, a task execution timeout occurs.
- Use the same data type for *InitVal*, *EndVal*, *StepVal*, and *Index*. Otherwise, a building error will occur.

- The value of Index after repeat processing is different in a ladder diagram and ST. In a ladder diagram, the value of *StepVal* is not added to *Index* at the end of repeat processing. In ST, the value of *StepVal* is added to *Index* at the end of repeat processing. Processing is repeated the same number of times.

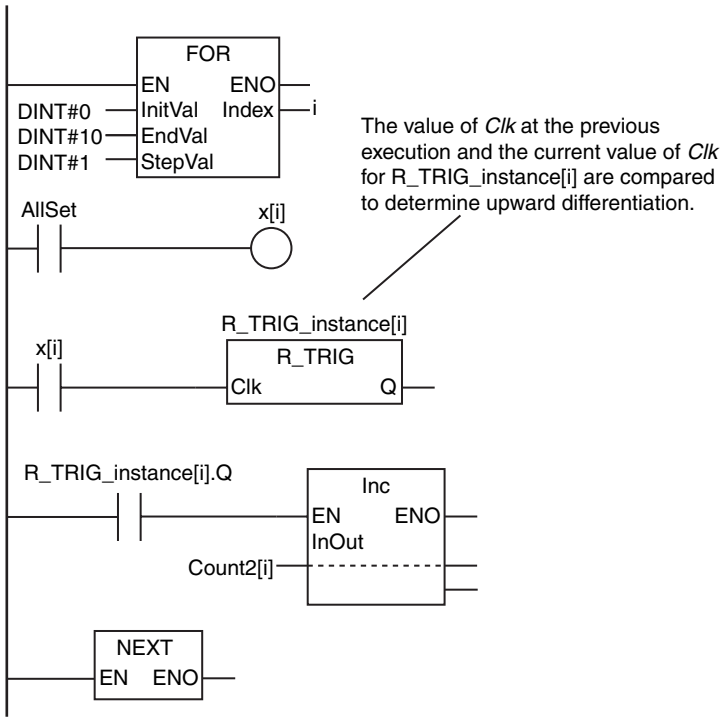
  The following example is for when *InitVal* is 1, *EndVal* is 100 and *StepVal* is 1.
  Ladder diagram: The value of *Index* will be 100 after 100 repetitions.
  ST: The value of *Index* will be 101 after 100 repetitions.

- Caution is required when you specify upward or downward differentiation for a LD, AND, or OR instruction in a FOR loop in a ladder diagram and an array is used for the LD, AND, or OR instruction.

  For upward or downward differentiation, the value of the specified variable at the previous execution is compared with the value of the specified variable at the current execution to determine upward or downward differentiation. Normally, the value of the specified variable does not change every time the instruction is executed. However, if an array is specified in a FOR loop, the array element changes each time the instruction is executed. Therefore, upward or downward differentiation is determined by comparing different array elements. In the following programming, the LD instruction in the third execution of the FOR loop (x = 2) compares the current value *x[2]* to the value of the specified variable the last time the LD instruction was executed, *x[1],* to determine that the value did not change to TRUE. As a result, *Count1[2]* is not incremented.



In the following programming, upward differentiation of *x[i]* is determined by the R_TRIG instruction. An instance of the R_TRIG instruction is provided for each element of *x[i]*, so it is possible to detect which element of *x[i]* changed its value. As a result, *Count2[0]* to *Count2[10]* are all incremented.



BREAK

The BREAK instruction is used to cancel repeat processing from the lowest level FOR instruction to the NEXT instruction.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BREAK	Break Loop	FUN	BREAK EN ENO	FOR Index:=0 TO 9 BY 1 DO IF Error[index] THEN EXIT*; END_IF END_FOR; * In ST, use EXIT instead of BREAK for the BREAK LOOP instruction. The meaning is the same.

Variables

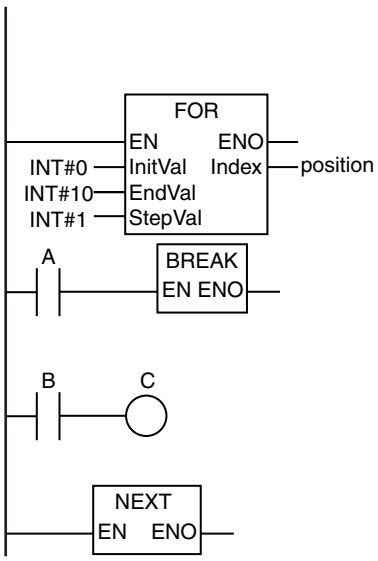
None

Function

The BREAK (EXIT) instruction cancels the repeat processing from the lowest level FOR instruction to the NEXT instruction (the END_FOR instruction for ST). It moves processing to the next instruction after the NEXT instruction. The processing between the BREAK instruction and the NEXT instruction (or the EXIT instruction in ST) will not be executed.

The following figure shows a programming example. When the FOR loop is executed, the value of variable *A* is checked each time. If the value of variable *A* is TRUE, the repeat processing is ended immediately. In this example, the Out instruction after the BREAK instruction is not executed, and the value of variable *C* is retained. (In ST, the EXIT instruction is used instead of the BREAK instruction.)

LD



ST

```
FOR position:=INT#0 TO INT#10 BY INT#1 DO
        IF (A=TRUE) THEN
                EXIT;
        END_IF;
        C:=B;
END_FOR;
```

Precautions for Correct Use

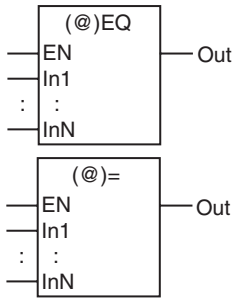
- Always place this instruction between the FOR and NEXT instructions (or the FOR and END_FOR instructions in ST).
- If FOR-NEXT loops (or FOR-END_FOR loops in ST) are nested, you will need one BREAK instruction (or one EXIT instruction in ST) for each nesting level to cancel all repeat processing.
- Do not use Jump Instructions (e.g., the JMP instruction) to interrupt repeat processing. Always use a BREAK instruction (or an EXIT instruction in ST) to cancel repeat processing.

Comparison Instructions

Instruction	Name	Page
EQ (=)	Equal	2-84
NE (<>)	Not Equal	2-86
LT (<), LE (<=), GT (>), and GE (>=)	Less Than/Less Than Or Equal/ Greater Than/Greater Than Or Equal	2-88
EQascii	Text String Comparison Equal	2-91
NEascii	Text String Comparison Not Equal	2-93
LTascii, LEascii, GTascii, and GEascii	Text String Comparison Less Than/Text String Comparison Less Than or Equal Text String Comparison Greater Than/Text String Comparison Greater Than or Equal	2-95
Cmp	Compare	2-98
ZoneCmp	Zone Comparison	2-100
TableCmp	Table Comparison	2-102
AryCmpEQ and AryCmpNE	Array Comparison Equal/ Array Comparison Not Equal	2-105
AryCmpLT, AryCmpLE, AryCmpGT, and AryCmpGE	Array Comparison Less Than/Array Comparison Less Than Or Equal Array Comparison Greater Than/Array Comparison Greater Than Or Equal	2-107
AryCmpEQV and AryCmpNEV	Array Value Comparison Equal/Array Value Comparison Not Equal	2-110
AryCmpLTV, AryCmpLEV, AryCmpGTV, and AryCmpGEV	Array Value Comparison Less Than/Array Value Comparison Less Than Or Equal Array Value Comparison Greater Than/Array Value Comparison Greater Than Or Equal	2-112

EQ (=)

The EQ (=) instruction determines if the contents of two or more variables are all equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EQ (=)	Equal	FUN		Out:=(In1=In2) & (In2=In3) & ··· & (InN-1=InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison data	Input	Values to compare, N = 2 to 5	Depends on data type.	---	0*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

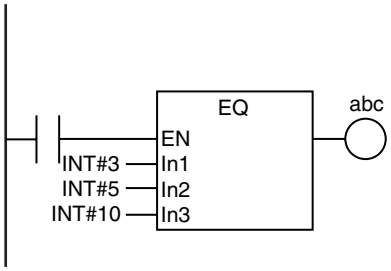
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	Enumerations can also be specified.																			
Out	OK																			

Function

The EQ (=) instruction determines if the contents of from two to five variables *In1* to *InN* are all equivalent. The comparison result *Out* is TRUE only when all values are equivalent. Otherwise, the value of *Out* is FALSE.

The following example is for when *In1* is INT#3, *In2* is INT#5 and *In3* is INT#10. The value of variable *abc* will be FALSE.

LD



ST

```
abc:=(INT#3=INT#5)&(INT#5=INT#10);
```

Additional Information

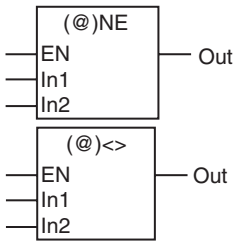
- The functions of the EQ instruction and the = instruction are exactly the same. Use the form that is easier to use.
- Use the EQascii instruction (page 2-91) to determine if text strings are equal.

Precautions for Correct Use

- If the data types of *In1* to *InN* are different, they will be expanded to a data type that includes the ranges of all of the data types.
- You cannot compare bit string data (BYTE, WORD, DWORD, or LWORD) with integers. You cannot compare bit string data to real number data (SINT, INT, DINT, LINT, USINT, UDINT, ULINT, REAL, and LREAL).
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- You can compare enumerations only to other enumerations.
- If *In1* to *InN* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

NE (<>)

The NE (<>) instruction determines if the contents of two variables are not equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NE (<>)	Not Equal	FUN		Out:=(In1<>In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 and In2	Comparison data	Input	Values to compare	Depends on data type.	---	*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

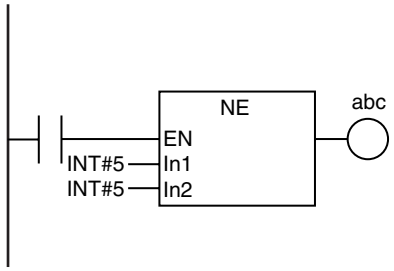
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 and In2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	Enumerations can also be specified.																			
Out	OK																			

Function

The NE (<>) instruction determines if the contents of two variables *In1* and *In2* are not equivalent. If they are not equivalent, the comparison result *Out* is TRUE. If they are equivalent, *Out* is FALSE.

The following example is for when *In1* equals *In2* (both have a value of INT#5). The value of variable *abc* will be FALSE.

LD



ST

```
abc:=(INT#5<>INT#5);
```

Additional Information

- The functions of the NE instruction and the <> instruction are exactly the same. Use the form that is easier to use.
- Use the NEascii instruction (page 2-93) to determine if text strings are not equal.

Precautions for Correct Use

- If the data types of *In1* and *In2* are different, the smaller one is expanded to a data type that includes the ranges of both of the data types.
- You cannot compare bit string data (BYTE, WORD, DWORD, or LWORD) with integers (SINT, INT, DINT, LINT, USINT, UDINT, ULINT). You cannot compare bit string data with real number data (REAL and LREAL).
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- You can compare enumerations only to other enumerations.
- If *In1* and *In2* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If the value of either *In1* or *In2* is nonnumeric data, the value of *Out* is FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

LT (<), LE (<=), GT (>), and GE (>=)

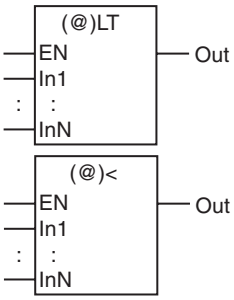
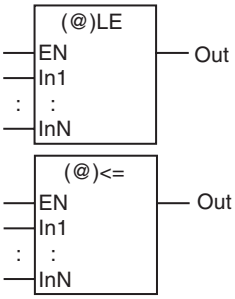
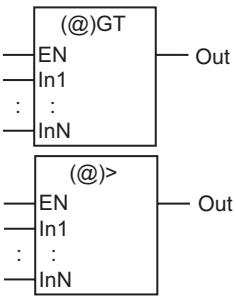
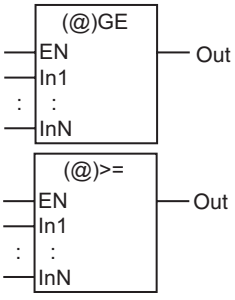
These instructions compare the sizes of two or more values.

LT (<): Performs a less than comparison.

LE (<=): Performs a less than or equal comparison.

GT (>): Performs a greater than comparison.

GE (>=): Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LT (<)	Less Than	FUN		Out:=(In1<In2) & (In2<In3) & ··· & (InN-1<InN);
LE (<=)	Less Than Or Equal	FUN		Out:=(In1<=In2) & (In2<=In3) & ··· & (InN-1<=InN);
GT (>)	Greater Than	FUN		Out:=(In1>In2) & (In2>In3) & ··· & (InN-1>InN);
GE (>=)	Greater Than Or Equal	FUN		Out:=(In1>=In2) & (In2>=In3) & ··· & (InN-1>=InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison data	Input	Values to compare, N = 2 to 5	Depends on data type.	---	0*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out	OK																			

Function

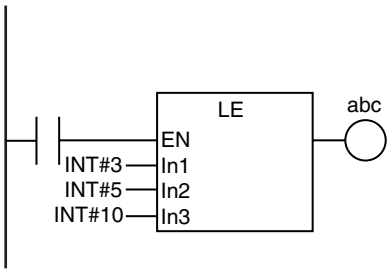
These instructions compare the values of *In1* to *InN* (N = 2 to 5).

The output value *Out* is shown below for each instruction.

Instruction	Value of *Out*
LT (<)	If *In1* < *In2* < ... < *InN*, *Out* is TRUE. Otherwise, it is FALSE.
LE (<=)	If *In1* <= *In2* <= ... <= *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GT (>)	If *In1* > *In2* > ... > *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GE (>=)	If *In1* >= *In2* >= ... >= *InN*, *Out* is TRUE. Otherwise, it is FALSE.

The following example shows the LE instruction when *In1* is INT#3, *In2* is INT#5 and *In3* is INT#10. The value of variable *abc* will be TRUE.

LD



ST

```
abc:=(INT#3<= INT#5)&(INT#5<=INT#10);
```

Additional Information

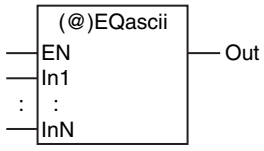
- The functions of the LT and < instructions, the LE and <= instructions, the GT and > instructions, and the GE and >= instructions are exactly the same. Use the form that is easier to use.
- Use the LTascii, LEascii, GTascii, and GEascii instructions (page 2-95) to compare the sizes of text strings.

Precautions for Correct Use

- If the data types of *In1* to *InN* are different, they will be expanded to a data type that includes the ranges of all of the data types.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- If *In1* to *InN2* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

EQascii

The EQascii instruction determines if two or more text strings are all equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EQascii	Text String Comparison Equal	FUN		Out:=EQascii(In1, ··, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison text strings	Input	Text strings to compare, N = 2 to 5	Depends on data type.	---	'' *
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

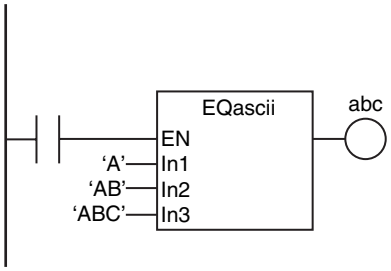
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN																				OK
Out	OK																			

Function

The EQascii instruction determines if from two to five text strings *In1* to *InN* are all equivalent. If the are all equivalent, comparison result *Out* changes to TRUE. Otherwise, the value of *Out* is FALSE. “Equivalent” means that both the lengths and contents of the text strings are the same.

The following example is for when *In1* is “A”, *In2* is “AB”, and *In3* is “ABC”. The value of variable *abc* will be FALSE.

LD



ST

```
abc:=EQascii('A', 'AB', 'ABC');
```

Additional Information

The text string comparison instructions are convenient when you want to reorder text strings according to the character codes. For example, the character codes for alphabet characters are in the same order as the alphabet characters. This allows you to alphabetize.

Precautions for Correct Use

- Do not use this instruction as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- Specify text strings that contain only ASCII characters for *In1* to *InN*.
- An error occurs in the following case. *Out* will be FALSE.
  - One of the text strings in *In1* to *InN* does not end in a NULL character.

NEascii

The NEascii instruction determines if two text strings are not equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NEascii	Text String Comparison Not Equal	FUN	(@)NEascii EN In1 In2 Out	Out:=NEascii(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 and In2	Comparison text strings	Input	Text strings to compare	Depends on data type.	---	*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

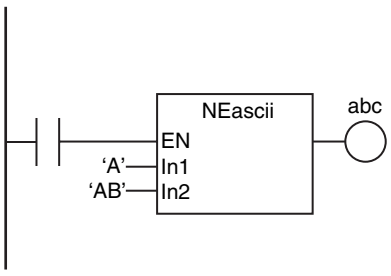
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 and In2																				OK
Out	OK																			

Function

The NEascii instruction determines if two text strings *In1* and *In2* are not equivalent. If they are different, comparison result *Out* will be TRUE. If they are the same, comparison result *Out* will be FALSE. “Equivalent” means that both the lengths and contents of the text strings are the same.

The following example is for when *In1* is “A” and *In2* is “AB”. The value of variable *abc* will be TRUE.

LD



ST

```
abc:=NEascii('A', 'AB');
```

Additional Information

The text string comparison instructions are convenient when you want to reorder text strings according to the character codes. For example, the character codes for alphabet characters are in the same order as the alphabet characters. This allows you to alphabetize.

Precautions for Correct Use

- Do not use this instruction as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- Specify text strings that contain only ASCII characters for *In1* and *In2*.
- An error occurs in the following case. *Out* will be FALSE.
  - The text string in *In1* or *In2* does not end in a NULL character.

LTascii, LEascii, GTascii, and GEascii

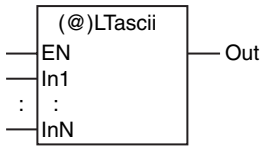
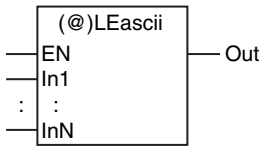
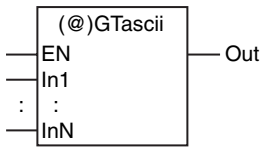
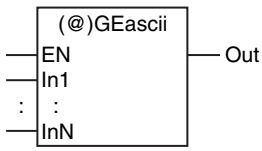
These instructions compare the sizes of two or more text strings.

LTascii: Performs a less than comparison.

LEascii: Performs a less than or equal comparison.

GTascii: Performs a greater than comparison.

GEascii: Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LTascii	Text String Comparison Less Than	FUN		Out:=LTascii(In1, ···, InN);
LEascii	Text String Comparison Less Than or Equal	FUN		Out:=LEascii(In1, ···, InN);
GTascii	Text String Comparison Greater Than	FUN		Out:=GTascii(In1, ···, InN);
GEascii	Text String Comparison Greater Than or Equal	FUN		Out:=GEascii(In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison text strings	Input	Text strings to compare, N = 2 to 5	Depends on data type.	---	''*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN																				OK
Out	OK																			

Function

These instructions compare the sizes of from two to five text strings in *In1* to *InN* (N = 2 to 5). The output value *Out* is shown below for each instruction.

Instruction	Value of *Out*
LTascii	If *In1* < *In2* <...< *InN*, *Out* is TRUE. Otherwise, it is FALSE.
LEascii	If *In1* ≤ *In2* ≤...≤ *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GTascii	If *In1* > *In2* >...> *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GEascii	If *In1* ≥ *In2* ≥...≥ *InN*, *Out* is TRUE. Otherwise, it is FALSE.

The sizes of the character codes are compared. The comparison procedure is as follows:

First, the first character codes in all of the text strings are compared. If the character codes are different, the result of the size comparison for the text strings is determined by the size relationship between those character codes. If the character codes are the same, comparison continues in order to the other characters until a different character code is found. If the lengths of the text strings are different, NULL characters (16#00) are added to the shorter text string to complete the comparison.

The relationships between various text strings are as follows:

'AD'(16#414400) < 'BC'(16#424400)

'ADC' (16#41444300)< 'B'(16#42000000)

'ABC' (16#41424300)< 'ABD'(16#41424400)

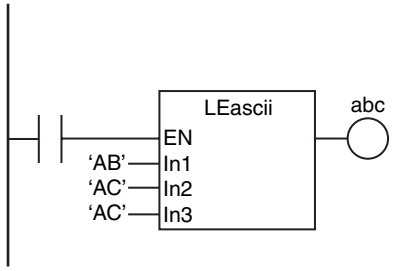
'ABC' (16#41424300)> 'AB'(16#41420000)

'AB' (16#414200)= 'AB'(16#414200)

If the text string contains multi-byte characters, the characters are separated into individual bytes before comparison. For example, the two-byte character 16#C281 is handled as 16#C2 and 16#81.

The following example for the LEascii instruction is for when *In1* is “AB”, *In2* is “AC”, and *In3* is “AC”. The value of variable *abc* will be TRUE.

LD



ST

```
abc:=LEascii('AB', 'AC', 'AC');
```

Additional Information

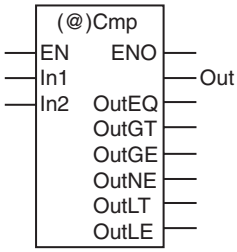
The text string comparison instructions are convenient when you want to reorder text strings according to the character codes. For example, the character codes for alphabet characters are in the same order as the alphabet characters. This allows you to alphabetize.

Precautions for Correct Use

- Do not use this instruction as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- Specify text strings that contain only ASCII characters for *In1* to *InN*.
- An error occurs in the following case. *Out* will be FALSE.
  - One of the text strings in *In1* to *InN* does not end in a NULL character.

Cmp

The Cmp instruction compares two values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Cmp	Compare	FUN		Out:=Cmp(In1, In2, OutEQ, OutGT, OutGE, OutNE, OutLT, OutLE); You can omit *Out*.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 and In2	Comparison data	Input	Values to compare	Depends on data type.	---	*
Out	Return value	Output	Always TRUE	TRUE only	---	---
OutEQ	Equal flag		Equal flag	Depends on data type.		
OutGT	Greater than flag		Greater than flag			
OutGE	Greater than or equal flag		Greater than or equal flag			
OutNE	Not equal flag		Not equal flag			
OutLT	Less than flag		Less than flag			
OutLE	Less than or equal flag		Less than or equal flag			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 and In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out	OK																			
OutEQ	OK																			
OutGT	OK																			
OutGE	OK																			
OutNE	OK																			
OutLT	OK																			
OutLE	OK																			

Function

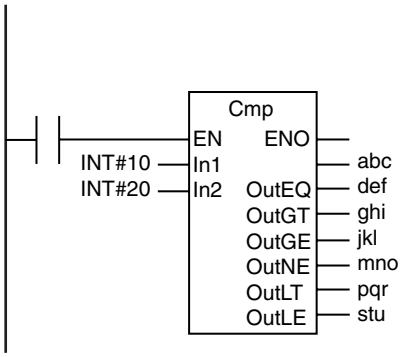
The Cmp instruction compares two values (*In1* and *In2*) and outputs flag values.

The values of the flags are as follows:

Flag	Value
OutEQ	If *In1* equals *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutGT	If *In1* is greater than *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutGE	If *In1* is greater than or equal to *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutNE	If *In1* is not equal to *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutLT	If *In1* is less than *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutLE	If *In1* is less than or equal to *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.

The following example is for when *In1* is INT#10 and *In2* is INT#20. The values of variables *def*, *ghi*, and *jkl* will be FALSE, and the values of *abc*, *mno, pqr*, and *stu* will be TRUE.

LD



ST

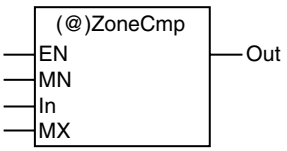
```
abc:=Cmp(INT#10, INT#20, def, ghi, jkl, mno, pqr, stu);
```

Precautions for Correct Use

- If the data types of *In1* and *In2* are different, the smaller one is expanded to a data type that includes the ranges of both of the data types.
- If *In1* and *In2* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If the value of either *In1* or *In2* is nonnumeric data, the values of *OutEQ*, *OutGT*, *OutGE*, *OutNE*, *OutLT*, and *OutLE* are FALSE.

ZoneCmp

The ZoneCmp instruction determines if the comparison data is within the specified maximum and minimum values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ZoneCmp	Zone Comparison	FUN		Out:=ZoneCmp(MN, In, MX);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
MN	Minimum value	Input	Minimum value	Depends on data type.	---	0
In	Comparison data		Value to compare			*
MX	Maximum value		Maximum value			0
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

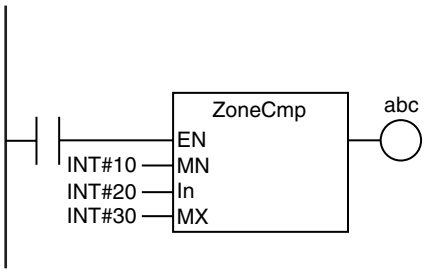
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
MX						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out	OK																			

Function

The ZoneCmp instruction determines if comparison data *In* is between maximum value *MX* and minimum value *MN*. If MX ≥ In ≥ MN, *Out* will be TRUE. Otherwise, *Out* will be FALSE.

The following example is for when *MN* is INT#10, *In* is INT#20 and *MX* is INT#30. The value of variable *abc* will be TRUE.

LD



ST

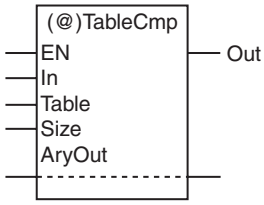
```
abc:=ZoneCmp(INT#10, INT#20, INT#30);
```

Precautions for Correct Use

- If the data types of *In, MX,* and *MN* are different, they will be expanded to a data type that includes the ranges of all of the data types.
- If *In*, *MX*, and *MN* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If the value of *In* is nonnumeric data, the value of *Out* is FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following cases. *Out* will be FALSE.
  - The value of *MN* is greater than the value of *MX.*
  - Either *MX* or *MN* contains nonnumeric data.

TableCmp

The TableCmp instruction compares the comparison data with multiple defined ranges in a comparison table.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TableCmp	Table Comparison	FUN		Out:=TableCmp(In, Table, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Comparison data	Input	Value to compare	Depends on data type.	---	*
Table[] (two-dimensional array)	Comparison table		Two-dimensional array that contains the elements for the defined ranges			
Size	Comparison size		Number of elements in *Table[]* to which to compare *In*			1
AryOut[] (array)	Individual comparison results array	In-out	Comparison results for *Table[]* elements TRUE: Condition met. FALSE: Condition not met.	Depends on data type.	---	---
Out	Comparison result	Output	TRUE: *In* meets all comparison conditions for elements of *Table[].* FALSE: The comparison condition is not met for one or more sets of elements.	Depends on data type.	---	---

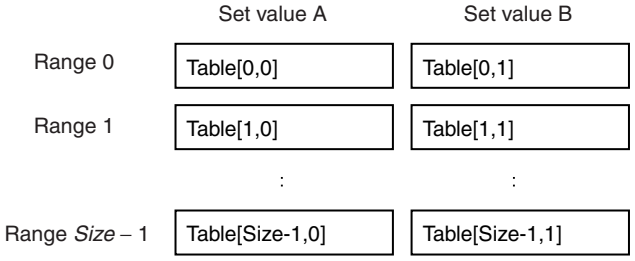
* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Table[] (two-dimensional array)	Must be a two-dimensional array with elements that have the same data type as *In*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

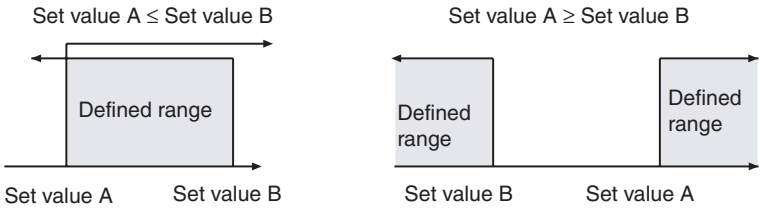
Function

The TableCmp instruction compares comparison data *In* with the number of defined ranges specified by the value of *Size* in comparison table *Table[]*.

Table[] is a two-dimensional array. The first dimension contains the numbers of the defined ranges. In the second dimension, element 0 is set value A of the defined range and element 1 is set value B of the defined range.

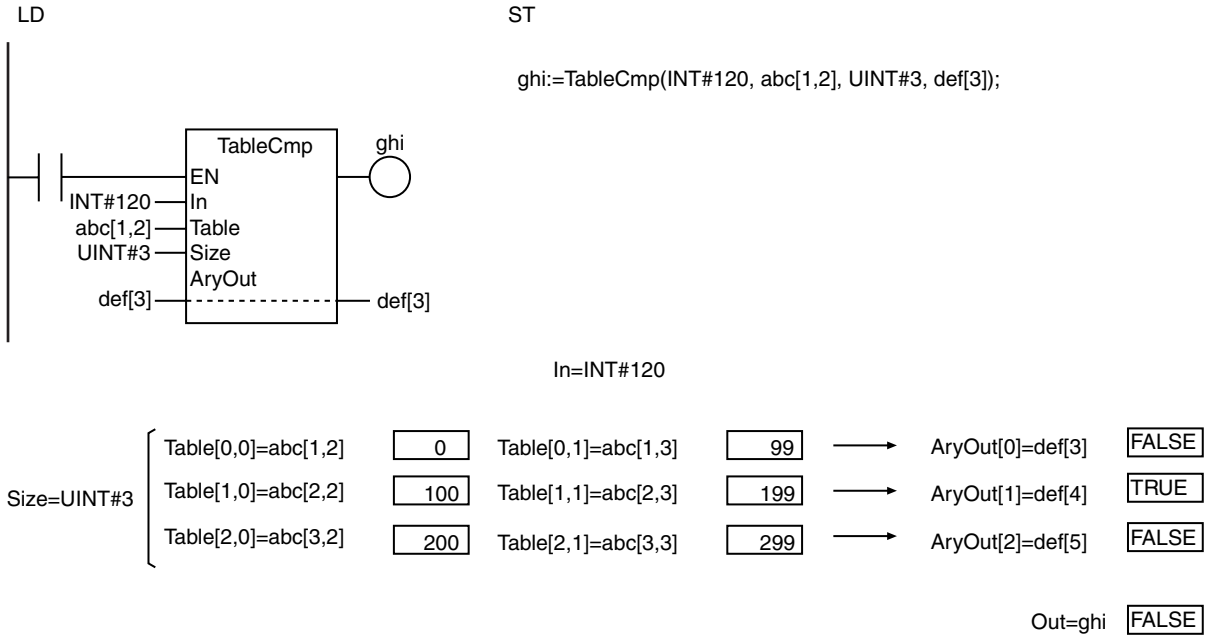


Set value A and set value B define range as shown below. Set value A and set value B are always included in the range.



The results of comparing *In* and *Table[]* are stored in individual comparison results array *AryOut[].* If *In* is within the defined range for element *i*, *ArayOut[i]* will be TRUE. If it is not within the range, *ArayOut[i]* will be FALSE. If all *Size* elements of *AryOut[]* are TRUE, comparison result *Out* will be TRUE. Otherwise, it will be FALSE.

The following example is for when *In* is INT#120 and *Size* is UINT#3.



Precautions for Correct Use

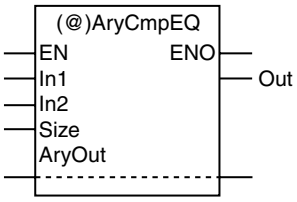
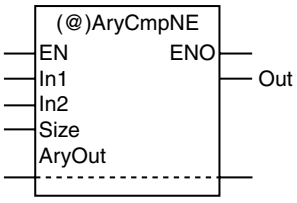
- Use the same data type for *In* and *Table[].* Otherwise, a compiling error will occur.
- Use a two-dimensional array for *Table[].* A compiling error will occur if you use any other size of array.
- If an array with more than two dimensions is used for *Table[]*, the elements in the third and higher dimensions are ignored.
- If the *AryOut[]* array is larger than the value of *Size*, the comparison results will be stored in *AryOut[0]* to *AryOut[Size−1]*. Other elements of the array will not change.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- If real numbers are compared, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be FALSE and *AryOut[]* will not change.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following cases. *Out* will be FALSE.
  - If the value of *Size* exceeds the size of the *AryOut[]* array.
  - If the value of *Size* exceeds the size of the first dimension of the *Table[]* array.
  - The size of the second dimension of *Table []* is 1.

AryCmpEQ and AryCmpNE

These instructions compare the values of the elements of two arrays.

AryCmpEQ: Determines if the elements are equal.

AryCmpNE: Determines if the elements are not equal.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpEQ	Array Comparison Equal	FUN		AryCmpEQ(In1, In2, Size, AryOut);
AryCmpNE	Array Comparison Not Equal	FUN		AryCmpNE(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] and In2[] (arrays)	Comparison arrays	Input	Arrays containing the elements to compare	Depends on data type.	---	*
Size	Number of comparison elements		Number of elements to compare	Depends on data type.		1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2[] (array)	Must be an array with the same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

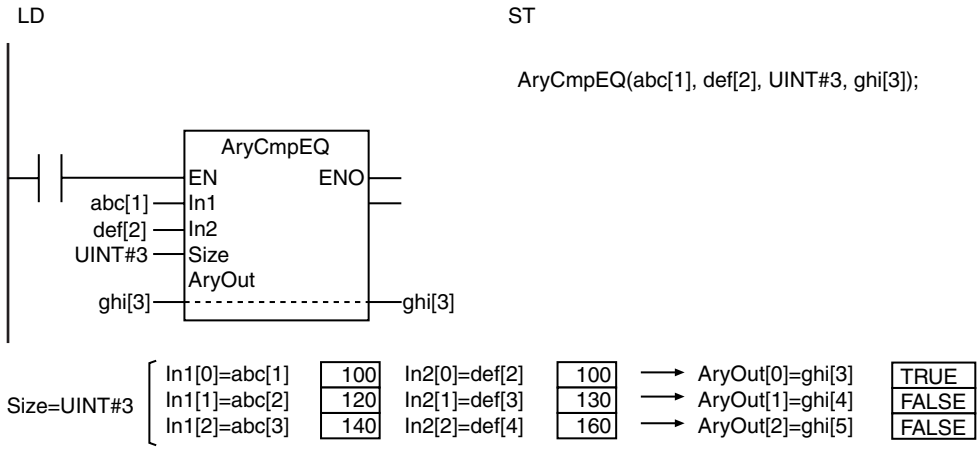
Function

These instructions compare the values of the elements with the same element numbers in two arrays (In1[0] to In1[*Size*–1] and In2[0] and In2[*Size*–1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (AryOut[0] to AryOut[*Size* –1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpEQ	If *In1[i]* = *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpNE	If *In1[i]* ≠ *In2[i]*, the result is TRUE. Otherwise, it is FALSE.

The following example shows the AryCmpEQ instruction when *Size* is UINT#3.



Precautions for Correct Use

- Use the same data type for *In1[]* and *In2[]*.
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* and *In2[]* contain real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If *In1[]* and *In2[]* contain different data types.
  - If the *In1[]*, *In2[]*, or *AryOut[]* array is smaller than the value of *Size.*

AryCmpLT, AryCmpLE, AryCmpGT, and AryCmpGE

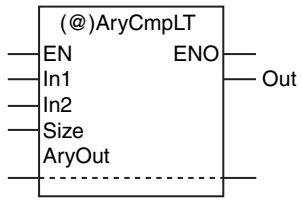
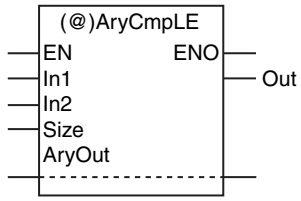
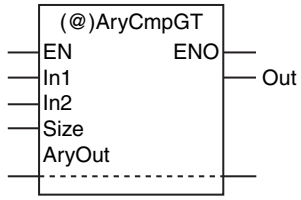
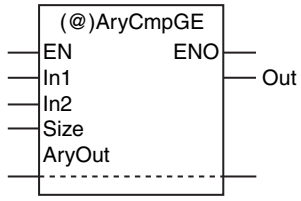
These instructions compare the values of the elements of two arrays.

AryCmpLT: Performs a less than comparison.

AryCmpLE: Performs a less than or equal comparison.

AryCmpGT: Performs a greater than comparison.

AryCmpGE: Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpLT	Array Comparison Less Than	FUN		AryCmpLT(In1, In2, Size, AryOut);
AryCmpLE	Array Comparison Less Than Or Equal	FUN		AryCmpLE(In1, In2, Size, AryOut);
AryCmpGT	Array Comparison Greater Than	FUN		AryCmpGT(In1, In2, Size, AryOut);
AryCmpGE	Array Comparison Greater Than Or Equal	FUN		AryCmpGE(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] and In2[] (arrays)	Comparison arrays	Input	Arrays containing the elements to compare	Depends on data type.	---	*
Size	Number of comparison elements		Number of elements to compare	Depends on data type.		1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK						
In2[] (array)	Must be an array with the same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

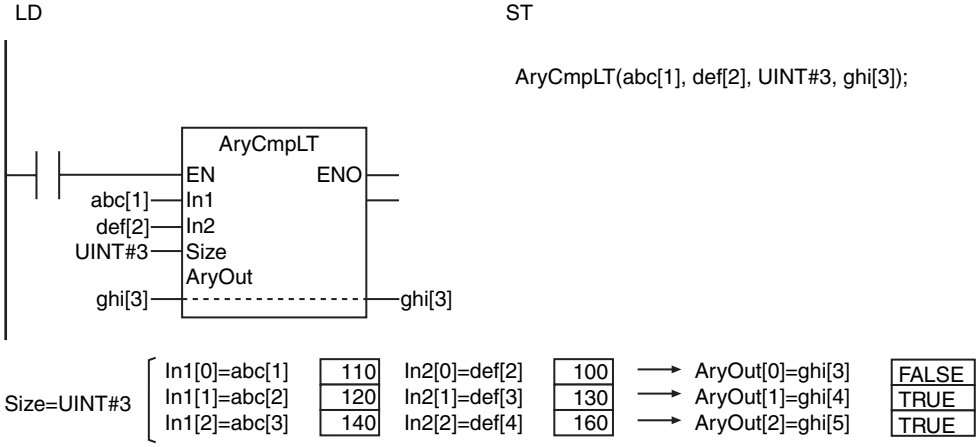
Function

These instructions compare the values of the elements with the same element numbers in two arrays (In1[0] to In1[*Size* −1] and In2[0] and In2[*Size* −1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (AryOut[0] to AryOut[*Size* −1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpLT	If *In1[i]* < *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpLE	If *In1[i]* <= *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGT	If *In1[i]* > *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGE	If *In1[i]* >= *In2[i]*, the result is TRUE. Otherwise, it is FALSE.

The following example shows the AryCmpLT instruction when *Size* is UINT#3.



Precautions for Correct Use

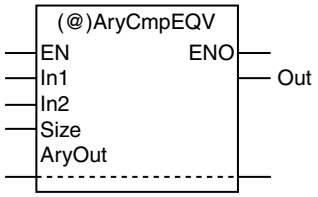
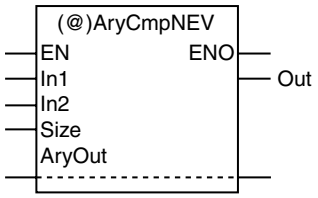
- Use the same data type for *In1[]* and *In2[].*
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* and *In2[]* contain real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If *In1[]* and *In2[]* contain different data types.
  - If the *In1[]*, *In2[]*, or *AryOut[]* array is smaller than the value of *Size.*

AryCmpEQV and AryCmpNEV

These instructions compare a value to the values of the elements of an array.

AryCmpEQV: Determines if the elements are equal.

AryCmpNEV: Determines if the elements are not equal.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpEQV	Array Value Comparison Equal	FUN		AryCmpEQV(In1, In2, Size, AryOut);
AryCmpNEV	Array Value Comparison Not Equal	FUN		AryCmpNEV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Comparison array	Input	Array containing the elements to compare	Depends on data type.	---	*
In2	Comparison value		Value to compare			
Size	Number of comparison elements		Number of elements to compare	Depends on data type.		1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2	Must be same data type as the elements of *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

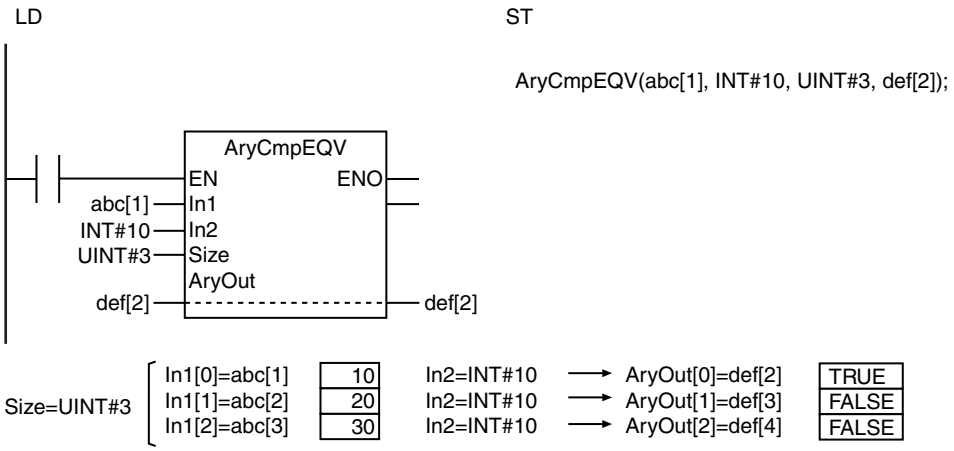
Function

These instructions compare comparison value *In2* with the specified elements in an array (*In1*[0] to *In1*[*Size* −1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (*AryOut*[0] to *AryOut*[*Size* −1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpEQV	If *In1[i]* = *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpNEV	If *In1[i]* ≠ *In2*, the result is TRUE. Otherwise, it is FALSE.

The following example shows the AryCmpEQV instruction when *In2* is INT#10 and *Size* is UINT#3.



Precautions for Correct Use

- Use the same data type for *In1[]* and *In2.*
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* contains real numbers and *In2* is a real number, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If the *In1[]* or *AryOut[]* array is smaller than the value of *Size.*

AryCmpLTV, AryCmpLEV, AryCmpGTV, and AryCmpGEV

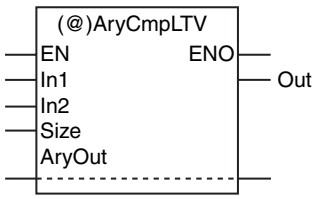
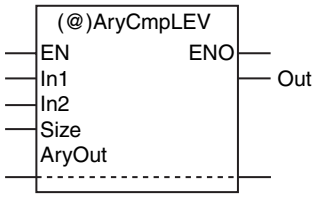
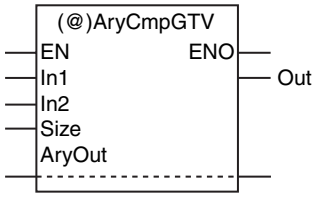
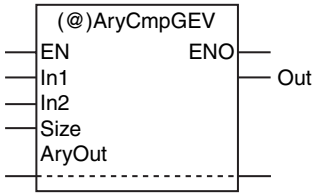
These instructions compare a value to the values of the elements of an array.

AryCmpLTV: Performs a less than comparison.

AryCmpLEV: Performs a less than or equal comparison.

AryCmpGTV: Performs a greater than comparison.

AryCmpGEV: Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpLTV	Array Value Comparison Less Than	FUN		AryCmpLTV(In1, In2, Size, AryOut);
AryCmpLEV	Array Value Comparison Less Than Or Equal	FUN		AryCmpLEV(In1, In2, Size, AryOut);
AryCmpGTV	Array Value Comparison Greater Than	FUN		AryCmpGTV(In1, In2, Size, AryOut);
AryCmpGEV	Array Value Comparison Greater Than Or Equal	FUN		AryCmpGEV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Comparison array	Input	Array containing the elements to compare	Depends on data type.	---	*
In2	Comparison value		Value to compare			
Size	Number of comparison elements		Number of elements to compare	Depends on data type.		1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK						
In2	Must be same data type as the elements of *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

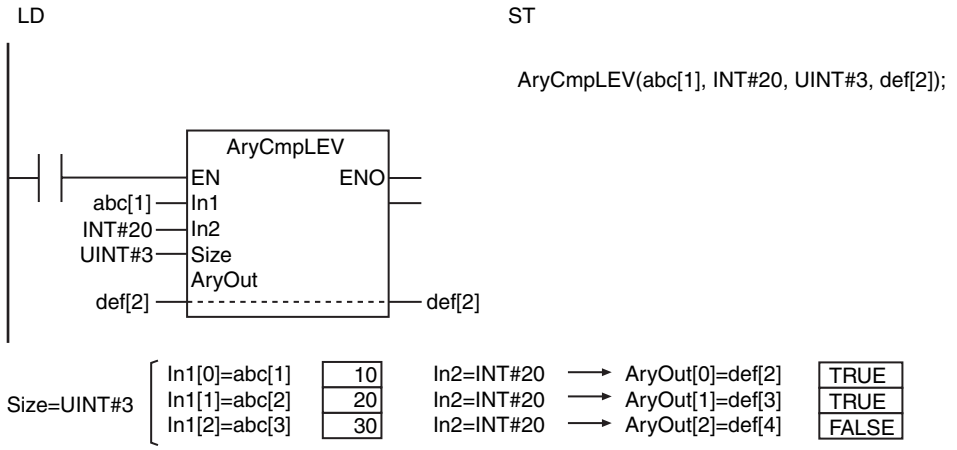
Function

These instructions compare comparison value *In2* with the specified elements in an array (In1[0] to In1[*Size* − 1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (AryOut[0] to AryOut[*Size* −1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpLTV	If *In1[i]* < *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpLEV	If *In1[i]* <= *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGTV	If *In1[i]* > *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGEV	If *In1[i]* >= *In2*, the result is TRUE. Otherwise, it is FALSE.

The following example shows the AryCmpLEV instruction when *In2* is INT#20 and *Size* is UINT#3.



Precautions for Correct Use

- Use the same data type for *In1[]* and *In2.*
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* contains real numbers and *In2* is a real number, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If the *In1[]* or *AryOut[]* array is smaller than the value of *Size.*

Timer Instructions

Instruction	Name	Page
TON	On-Delay Timer	2-116
TOF	Off-Delay Timer	2-120
TP	Timer Pulse	2-123
AccumulationTimer	Accumulation Timer	2-126
Timer	Hundred-ms Timer	2-129

TON

The TON instruction outputs TRUE when the set time elapses after the timer starts.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TON	On-Delay Timer	FB	TON_instance TON In Q PT ET	TON_instance (In, PT, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer start signal FALSE: Timer reset signal	Depends on data type.	---	FALSE
PT	Set time		Time from when timer starts until *Q* changes to TRUE	*	ms	0
Q	Timer output	Output	TRUE: Timer output ON FALSE: Timer output OFF	Depends on data type.	---	---
ET	Elapsed time		Elapsed time since timer started	*	ms	

* T#0ms to T#106751d_23h_47m_16s_854.775807ms

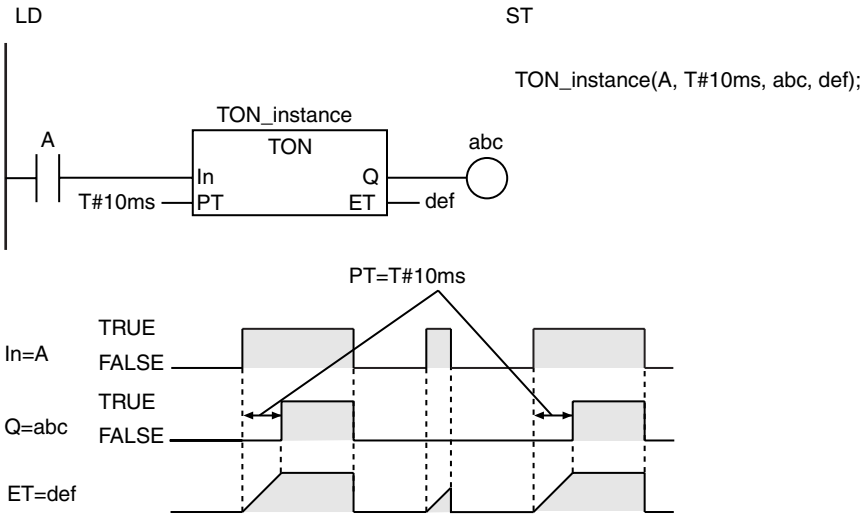
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Q	OK																			
ET																OK				

Function

The TON instruction outputs TRUE when the set time elapses after the timer starts. Set the time in nanoseconds (ns). The timing accuracy is 100 ns. The timer starts when timer input *In* changes to TRUE. Elapsed time *ET* is incremented as time elapses. When *ET* reaches set time *PT*, timer output *Q* changes to TRUE. *ET* is not incremented after that. The timer is reset when *In* changes to FALSE. *ET* changes to 0 and *Q* changes to FALSE.

If the timer is started and then *In* changes to FALSE before *ET* reaches *PT*, the timer is reset.

The following figure shows a programming example and timing chart when *PT* is T#10ms. Variable *abc* will change to TRUE 10 ms after variable *A* changes to TRUE.



Additional Information

- Use the TP instruction (page 2-123) for a timer that changes the timer output to TRUE when timing starts and then changes the timer output to FALSE when the set time is reached.
- Use the TOF instruction (page 2-120) for a timer that starts when *In* changes to FALSE and then changes the timer output to FALSE when the elapsed time reaches the set time.
- To reduce timer execution time, use the Timer instruction (page 2-129), which times in increments of 100 ms.

Precautions for Correct Use

- *ET* and *Q* are updated only when the instruction is executed. Therefore, *Q* does not change to TRUE precisely when the elapsed time from when the timer starts reaches *PT*. *Q* changes to TRUE the next time the instruction is executed after the elapsed time from when the timer starts reaches *PT*. The change in *Q* can therefore occur with a delay of up to one task period.
- Set *PT* and *ET* in nanoseconds (ns), but remember the timing accuracy is 100 ns.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If T#0ms or a negative number is set for *PT*, *Q* will change to TRUE as soon as the value of *In* changes to TRUE.
- You can change the value of *PT* while the value of *In* is TRUE. Operation is as follows:

Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	TRUE	---	The value of *Q* remains TRUE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	FALSE	$PT \geq ET$	Timing is continued. When the value of *ET* reaches the value of *PT*, the value of *Q* changes to TRUE and *ET* is no longer incremented.
		$PT < ET$	The value of *Q* changes to TRUE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, the timer is reset. The value of *ET* changes to 0 and the value of *Q* changes to FALSE.

- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

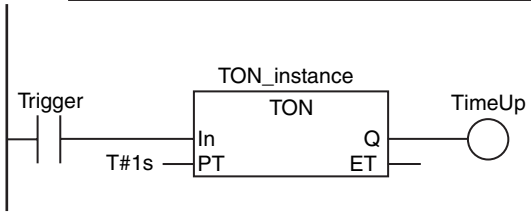
Sample Programming

● Measuring Time with One On-Delay Timer

The value of *TimeUp* will change to TRUE 1 second after the value of *Trigger* changes to TRUE.

LD

Variable	Data type	Initial value	Comment
Trigger	BOOL	False	Execution condition
TimeUp	BOOL	False	Timer output
TON_instance	TON		



ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	False	Execution condition
TimeUp	BOOL	False	Timer output
TON_instance	TON		

```
IF (Trigger=TRUE) THEN
    TON_instance(In:=TRUE, PT:=T#1s, Q=>TimeUp);
ELSE
    TON_instance(In:=FALSE, Q=>TimeUp);
END_IF;
```

The following ST programming performs the same operation.

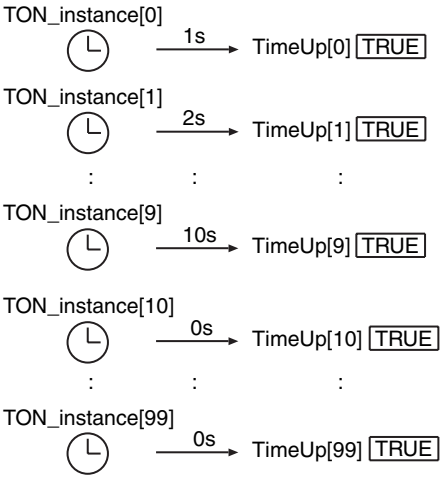
ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	False	Execution condition
TimeUp	BOOL	False	Timer output
TON_instance	TON		

```
TON_instance(In:=Trigger, PT:=T#1s, Q=>TimeUp);
```

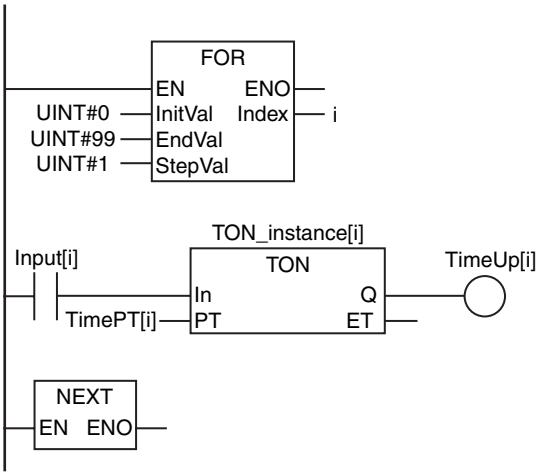
● Measuring Time with Multiple On-Delay Timers

In this example, a total of 100 instances of the On-Delay Timer instruction, TON_instance[0] to TON_instance[99], are programmed. Each timer starts when the value of the corresponding timer input *Input[0]* to *Input[99]* changes to TRUE.
The timers for the first 10 instances, TON_instance[0] to TON_instance[9], change the corresponding values in *TimeUp[i]* to TRUE i+1 seconds (i = 0 to 9) after execution is started.
The timers for the remaining 90 instances, TON_instance[10] to TON_instance[99], change the corresponding values in *TimeUp[i]* (i = 10 to 99) to TRUE as soon as execution is started.



LD

Variable	Data type	Initial value	Comment
Input	ARRAY[0..99] OF BOOL	[100(False)]	Timer input
TimeUp	ARRAY[0..99] OF BOOL	[100(False)]	Timer output
TimePT	ARRAY[0..99] OF TIME	[T#1s, T#2s, T#3s, T#4s, T#5s, T#6s, T#7s, T#8s, T#9s, T#10s, 90(T#0s)]	Set time
TON_instance	ARRAY[0..99] OF TON		
i	UINT	0	Index



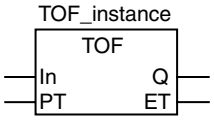
ST

Variable	Data type	Initial value	Comment
Input	ARRAY[0..99] OF BOOL	[100(False)]	Timer input
TimeUp	ARRAY[0..99] OF BOOL	[100(False)]	Timer output
TimePT	ARRAY[0..99] OF TIME	[T#1s, T#2s, T#3s, T#4s, T#5s, T#6s, T#7s, T#8s, T#9s, T#10s, 90(T#0s)]	Set time
TON_instance	ARRAY[0..99] OF TON		
i	UINT	0	Index

```
FOR i:=0 TO 99 DO
    TON_instance[i](
        In := Input[i],
        PT:= TimePT[i],
        Q =>TimeUp[i]);
END_FOR;
```

TOF

The TOF instruction outputs FALSE when the set time elapses after the timer starts.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TOF	Off-Delay Timer	FB		TOF_instance (In, PT, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer reset signal FALSE: Timer start signal	Depends on data type.	---	FALSE
PT	Set time		Time from when timer starts until *Q* changes to FALSE	*	ms	0
Q	Timer output	Output	TRUE: Timer output ON FALSE: Timer output OFF	Depends on data type.	---	---
ET	Elapsed time		Elapsed time since timer started	*	ms	

* T#0ms to T#106751d_23h_47m_16s_854.775807ms

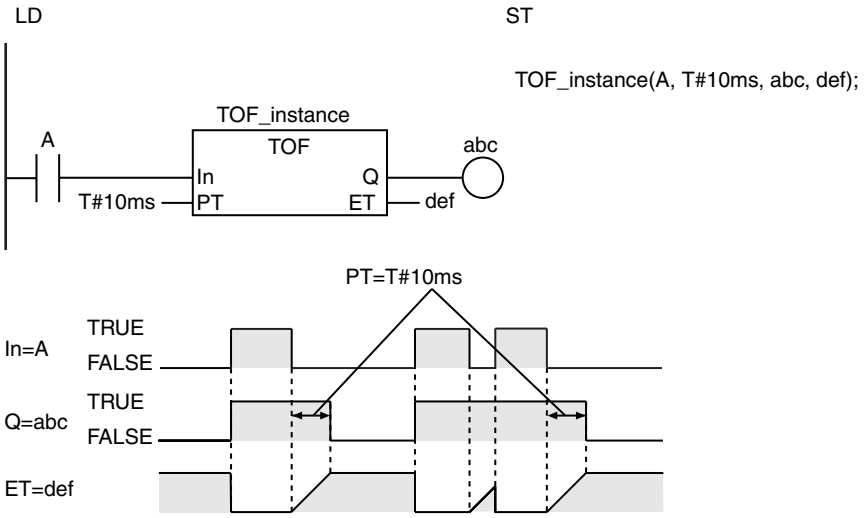
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Q	OK																			
ET																OK				

Function

The TOF instruction outputs FALSE when the set time elapses after the timer starts. Set the time in nanoseconds (ns). The timing accuracy is 100 ns. The timer starts when timer input *In* changes to FALSE. Elapsed time *ET* is incremented as time elapses. When *ET* reaches set time *PT*, timer output *Q* changes to FALSE. *ET* is not incremented after that. The timer is reset when *In* changes to TRUE. *ET* changes to 0 and *Q* changes to TRUE.

If the timer is started and then *In* changes to FALSE before *ET* reaches *PT*, the timer is reset.

The following figure shows a programming example and timing chart for a *PT* of T#10ms. Variable *abc* will change to FALSE 10 ms after variable *A* changes to FALSE.



Additional Information

- Use the TP instruction (page 2-123) for a timer that changes the timer output to TRUE when timing starts and then changes the timer output to FALSE when the set time is reached.
- Use the TON instruction (page 2-116) for a timer that starts when *In* changes to TRUE and then changes the timer output to TRUE when the elapsed time reaches the set time.

Precautions for Correct Use

- *ET* and *Q* are updated only when the instruction is executed. Therefore, *Q* does not change to FALSE precisely when the elapsed time from when the timer starts reaches *PT*. *Q* changes to FALSE the next time the instruction is executed after the elapsed time from when the timer starts reaches *PT*. The change in *Q* can therefore occur with a delay of up to one task period.
- Set *PT* and *ET* in nanoseconds (ns), but remember the timing accuracy is 100 ns.
- If T#0ms or a negative number is set for *PT*, *Q* will change to FALSE as soon as the value of *In* changes to FALSE.
- The value of *Q* changes to TRUE immediately after execution of this instruction regardless of the value of *In*. *Q* is FALSE from only when the timer is started until the time that is set with *PT* elapses.
- You can change the value of *PT* while the value of *In* is FALSE. Operation is as follows:

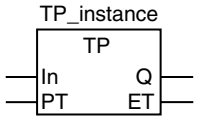
Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	FALSE	---	The value of *Q* remains FALSE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	TRUE	$PT \geq ET$	Timing is continued. When the value of *ET* reaches the value of *PT*, the value of *Q* changes to FALSE and *ET* is no longer incremented.
		$PT < ET$	The value of *Q* changes to FALSE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - The value of *ET* changes to 0 and the value of *Q* changes to TRUE.
  - If an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE.
  - Timing starts as soon as the reset is released.

- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

TP

The TP instruction outputs TRUE while the set time elapses after the timer starts.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TP	Timer Pulse	FB		TP_instance (In, PT, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer start signal FALSE: Timer reset signal	Depends on data type.	---	FALSE
PT	Set time		Time that Q remains at TRUE	*	ms	0
Q	Timer output	Output	TRUE: Timer output ON FALSE: Timer output OFF	Depends on data type.	---	---
ET	Elapsed time		Elapsed time since timer started	*	ms	

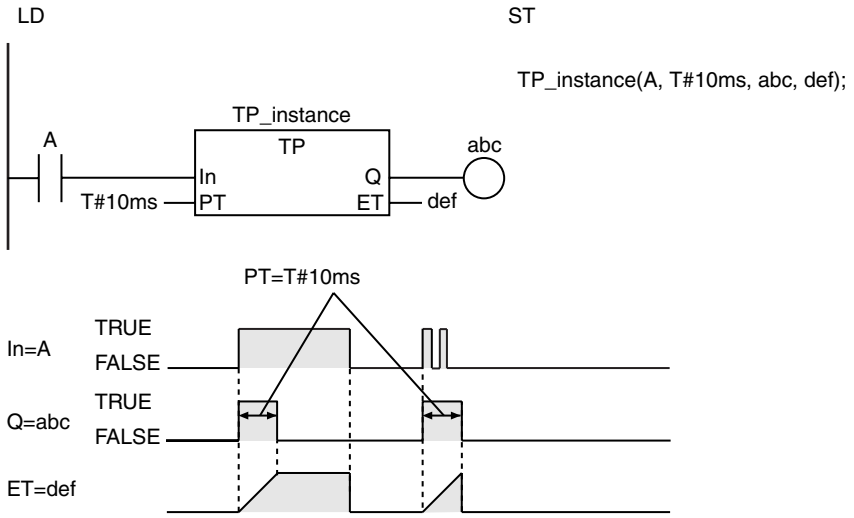
* T#0ms to T#106751d_23h_47m_16s_854.775807ms

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Q	OK																			
ET																OK				

Function

The TP instruction outputs TRUE while the set time elapses after the timer starts. Set the time in nano-seconds (ns). The timing accuracy is 100 ns. The timer starts when timer input *In* changes to TRUE and timer output *Q* changes to TRUE. Elapsed time *ET* is incremented as time elapses. When *ET* reaches set time *PT*, timer output *Q* changes to FALSE. *ET* is not incremented after that. The timer is reset when *In* changes to FALSE. *ET* changes to 0. The timer is not reset even if *In* changes to FALSE after the timer starts but before *ET* reaches *PT*.

The following figure shows a programming example and timing chart for a *PT* of T#10ms. Variable *abc* changes to TRUE as soon as variable *A* changes to TRUE. Variable *abc* changes to FALSE 10 ms later.



Additional Information

- Use the TON instruction (page 2-116) for a timer that starts when *In* changes to TRUE and then changes the timer output to TRUE when the elapsed time reaches the set time.
- Use the TOF instruction (page 2-120) for a timer that starts when *In* changes to FALSE and then changes the timer output to FALSE when the elapsed time reaches the set time.

Precautions for Correct Use

- *ET* and *Q* are updated only when the instruction is executed. Therefore, *Q* does not change to FALSE precisely when the elapsed time from when the timer starts reaches *PT*. *Q* changes the next time the instruction is executed after the elapsed time from when the timer starts reaches *PT*. The change in *Q* can therefore occur with a delay of up to one task period.
- Set *PT* and *ET* in nanoseconds (ns), but remember the timing accuracy is 100 ns.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If T#0ms or a negative number is set for *PT*, *Q* will not change to TRUE even if the value of *In* changes to TRUE.
- You can change the value of *PT* while the value of *In* is TRUE. Operation is as follows:

Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	FALSE	---	The value of *Q* remains FALSE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	TRUE	*PT* ≥ *ET*	Timing is continued. When the value of *ET* reaches the value of *PT*, the value of Q changes to FALSE and *ET* is no longer incremented.
		PT < *ET*	The value of *Q* changes to FALSE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, timing is continued to the end if the timer is operating. Then, the value of *ET* changes to 0 and the value of *Q* changes to FALSE. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even if the value of *Q* is TRUE.

- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated and timing is not performed. Timing restarts when the instruction is executed again.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

AccumulationTimer

The AccumulationTimer instruction totals the time that the timer input is TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AccumulationTimer	Accumulation Timer	FB	AccumulationTimer_instance AccumulationTimer In PT Reset Q ET	AccumulationTimer_instance(In, PT, Reset, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer operates FALSE: Timer stops	Depends on data type.	---	FALSE
PT	Set time		Maximum time	*	ms	0
Reset	Reset		TRUE: Timer reset FALSE: Timer not reset	Depends on data type.	---	FALSE
Q	Timer output	Output	TRUE: *ET* reached *PT.* FALSE: *ET has not* reached *PT.*	Depends on data type.	---	---
ET	Total time		Total time	*	ms	

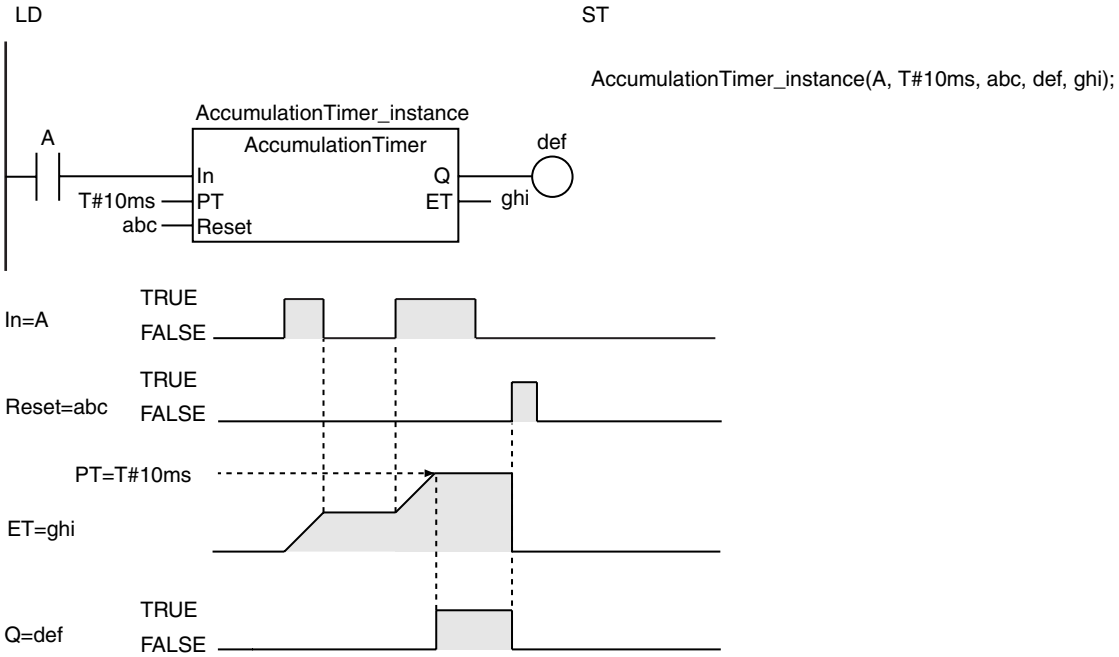
* T#0ms to T#106751d_23h47m_16s_854.775807ms

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Reset	OK																			
Q	OK																			
ET																OK				

Function

The AccumulationTimer instruction totals the time that the timer input is TRUE. Set the time in nanoseconds (ns). The timing accuracy is 100 ns. If reset *Reset* is FALSE, the timer starts when *In* changes to TRUE. Total time *ET* is incremented as time elapses. The timer stops when *In* changes to FALSE. *ET* is held. When *In* changes to TRUE again, the timer starts again. *ET* is incremented from the value that was previously held. When *ET* reaches set time *PT*, timer output *Q* changes to TRUE. *ET* is not incremented after that. The timer is reset when *Reset* changes to TRUE.
ET changes to 0 and *Q* changes to FALSE.

The following figure shows a programming example and timing chart for a *PT* of T#10ms. Variable *abc* changes to TRUE when variable *A* is TRUE for a total of 10 ms (i.e., the total time).



Additional Information

Use the TON instruction (page 2-116) for a timer that resets the timer output and elapsed time when *In* changes to FALSE.

Precautions for Correct Use

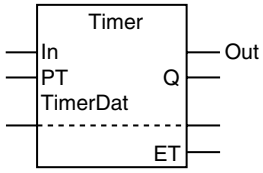
- *ET* and *Q* are updated only when the instruction is executed. Therefore, *Q* does not change to TRUE precisely when the total time of timer operation reaches *PT*. *Q* changes the next time the instruction is executed after the total time of timer operation reaches *PT*. The change in *Q* can therefore occur with a delay of up to one task period.
- Set *PT* and *ET* in nanoseconds (ns), but remember the timing accuracy is 100 ns.
- If *In* and *Reset* are both TRUE, *Reset* has priority. That is, *ET* changes to 0 and *Q* changes to FALSE.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If T#0ms or a negative number is set for *PT*, *Q* will change to TRUE as soon as the value of *In* changes to TRUE.
- You can change the value of *PT* before the value of *ET* reaches the value of *PT*. Operation is as follows:

Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	TRUE	---	The value of *Q* remains TRUE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	FALSE	$PT \geq ET$	When the value of *In* changes to TRUE, timing is continued. When the value of *ET* reaches the value of *PT*, the value of *Q* changes to TRUE and *ET* is no longer incremented.
		$PT < ET$	When the value of *In* changes to TRUE, the value of *Q* changes to TRUE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - The timer stops. The values of *ET* and *Q* at that time are retained.
  - When the master control reset is cleared, *ET* is incremented again from the value that was retained.
  - If an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even if the value of *Q* is TRUE.
  - *Reset* is enabled.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

Timer

The Timer instruction outputs TRUE when the set time elapses after the timer starts. Set the time in increments of 100 ms. The timing accuracy is 100 ms.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Timer	Hundred-ms Timer	FUN		Out:=Timer (In, PT, TimerDat, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer start specification FALSE: Timer reset specification	Depends on data type.	---	FALSE
PT	Set time		Time from when timer starts until *Q* changes to TRUE		ms	*
TimerDat	Timer status	In-out	Current status of timer	---	---	---
Out	Return value	Output	TRUE: Make timer output TRUE FALSE: Make timer output FALSE	Depends on data type.	---	---
Q	Timer output		Same meaning as *Out*.			
ET	Remaining time		Remaining time		ms	

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT							OK													
TimerDat	Structure _sTimer																			
Out	OK																			
Q	OK																			
ET							OK													

Function

The Timer instruction outputs TRUE when the set time elapses after the timer starts. Set the time in increments of 100 ms. The timing unit is 100 ms.

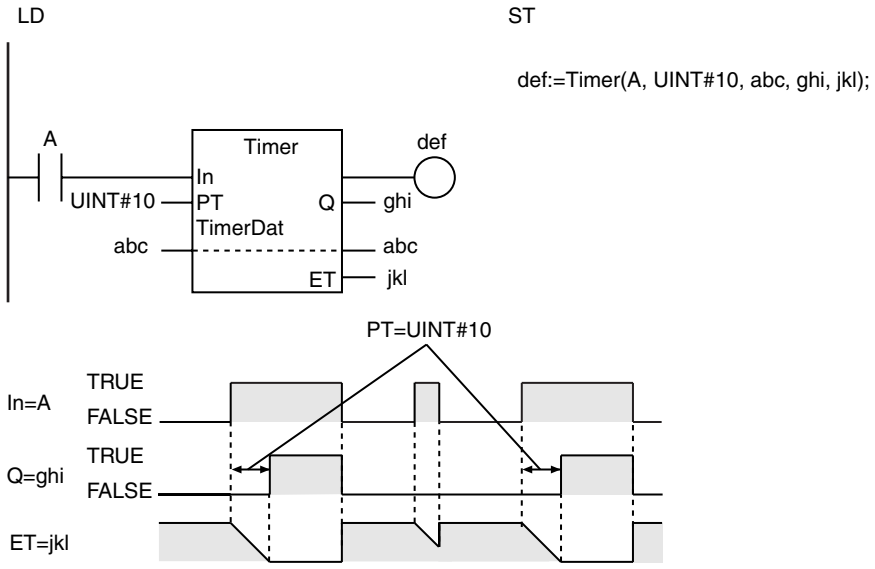
The timer is reset when timer input *In* changes to FALSE. Remaining time *ET* is set to set time *PT*, and timer output *Q* changes to FALSE.

The timer starts when *In* changes to TRUE. The value of *ET* is timed down. When the value of *ET* reaches 0, timer output *Q* changes to TRUE. *ET* is not timed down after that.

The timer is reset if *In* changes to FALSE after the timer starts but before *ET* reaches 0.

The data type of timer status *TimerDat* is structure _sTimer.

The following figure shows a programming example and timing chart when *PT* is UINT#10. Variable *ghi* will change to TRUE 1,000 ms (1 s) after variable *A* changes to TRUE.



Additional Information

For more precise timing, use the TON instruction (page 2-116), which is set in increments of 100 nanoseconds (ns). The TON instruction times in increments of 100 nanoseconds (ns) when the instruction is executed, so it is more precise than the Timer instruction. However, the execution time of the Timer instruction is shorter.

Precautions for Correct Use

- Timing is performed at the beginning of the POU that contains this instruction. Therefore, the value of *ET* will be the same regardless of where the instruction is executed in the POU.
- *Q* is updated when the instruction is executed. Therefore, *Q* does not change to TRUE precisely when the time that is set with *PT* elapses after the timer starts. *Q* changes to TRUE the next time the instruction is executed after the time that is set with *PT* elapses after the timer starts. The change in *Q* can therefore occur with a delay of up to one task period.
- Although *TimerDat* is an in-out variable, it is not necessary to pass any values. Create a memory area for the size of the _sTimer structure and pass it to the instruction.
- Do not change the contents of *TimerDat*.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If the value of *PT* changes, the new value is used from the next time that the timer is reset. The value is not updated while timing is in progress.
- If this instruction is in a master control region and the master control region is reset, the timer is reset. *ET* is set to the value of *PT* and the value of *Q* changes to FALSE.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.

- If this instruction is used in a ladder diagram, the values of *Q* and *Out* change to FALSE if an error occurs in the previous instruction on the rung.

Counter Instructions

Instruction	Name	Page
CTD	Down-counter	2-134
CTD_**	Down-counter Group	2-136
CTU	Up-counter	2-138
CTU_**	Up-counter Group	2-140
CTUD	Up-down Counter	2-142
CTUD_**	Up-down Counter Group	2-146

CTD

The CTD instruction decrements the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTD	Down-counter	FB	CTD_instance CTD CD Q Load CV PV	CTD_instance (CD, Load, PV, Q, CV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CD	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Load	Load signal		TRUE: Set CV to PV.			
PV	Preset value		Counter preset value	0 to 32767		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	0 to 32767		

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	OK																			
Load	OK																			
PV											OK									
Q	OK																			
CV											OK									

Function

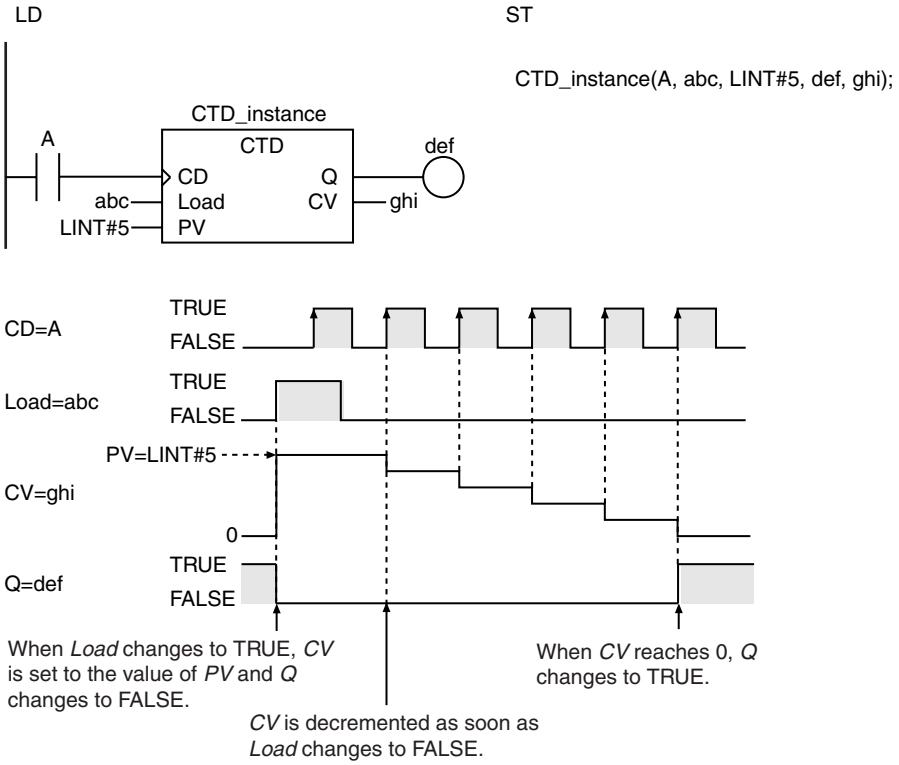
The CTD instruction creates a down counter. The preset value and counter value must have an INT data type.

When load signal *Load* changes to TRUE, counter value CV is set to the value of preset value *PV* and counter output *Q* changes to FALSE. When counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *Q* changes to TRUE.

After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

CD is ignored while *Load* is TRUE. *CV* is not decremented.

The following figure shows a programming example and timing chart for a *PV* of LINT#5.



Additional Information

- Use the CTU instruction (page 2-138) to create a counter that increments the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-142) to create a counter that is both incremented and decremented.

Precautions for Correct Use

- Change *Load* to TRUE and then back to FALSE to restart a counter that has completed counting down.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of *Q* changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes.
- If the value of *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is decremented once if the value of *CD* is TRUE when instruction execution is restarted.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTD_**

The CTD_** instruction decrements the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTD_**	Down-counter Group	FB	CTD_**_instance CTD_** CD Q Load CV PV "**" must be DINT, LINT, UDINT, or ULINT.	CTD_**_instance (CD, Load, PV, Q, CV); "**" must be DINT, LINT, UDINT, or ULINT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CD	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Load	Load signal		TRUE: Set CV to PV.			
PV	Preset value		Counter preset value	Depends on data type.*		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	Depends on data type.*		

* Negative numbers are excluded.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	OK																			
Load	OK																			
PV								OK	OK			OK	OK							
Q	OK																			
CV	Must be the same data type as *PV*																			

Function

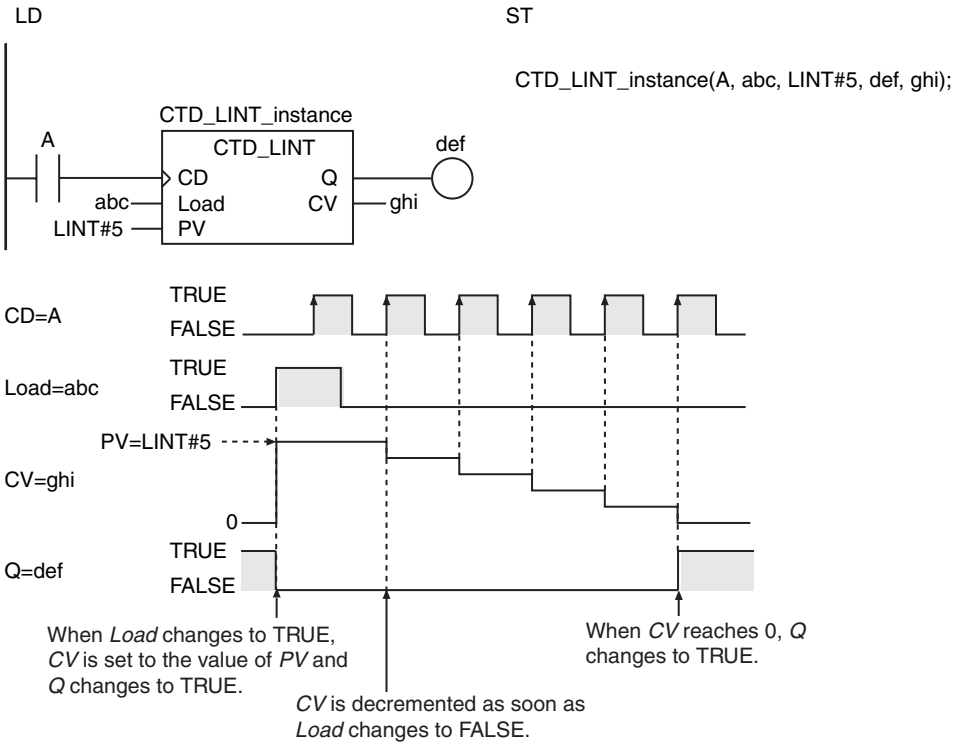
A CTD_** instruction creates a down counter. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT. The name of the instruction is determined by the data type of *PV* and *CV*. For example, if they are the CV data type, the instruction is CTD_LINT.

When load signal *Load* changes to TRUE, counter value *CV* is set to the value of preset value *PV* and counter output *Q* changes to FALSE. When counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *Q* changes to TRUE.

After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

CD is ignored while *Load* is TRUE. *CV* is not decremented.

The following figure shows a CTD_LINT programming example and timing chart for a *PV* of LINT#5.



Additional Information

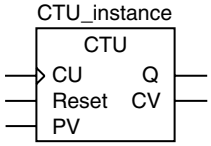
- Use the CTU instruction (page 2-138) to create a counter that increments the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-142) to create a counter that is both incremented and decremented.

Precautions for Correct Use

- Change *Load* to TRUE and then back to FALSE to restart a counter that has completed counting down.
- Use the same data type for *PV* and *CV*.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of Q changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes.
- If the value of *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is decremented once if the value of *CD* is TRUE when instruction execution is restarted.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTU

The CTU instruction increments the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTU	Up-counter	FB		CTU_instance (CU, Reset, PV, Q, CV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Reset	Reset signal		TRUE: Reset *CV* to 0.			
PV	Preset value		Counter preset value	0 to 32767		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	0 to 32767		

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
Reset	OK																			
PV											OK									
Q	OK																			
CV											OK									

Function

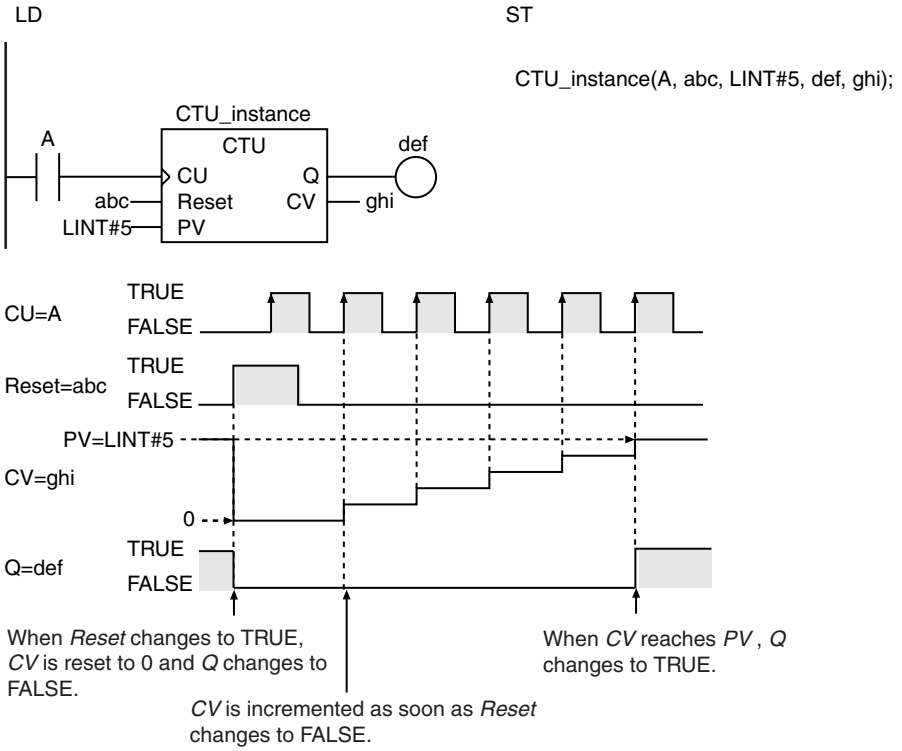
The CTU instruction creates an up counter. The preset value and counter value must have an INT data type.

When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and counter output *Q* changes to FALSE. When counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *Q* changes to TRUE.

After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

CU is ignored while *Reset* is TRUE. *CV* is not incremented.

The following figure shows a programming example and timing chart for a *PV* of LINT#5.



Additional Information

- Use the CTD instruction (page 2-134) to create a counter that decrements the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-142) to create a counter that is both incremented and decremented.

Precautions for Correct Use

- Change *Reset* to TRUE and then back to FALSE to restart a counter that has completed counting up.
- Even when *PV* is set to a negative value, *CV* is set to 0 when the value of *Reset* changes to TRUE. The value of *CV* will be higher than the value of *PV*, so the value of *Q* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- The following operation is performed if the value of *PV* changes while the value of *Reset* is FALSE.

Value of *PV*	Meaning
Larger than the current value of *CV*	The count operation is continued.
Equal to or smaller than the current value of *CV*	The count operation is ended. The value of *Q* changes to TRUE. The current value of *CV* is retained. It does not change.

- If the value of *CU* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented once if the value of *CU* is TRUE when instruction execution is restarted.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTU_**

The CTU_** instruction increments the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTU_**	Up-counter Group	FB	CTU_**_instance CTU_** CU Q Reset CV PV "**" must be DINT, LINT, UDINT, or ULINT.	CTU_**__instance (CU, Reset, PV, Q, CV); "**" must be DINT, LINT, UDINT, or ULINT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Reset	Reset signal		TRUE: Reset *CV* to 0.			
PV	Preset value		Counter preset value	Depends on data type.*		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	Depends on data type.*		

* Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
Reset	OK																			
PV								OK	OK			OK	OK							
Q	OK																			
CV	Must be the same data type as *PV*																			

Function

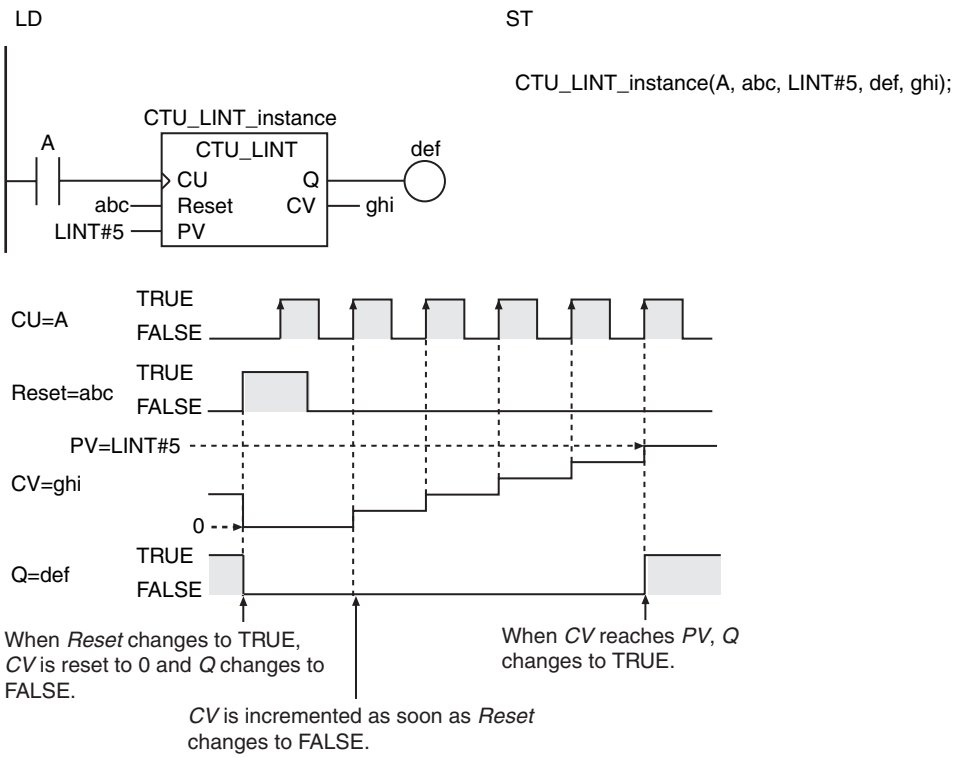
A CTU_** instruction creates an up counter. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT. The name of the instruction is determined by the data type of *PV* and *CV*. For example, if they are the LINT data type, the instruction is CTU_LINT.

When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and counter output *Q* changes to FALSE. When counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *Q* changes to TRUE.

After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

CU is ignored while *Reset* is TRUE. *CV* is not incremented.

The following figure shows a CTU_LINT programming example and timing chart for a *PV* of LINT#5.



Additional Information

- Use the CTD instruction (page 2-134) to create a counter that decrements the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-142) to create a counter that is both incremented and decremented.

Precautions for Correct Use

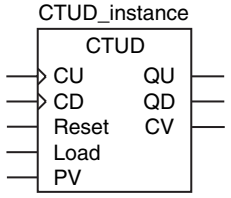
- Change *Reset* to TRUE and then back to FALSE to restart a counter that has completed counting up.
- Even when *PV* is set to a negative value, *CV* is set to 0 when the value of *Reset* changes to TRUE. The value of *CV* will be higher than the value of *PV*, so the value of *Q* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- Use the same data type for *PV* and *CV*.
- The following operation is performed if the value of *PV* changes while the value of *Reset* is FALSE.

Value of *PV*	Meaning
Larger than the current value of *CV*	The count operation is continued.
Equal to or smaller than the current value of *CV*	The count operation is ended. The value of *Q* changes to TRUE. The current value of *CV* is retained. It does not change.

- If the value of *CU* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented once if the value of *CU* is TRUE when instruction execution is restarted.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTUD

The CTUD instruction creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must have an INT data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTUD	Up-down Counter	FB		CTUD_instance (CU, CD, Reset, Load, PV, QU, QD, CV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Up-counter input	Input	Up counter input	Depends on data type.	---	FALSE
CD	Down-counter input		Down counter input			
Reset	Reset signal		TRUE: Reset *CV* to 0.			
Load	Load signal		TRUE: *CV* set to *PV*.			
PV	Preset value		The final counter value when operating as an up counter The initial counter value when operating as a down counter	0 to 32767		0
QU	Up-counter output	Output	TRUE: up-counter output ON FALSE: up-counter output OFF	Depends on data type.	---	---
QD	Down-counter output		TRUE: down-counter output ON FALSE: down-counter output OFF			
CV	Counter value		Counter present value	0 to 32767		

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
CD	OK																			
Reset	OK																			
Load	OK																			

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
PV											OK									
QU	OK																			
QD	OK																			
CV											OK									

Function

The CTUD instruction creates an up-down counter that operates according to an up-counter input signal and a down-counter input signal. It has the functions of both an up counter and a down counter. The preset value and counter value must have an INT data type.

Operation as an Up Counter

When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and up-counter output *QU* changes to FALSE. When up-counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *QU* changes to TRUE. After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

Operation as a Down Counter

When load signal *Load* changes to TRUE, counter value *CV* changes to the value of preset value *PV* and down-counter output *QD* changes to FALSE. When down-counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *QD* changes to TRUE. After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

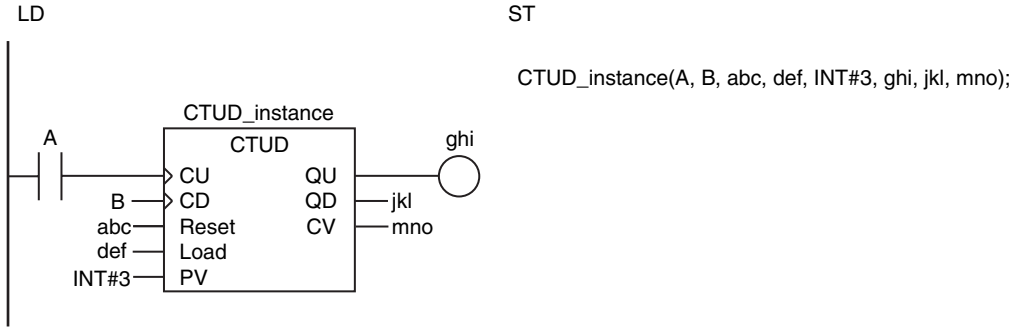
Common Operation for Up and Down Counters

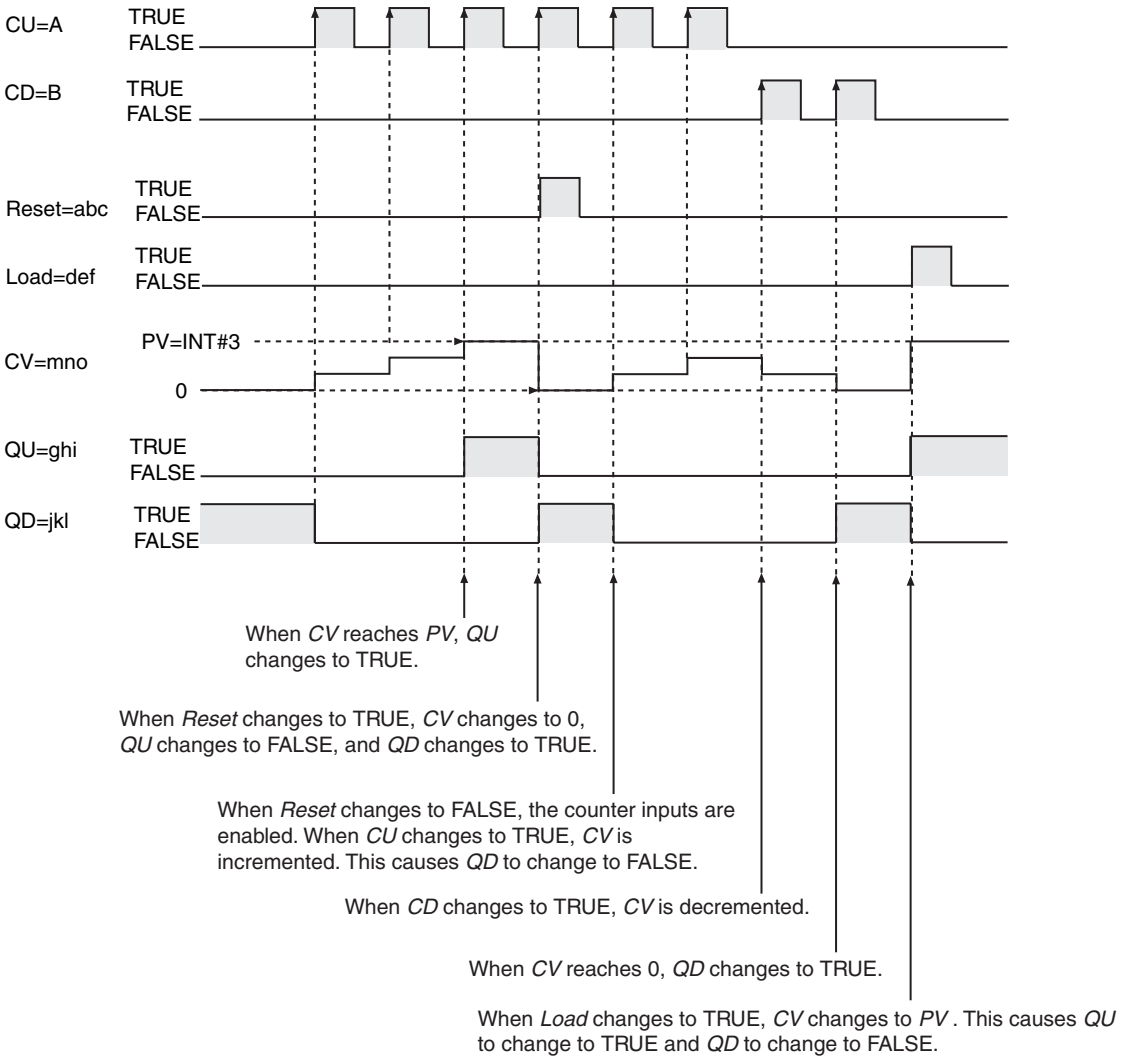
CU and *CD* are ignored while *Load* and *Reset* are TRUE. *CV* is not incremented or decremented. If both *CU* and *CD* change to TRUE at the same time, *CV* will not change. If *Reset* and *Load* are both TRUE, *Reset* has priority and the value of *CV* changes to 0. If *Reset* changes to TRUE, *CV* changes to 0, and so *QD* changes to TRUE. If *Load* changes to TRUE, the value of *CV* changes to *PV*, and so *QU* changes to TRUE.

The following table shows the relationship between *Reset*, *Load*, *CV*, *QU*, and *QD*. This assumes that the value of PV is larger than 0.

Reset	*Load*	*CV*	*QU*	*QD*	Operation
FALSE	FALSE	0 or lower	FALSE	TRUE	Only up counter operation is performed. • *CV* is incremented when *CU* changes to TRUE. It is not decremented when *CD* changes to TRUE.
		Between 0 and *PV*	FALSE	FALSE	Both up and down counter operation is performed. • *CV* is incremented when *CU* changes to TRUE and decremented when *CD* changes to TRUE.
		PV or higher	TRUE	FALSE	Only down counter operation is performed. • *CV* is decremented when *CD* changes to TRUE. It is not incremented when *CU* changes to TRUE.
TRUE	FALSE	0	FALSE	TRUE	The up counter is reset. • The value of *CV* is set to 0.
FALSE	TRUE	*PV*	TRUE	FALSE	The down counter is reset. • The value of *CV* is set to *PV*.
TRUE	TRUE	0	FALSE	TRUE	The up counter is reset. *Reset* take priority over *Load*. • The value of *CV* is set to 0.

The following figure shows a programming example and timing chart for a *PV* of INT#3.





Additional Information

Use the CTD instruction (page 2-134) or CTU instruction (page 2-138) to create a counter that only decrements or only increments.

Precautions for Correct Use

- If you change *Reset* to TRUE to reset the up-counter operation, *QU* will change to FALSE and *QD* will change to TRUE.
- If you change *Load* to TRUE to reset the down-counter operation, *QD* will change to FALSE and *QU* will change to TRUE.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of *QD* changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes. When the value of *Reset* changes to TRUE, the value of *CV* changes to 0. The value of *CV* will be the value of *PV* or higher, so the value of *QU* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- You can change the value of *PV* during execution of the instruction. If the new value of *PV* is less than the current value of *CV*, the value of *QU* changes to TRUE immediately.
- If the value of *CU* or *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented or decremented once if the value of *CU* or *CD* is TRUE when instruction execution is restarted.

CTUD_**

The CTUD_** instruction creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTUD_**	Up-down Counter Group	FB	CTUD_**_instance CTUD_** CU QU CD QD Reset CV Load PV "**" must be DINT, LINT, UDINT, or ULINT.	CTUD_**_instance (CU, CD, Reset, Load, PV, QU, QD, CV); "**" must be DINT, LINT, UDINT, or ULINT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Up-counter input	Input	Up counter input	Depends on data type.	---	FALSE
CD	Down-counter input		Down counter input			
Reset	Reset signal		TRUE: Reset *CV* to 0.			
Load	Load signal		TRUE: *CV* set to *PV*.			
PV	Preset value		The final counter value when operating as an up counter The initial counter value when operating as a down counter	Depends on data type.*		0
QU	Up-counter output	Output	TRUE: up-counter output ON FALSE: up-counter output OFF	Depends on data type.	---	---
QD	Down-counter output		TRUE: down-counter output ON FALSE: down-counter output OFF			
CV	Counter value		Counter present value	Depends on data type.*		

* Negative numbers are excluded.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
CD	OK																			
Reset	OK																			
Load	OK																			
PV								OK	OK			OK	OK							
QU	OK																			
QD	OK																			
CV	Must be the same data type as *PV*																			

Function

A CTUD_** instruction creates an up-down counter that operates according to an up-counter input signal and a down-counter input signal. The counter has the functions of both an up counter and a down counter. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT. The name of the instruction is determined by the data type of *PV* and *CV*. For example, if they are the LINT data type, the instruction is CTUD_LINT.

Operation as an Up Counter

When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and up-counter output *QU* changes to FALSE. When up-counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *QU* changes to TRUE. After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

Operation as a Down Counter

When load signal *Load* changes to TRUE, counter value *CV* changes to the value of preset value *PV* and down-counter output *QD* changes to FALSE. When down-counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *QD* changes to TRUE. After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

Common Operation for Up and Down Counters

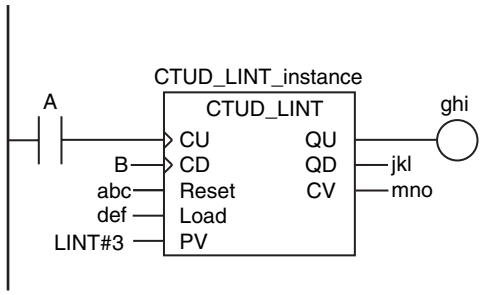
CU and *CD* are ignored while *Load* or *Reset* is TRUE. *CV* is not incremented or decremented. If both *CU* and *CD* change to TRUE at the same time, *CV* will not change. If *Reset* and *Load* are both TRUE, *Reset* has priority and the value of *CV* changes to 0. If *Reset* changes to TRUE, *CV* changes to 0, and so *QD* changes to TRUE. If *Load* changes to TRUE, the value of *CV* changes to *PV*, and so *QU* changes to TRUE.

The following table shows the relationship between *Reset, Load, CV, QU,* and *QD*. This assumes that the value of *PV* is larger than 0.

Reset	*Load*	*CV*	*QU*	*QD*	Operation
FALSE	FALSE	0 or lower	FALSE	TRUE	Only up counter operation is performed. • *CV* is incremented when *CU* changes to TRUE. It is not decremented when *CD* changes to TRUE.
		Between 0 and *PV*	FALSE	FALSE	Both up and down counter operation is performed. • *CV* is incremented when *CU* changes to TRUE and decremented when *CD* changes to TRUE.
		PV or higher	TRUE	FALSE	Only down counter operation is performed. • *CV* is decremented when *CD* changes to TRUE. It is not incremented when *CU* changes to TRUE.
TRUE	FALSE	0	FALSE	TRUE	The up counter is reset. • The value of *CV* is set to 0.
FALSE	TRUE	*PV*	TRUE	FALSE	The down counter is reset. • The value of *CV* is set to *PV*.
TRUE	TRUE	0	FALSE	TRUE	The up counter is reset. *Reset* take priority over *Load*. • The value of *CV* is set to 0.

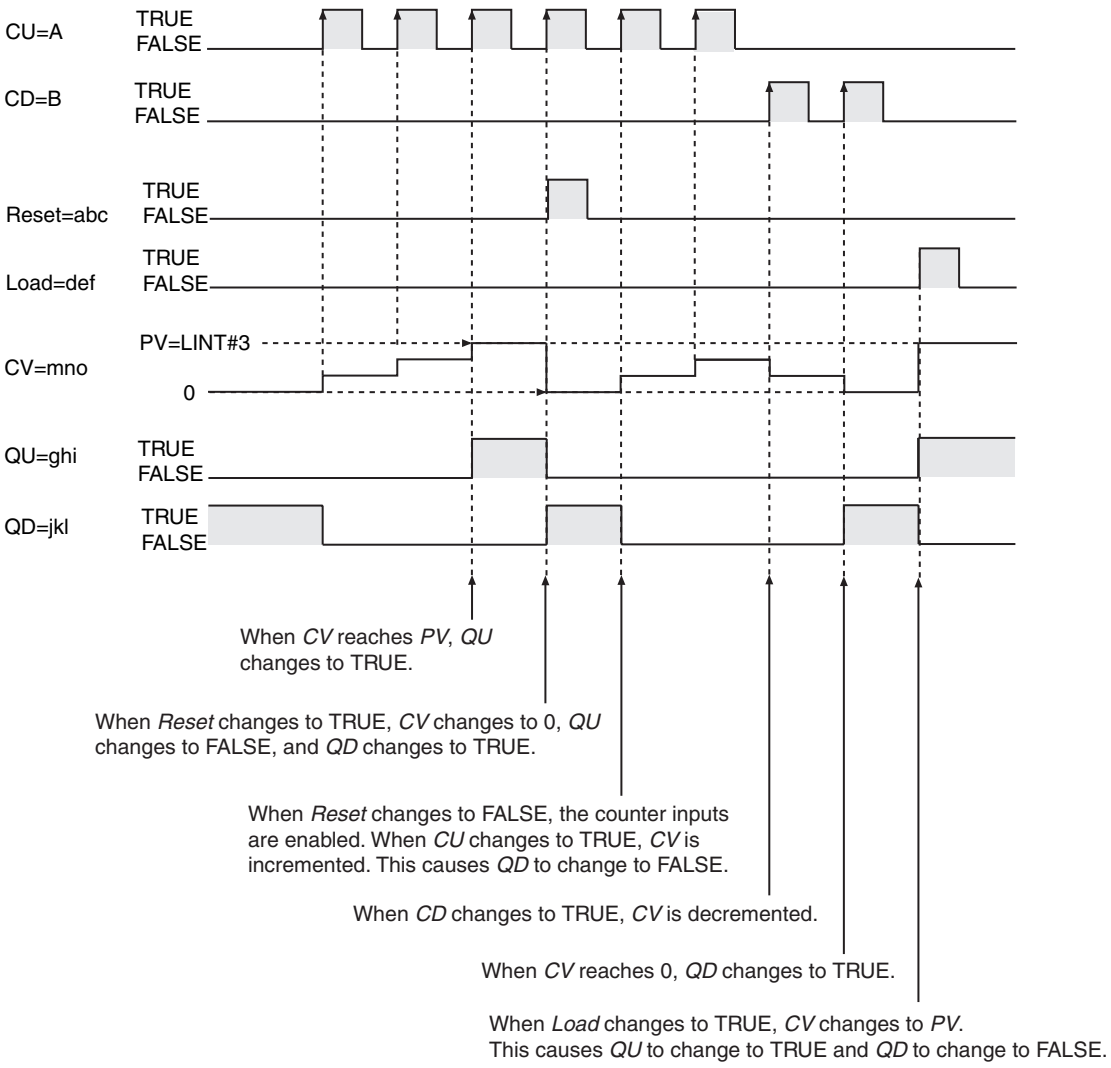
The following figure shows a CTUD_LINT programming example and timing chart for a *PV* of LINT#3.

LD



ST

```
CTUD_LINT_instance(A, B, abc, def, LINT#3, ghi, jkl, mno);
```



Additional Information

Use the CTD instruction (page 2-134) or CTU instruction (page 2-138) to create a counter that only decrements or only increments.

Precautions for Correct Use

- If you change *Reset* to TRUE to reset the up-counter operation, *QU* will change to FALSE and *QD* will change to TRUE.
- If you change *Load* to TRUE to reset the down-counter operation, *QD* will change to FALSE and *QU* will change to TRUE.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of *QD* changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes. When the value of Reset changes to TRUE, the value of *CV* changes to 0. The value of *CV* will be the value of *PV* or higher, so the value of *QU* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- You can change the value of *PV* during execution of the instruction. If the new value of *PV* is less than the current value of *CV*, the value of *QU* changes to TRUE immediately.
- Use the same data type for *PV* and *CV*.

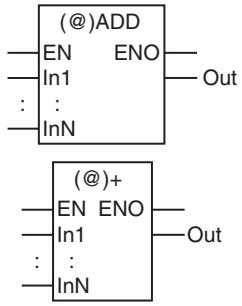
- If the value of *CU* or *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented or decremented once if the value of *CU* or *CD* is TRUE when instruction execution is restarted.

Math Instructions

Instruction	Name	Page	Instruction	Name	Page
ADD (+)	Addition	2-152	EXP	Natural Exponential Operation	2-185
AddOU (+OU)	Addition with Overflow/Underflow Check	2-154	EXPT (**)	Exponentiation	2-187
SUB (-)	Subtraction	2-156	Inc and Dec	Increment/Decrement	2-189
SubOU (-OU)	Subtraction with Overflow/Underflow Check	2-158	Rand	Random Number	2-191
MUL (*)	Multiplication	2-161	AryAdd	Array Addition	2-193
MulOU (*OU)	Multiplication with Overflow/Underflow Check	2-163	AryAddV	Array Value Addition	2-195
DIV (/)	Division	2-166	ArySub	Array Subtraction	2-197
MOD	Modulo-division	2-168	ArySubV	Array Value Subtraction	2-199
ABS	Absolute Value	2-170	AryMean	Array Mean	2-201
RadToDeg and DegToRad	Radians to Degrees/ Degrees to Radians	2-172	ArySD	Array Element Standard Deviation	2-203
SIN, COS, and TAN	Sine in Radians/ Cosine in Radians/ Tangent in Radians	2-174	ModReal	Real Number Modulo-division	2-205
ASIN, ACOS, and ATAN	Principal Arc Sine/ Principal Arc Cosine/ Principal Arc Tangent	2-177	Fraction	Real Number Fraction	2-207
SQRT	Square Root	2-180	CheckReal	Real Number Check	2-209
LN and LOG	Natural Logarithm/ Logarithm Base 10	2-182			

ADD (+)

The ADD (+) instruction adds integers or real numbers. It also joins text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD (+)	Addition	FUN		Out:=In1 + ··· + InN;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Add values	Input	Numbers to add, N = 2 to 5	Depends on data type.	---	0*
Out	Addition result	Output	Addition result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

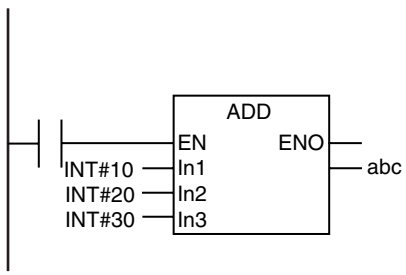
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK

Function

The ADD (+) instruction adds between two and five integers or real numbers. The data types of add values *In1* to *InN* and addition result *Out* can have different data types.

The following example is for when *In1* is INT#10, *In2* is INT#20 and *In3* is INT#30. The value of variable *abc* will be INT#60.

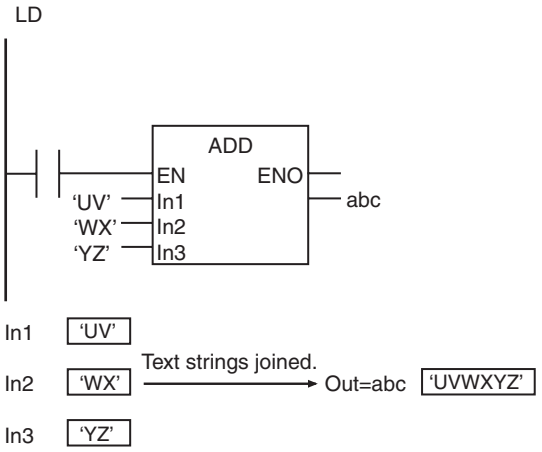
LD



ST

```
abc:=INT#10+INT#20+INT#30;
```

If *In1* to *InN* are STRING data, the text strings are joined. However, if *In1* to *InN* are STRING data, you must use the instruction in a ladder diagram.
The following example is for when *In1* is UV, *In2* is WX and *In3* is YZ. The value of variable *abc* will be UVWXYZ.



The functions of the ADD instruction and the + instruction are exactly the same. Use the form that is easier to use.

Additional Information

- When you calculate real numbers, use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.
- Use the CONCAT instruction (page 2-520) to join text strings in structured text.

Precautions for Correct Use

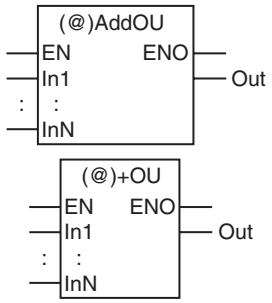
- When you add numbers, set the data type of *Out* to include the valid ranges of *In1* to *InN*.
- If *In1* to *InN* and *Out* are integers, make sure the addition result will fit in the valid range of *Out*. Otherwise, the value of *Out* will be an illegal value. An error will not occur.
- If any of *In1* to *InN* is a real number and the addition result will not fit in the valid range of *Out*, the value of *Out* will be positive or negative infinity.
- When you join text strings, use STRING data for *In1* to *InN* and *Out*.
- The results for overflows in addition are different for ladder diagrams and ST. In a ladder diagram, the calculation is performed within the range of the data type of the input variables. In ST, the precision of the numbers is increased to perform the calculation.
- Addition results of positive or negative infinity are handled as follows for real number values.

Addition	Addition result
+∞ plus number	+∞
−∞ plus number	−∞
+∞ plus +∞	+∞
−∞ plus −∞	−∞
+∞ plus −∞	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is nonnumeric data.
- You can add real numbers and integers. If you do, *Out* is a real number.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - One of *In1* to *InN* does not end in the NULL character when joining strings.
  - The size of the joined text string exceeds the valid range of *Out* when joining strings.

AddOU (+OU)

The AddOU (+OU) instruction adds integers and real numbers. It also performs an overflow/underflow check.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AddOU (+OU)	Addition with Overflow/Underflow Check	FUN		Out:=AddOU(In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Add values	Input	Numbers to add, N = 2 to 5	Depends on data type.	---	0*
Out	Addition result	Output	Addition result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

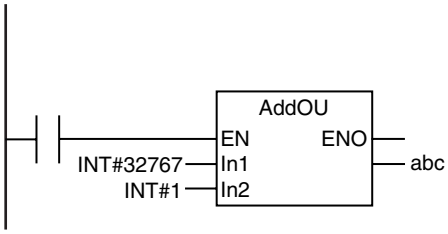
Function

The AddOU (+OU) instruction adds between two and five integers or real numbers and outputs the result. The data types of add values *In1* to *InN* and addition result *Out* can have different data types. If the addition result exceeds the valid range of the data type that includes all of the data types of *In1* to *InN*, the value of the *P_CY* system-defined variable (Carry Flag) changes to TRUE. This indicates that an overflow or an underflow has occurred.

If *Out* is a real number and an overflow or underflow occurs, the value of *Out* is positive or negative infinity. If *Out* is an integer, only the bits of the addition result that fit in the data type of *Out* are assigned to *Out*.

The following example is for when *In1* is INT#32767, *In2* is INT#1 and variable *abc* has an INT data type. The addition result (32768) exceeds the valid range of INT data, so the value of P_CY changes to TRUE. The value of variable *abc* will be INT#−32768 (the lower 16 bits of 32768).

LD



ST

```
abc:=AddOU(INT#32767, INT#1);
```

The functions of the AddOU instruction and the +OU instruction are exactly the same. Use the form that is easier to use.

Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	TRUE: There is an overflow or underflow. FALSE: There is no overflow or underflow.

Additional Information

- When you calculate real numbers, use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.
- Use the ADD (+) instruction (page 2-152) if there is no need for an overflow/underflow check. It will reduce processing time.

Precautions for Correct Use

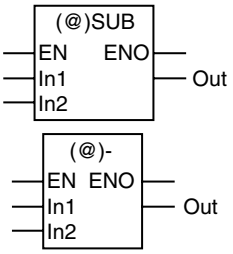
- Set the data type of *Out* to include the valid ranges of *In1* to *InN*.
- If *In1* to *InN* and *Out* are integers, make sure the addition result will fit in the valid range of *Out*. Otherwise, the value of *Out* will be an illegal value. An error will not occur.
- If the data types of *In1* to *InN* are different, calculations and processing of *P_CY* are performed with the data type that includes all of the data types of *In1* to *InN*. For example, if *In1* is INT data and *In2* is DINT data, calculations and *P_CY* processing are performed with DINT data.
- If *In1* to *InN* contains real data, the value of *P_CY* does not change.
- Addition results of positive or negative infinity are handled as follows for real number values.

Addition	Addition result
+∞ plus number	+∞
−∞ plus number	−∞
+∞ plus +∞	+∞
−∞ plus −∞	−∞
+∞ plus −∞	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the value of *Out* is positive infinity, negative infinity, or nonnumeric data, the value of *P_CY* does not change.
- You can add real numbers and integers. If you do, *Out* is a real number.

SUB (-)

The SUB (-) instruction subtracts integers and real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB (-)	Subtraction	FUN		Out:=In1 - In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Minuend	Input	Minuend	Depends on data type.	---	0*
In2	Subtrahend		Subtrahend			
Out	Subtraction result	Output	Subtraction result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

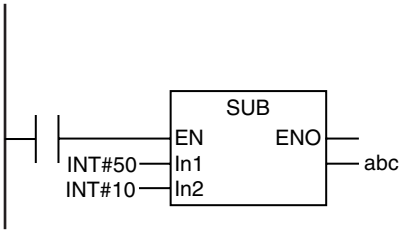
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The SUB (-) instruction subtracts subtrahend *In2* from minuend *In1* and outputs the result. The data types of *In1*, *In2,* and subtraction result *Out* can have different data types.

The following example is for when *In1* is INT#50 and *In2* is INT#10. The value of variable *abc* will be INT#40.

LD



ST

```
abc:=INT#50-INT#10;
```

The functions of the SUB instruction and the - instruction are exactly the same. Use the form that is easier to use.

Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

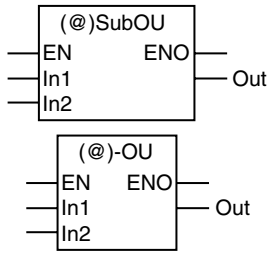
- Set the data type of *Out* to include the valid ranges of *In1* and *In2*.
- If *In1*, *In2*, and *Out* are integers, make sure the subtraction result will fit in the valid range of *Out*. Otherwise, the value of *Out* will be an illegal value. An error will not occur.
- If either *In1* or *In2* is a real number and the addition result will not fit in the valid range of *Out*, the value of *Out* will be positive or negative infinity.
- The results for underflows in subtraction are different for ladder diagrams and ST. In a ladder diagram, the calculation is performed within the range of the data type of the input variables. In ST, the precision of the numbers is increased to perform the calculation.
- Subtraction results of positive or negative infinity are handled as follows for real number values.

Subtraction	Subtraction result
+∞ minus number	+∞
Number minus +∞	−∞
−∞ minus number	−∞
Number minus −∞	+∞
+∞ minus +∞	Nonnumeric data
+∞ minus −∞	+∞
−∞ minus +∞	−∞
−∞ minus −∞	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is nonnumeric data.
- You can subtract a real number from an integer or an integer from a real number. If you do, *Out* is a real number.

SubOU (-OU)

The SubOU (-OU) instruction subtracts integers or real numbers. It also performs an overflow/underflow check.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SubOU (-OU)	Subtraction with Overflow/Underflow Check	FUN		Out:=SubOU(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Minuend	Input	Minuend	Depends on data type.	---	0*
In2	Subtrahend		Subtrahend			
Out	Subtraction result	Output	Subtraction result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

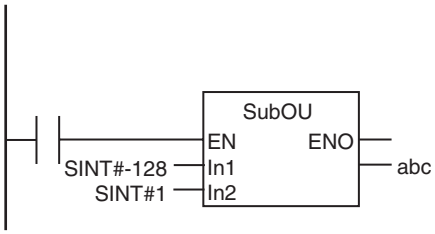
Function

The SubOU (-OU) instruction subtracts subtrahend *In2* from minuend *In1* and outputs the result. The data types of *In1*, *In2,* and subtraction result *Out* can have different data types. If the subtraction result exceeds the valid range of the data type that includes the data types of *In1* and *In2*, the value of the *P_CY* system-defined variable (Carry Flag) changes to TRUE. This indicates that an overflow or an underflow has occurred.

If *Out* is a real number and an overflow or underflow occurs, the value of *Out* is positive or negative infinity. If *Out* is an integer, only the bits of the subtraction result that fit in the data type of *Out* are assigned to *Out*.

The following example is for when *In1* is SINT#-128, *In2* is SINT#1 and variable *abc* has an SINT data type. The subtraction result (−129) exceeds the valid range of SINT data, so the value of P_CY changes to TRUE. The value of variable *abc* will be SINT#127 (the lower 8 bits of −129).

LD



ST

```
abc:=SubOU(SINT#-128, SINT#1);
```

The functions of the SubOU instruction and the -OU instruction are exactly the same. Use the form that is easier to use.

Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	TRUE: There is an overflow or underflow. FALSE: There is no overflow or underflow.

Additional Information

- When you calculate real numbers, use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.
- Use the SUB (–) instruction (page 2-156) if there is no need for an overflow/underflow check. It will reduce processing time.

Precautions for Correct Use

- Set the data type of *Out* to include the valid ranges of *In1* and *In2*.
- If *In1*, *In2*, and *Out* are integers, make sure the subtraction result will fit in the valid range of *Out*. Otherwise, the value of *Out* will be an illegal value. An error will not occur.
- If the data types of *In1* and *In2* are different, calculations and processing of *P_CY* are performed with the data type that includes the data types of *In1* and *In2*. For example, if *In1* is INT data and *In2* is DINT data, calculations and *P_CY* processing are performed with DINT data.
- If *In1* or *In2* contains real data, the value of *P_CY* does not change.
- Subtraction results of positive or negative infinity are handled as follows for real number values.

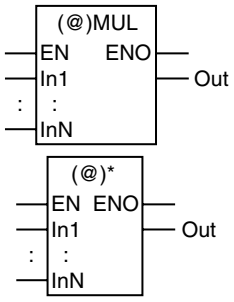
Subtraction	Subtraction result
+∞ minus number	+∞
Number minus +∞	−∞
−∞ minus number	−∞
Number minus −∞	+∞
+∞ minus +∞	Nonnumeric data
+∞ minus −∞	+∞
−∞ minus +∞	−∞
−∞ minus −∞	Nonnumeric data

- If the value of either *In1* or *In2* is nonnumeric data, the value of *Out* is nonnumeric data.

- If the value of *Out* is positive infinity, negative infinity, or nonnumeric data, the value of *P_CY* does not change.
- You can subtract a real number from an integer or an integer from a real number. If you do, *Out* is a real number.

MUL (*)

The MUL (*) instruction multiplies integers and real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MUL (*)	Multiplication	FUN		Out:=In1 * ··· * InN;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Values to multiply	Input	Numbers to multiply, N = 2 to 5	Depends on data type.	---	1*
Out	Multiplication result	Output	Multiplication result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

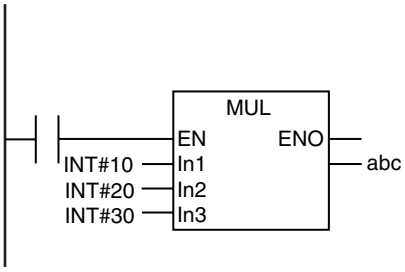
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The MUL (*) instruction multiplies between two and five integers and real numbers and outputs the result. The data types of values to multiply *In1* to *InN* and multiplication result *Out* can have different data types.

The following example is for when *In1* is INT#10, *In2* is INT#20 and *In3* is INT#30. The value of variable *abc* will be INT#6000.

LD



ST

```
abc:=INT#10*INT#20*INT#30;
```

The functions of the MUL instruction and the * instruction are exactly the same. Use the form that is easier to use.

Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

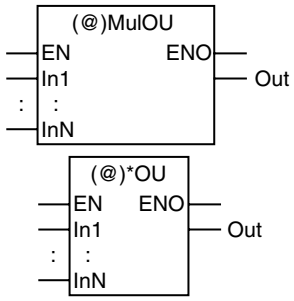
- Set the data type of *Out* to include the valid ranges of *In1* to *InN*.
- If *In1* to *InN* and *Out* are integers, make sure the multiplication result will fit in the valid range of *Out*. Otherwise, the value of *Out* will be an illegal value. An error will not occur.
- If any of *In1* to *InN* is a real number and the multiplication result will not fit in the valid range of *Out*, the value of *Out* will be positive or negative infinity.
- The results for overflows in multiplication are different for ladder diagrams and ST. In a ladder diagram, the calculation is performed within the range of the data type of the input variables. In ST, the precision of the numbers is increased to perform the calculation.
- Multiplication results of positive or negative infinity are handled as follows for real number values.

Multiplication	Multiplication result
+∞ times positive number	+∞
+∞ times negative number	−∞
−∞ times positive number	−∞
−∞ times negative number	+∞
+∞ times +∞	+∞
−∞ times −∞	+∞
+∞ times −∞	−∞
+∞ times 0	Nonnumeric data
−∞ times 0	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is nonnumeric data.
- You can multiply real numbers and integers. If you do, *Out* is a real number.

MulOU (*OU)

The MulOU (*OU) instruction multiplies integers and real numbers and outputs the result. It also performs an overflow/underflow check.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MulOU (*OU)	Multiplication with Overflow/Underflow Check	FUN		Out:=MulOU(In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Values to multiply	Input	Numbers to multiply, N = 2 to 5	Depends on data type.	---	1*
Out	Multiplication result	Output	Multiplication result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

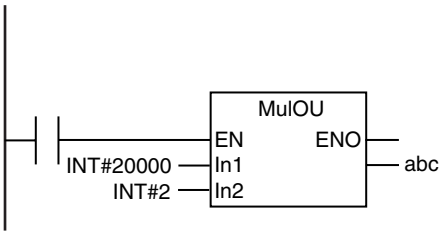
Function

The MulOU (*OU) instruction multiplies between two and five integers and real numbers and outputs the result. The data types of values to multiply *In1* to *InN* and multiplication result *Out* can have different data types. If the multiplication result exceeds the valid range of the data type that includes all of the data types of *In1* to *InN*, the value of the *P_CY* system-defined variable (Carry Flag) changes to TRUE. This indicates that an overflow or an underflow has occurred.

If *Out* is a real number and an overflow or underflow occurs, the value of *Out* is positive or negative infinity. If *Out* is an integer, only the bits of the multiplication result that fit in the data type of *Out* are assigned to *Out*.

The following example is for when *In1* is INT#20000, *In2* is INT#2 and variable *abc* has an INT data type. The multiplication result (40000) exceeds the valid range of INT data, so the value of P_CY changes to TRUE. The value of variable *abc* will be INT#–25536 (the lower 16 bits of 40000).

LD



ST

```
abc:=MulOU(INT#20000, INT#2);
```

The functions of the MulOU instruction and the *OU instruction are exactly the same. Use the form that is easier to use.

Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	TRUE: There is an overflow or underflow. FALSE: There is no overflow or underflow.

Additional Information

Use the MUL (*) instruction (page 2-161) if there is no need for an overflow/underflow check. It will reduce processing time.

Precautions for Correct Use

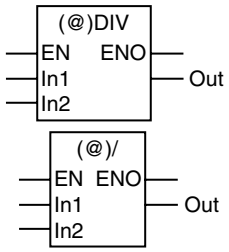
- Set the data type of *Out* to include the valid ranges of *In1* to *InN*.
- If *In1* to *InN* and *Out* are integers, make sure the multiplication result will fit in the valid range of *Out*. Otherwise, the value of *Out* will be an illegal value. An error will not occur.
- If the data types of *In1* to *InN* are different, calculations and processing of *P_CY* are performed with the data type that includes all of the data types of *In1* to *InN*. For example, if *In1* is INT data and *In2* is DINT data, calculations and *P_CY* processing are performed with DINT data.
- If *In1* to *InN* contains real data, the value of *P_CY* does not change.
- Multiplication results of positive or negative infinity are handled as follows for real number values.

Multiplication	Multiplication result
+∞ times positive number	+∞
+∞ times negative number	−∞
−∞ times positive number	−∞
−∞ times negative number	+∞
+∞ times +∞	+∞
−∞ times −∞	+∞
+∞ times −∞	−∞
+∞ times 0	Nonnumeric data
−∞ times 0	Nonnumeric data

- If the value of *Out* is positive infinity, negative infinity, or nonnumeric data, the value of *P_CY* does not change.
- You can multiply real numbers and integers. If you do, *Out* is a real number.

DIV (/)

The DIV (/) instruction divides integers or real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DIV (/)	Division	FUN		Out:=In1/ In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Dividend	Input	Dividend	Depends on data type.	---	*
In2	Divisor		Divisor			
Out	Division result	Output	Division result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

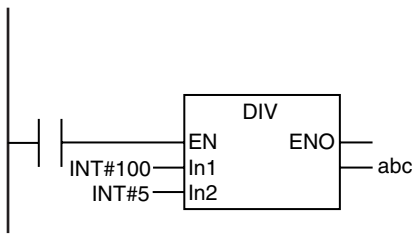
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The DIV (/) instruction divides dividend *In1* by divisor *In2* and outputs the result. The data types of *In1*, *In2,* and division result *Out* can have different data types. If *In1*, *In2*, and *Out* are integers and there is a remainder, the remainder is truncated.

The following example is for when *In1* is INT#100 and *In2* is INT#5. The value of variable *abc* will be INT#20.

LD



ST

```
abc:=INT#100/INT#5;
```

The functions of the DIV instruction and the / instruction are exactly the same. Use the form that is easier to use.

Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

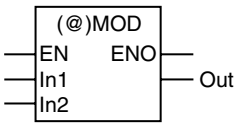
- Set the data type of *Out* to include the valid ranges of *In1* and *In2*.
- If *In1, In2,* and *Out* are integers, make sure the division result will fit in the valid range of *Out*. Otherwise, the value of *Out* will be an illegal value. An error will not occur.
- The results for underflows in division are different for ladder diagrams and ST. In a ladder diagram, the calculation is performed within the range of the data type of the input variables. In ST, the precision of the numbers is increased to perform the calculation.
- Division results of positive infinity, negative infinity, or 0 are handled as follows for real number values.

		In1				
		+∞	**Positive number**	**0**	**Negative number**	−∞
In2	+∞	Nonnumeric data	0	0	0	Nonnumeric data
	Positive number	+∞	Positive number	0	Negative number	−∞
	0	+∞	+∞	Nonnumeric data	−∞	−∞
	Negative number	−∞	Negative number	0	Positive number	+∞
	−∞	Nonnumeric data	0	0	0	Nonnumeric data

- If the value of either *In1* or *In2* is nonnumeric data, the value of *Out* is nonnumeric data.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - *In1*, *In2*, and *Out* are integers and the value of *In2* is 0.

MOD

The MOD instruction finds the remainder for division of integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MOD	Modulo-division	FUN		Out:=In1 MOD In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Dividend	Input	Dividend	Depends on data type.	---	*
In2	Divisor		Divisor			
Out	Remainder	Output	Remainder	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK							
In2						OK	OK	OK	OK	OK	OK	OK	OK							
Out						OK	OK	OK	OK	OK	OK	OK	OK							

Function

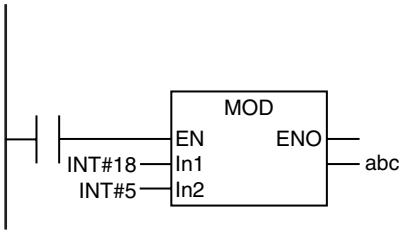
The MOD instruction divides dividend *In1* by divisor *In2* to find the remainder. The data types of *In1*, *In2*, and remainder *Out* can have different data types.

This instruction performs the calculation with the following formula.
Out = In1 − (In1/In2)* In2 (The decimal point is truncated in the division operation.)
Examples of the values of *In1*, *In2*, and *Out* are given in the following table.

Value of *In1*	Value of *In2*	Value of *Out*
5	3	2
5	–3	2
–5	3	–2
–5	–3	–2

The following example is for when *In1* is INT#18 and *In2* is INT#5. The value of variable *abc* will be INT#3.

LD



ST

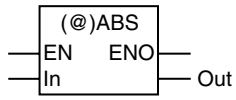
```
abc:=INT#18 MOD INT#5;
```

Precautions for Correct Use

- Set the data type of *Out* to include the valid ranges of *In1* and *In2*.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In2* is 0.

ABS

The ABS instruction finds the absolute value of an integer or real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ABS	Absolute Value	FUN		Out:=ABS(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type.	---	*1
Out	Absolute value	Output	Absolute value	Depends on data type. *2	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 Negative numbers are excluded.

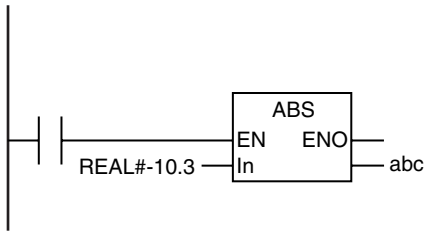
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The ABS instruction outputs the absolute value of the number to process *In*. The data types of *In* and absolute value *Out* can have different data types.

The following example is for when *In* is REAL#−10.3. The value of variable *abc* will be REAL#10.3.

LD



ST

```
abc:=ABS(REAL#-10.3);
```

Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- Set the data type of *Out* to include the absolute value of *In*.
- If the value of *In* is positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
+∞	+∞
−∞	+∞
Nonnumeric data	Nonnumeric data

RadToDeg and DegToRad

RadToDeg: Converts a real number from radians (rad) to degrees (°).
DegToRad: Converts a real number from degrees (°) to radians (rad).

Instruction	Name	FB/FUN	Graphic expression	ST expression
RadToDeg	Radians to Degrees	FUN	(@)RadToDeg EN ENO In Out	Out:=RadToDeg(In);
DegToRad	Degrees to Radians	FUN	(@)DegToRad EN ENO In Out	Out:=DegToRad(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	• RadToDeg: Radians • DegToRad: Degrees	*
Out	Conversion result	Output	Conversion result	Depends on data type.	• RadToDeg: Degrees • DegToRad: Radians	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

● RadToDeg

The RadToDeg instruction converts the data to convert *In* from radians (rad) to degrees (°). The following conversion is used.

$Out=In*180/\pi$

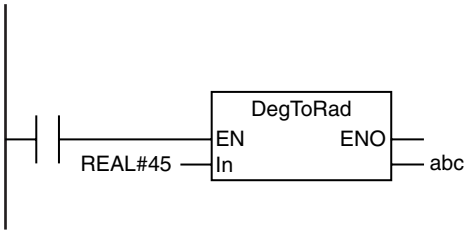
● DegToRad

The DegToRad instruction converts the data to convert *In* from degrees (°) to radians (rad). The following conversion is used.

$Out=In*\pi/180$

The following example for the DegToRad instruction is for when *In* is REAL#45. The value of the REAL variable *abc* will be REAL#0.785398.

LD



ST

```
abc:=DegToRad(REAL#45);
```

Additional Information

Use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- If the absolute value of the conversion result exceeds the maximum value of the data type of *Out*, the value of *Out* will be positive or negative infinity.
- If the absolute value of the conversion result is lower than the minimum value of the data type of *Out*, the value of *Out* will be 0.
- Make sure that the data type of *Out* is equal to or larger than the data type of *In*.
- If the value of *In* is positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
+∞	+∞
−∞	−∞
Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

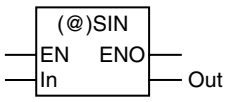
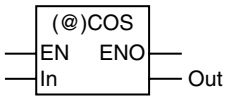
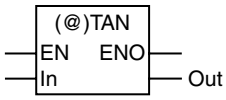
SIN, COS, and TAN

These instructions perform trigonometric calculations on real numbers.

SIN: Finds the sine of a number.

COS: Finds the cosine of a number.

TAN: Finds the tangent of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SIN	Sine in Radians	FUN		Out:=SIN(In);
COS	Cosine in Radians	FUN		Out:=COS(In);
TAN	Tangent in Radians	FUN		Out:=TAN(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type.	Radians	*1
Out	Calculation result	Output	Calculation result	• SIN*2 • COS*2 • TAN Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 The valid range is −1.000000e+0 to 1.000000e+0 for REAL data. The valid range is −1.00000000000000e+0 to 1.00000000000000e+0 for LREAL data.

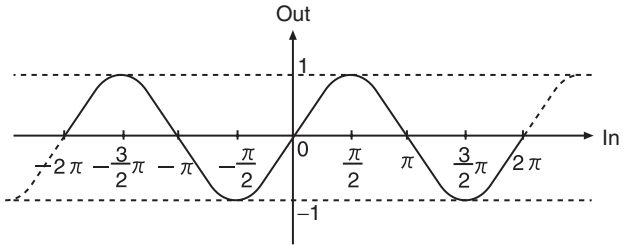
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

These instructions perform trigonometric calculations on real numbers. Number to process *In* is an angle in radians (rad).

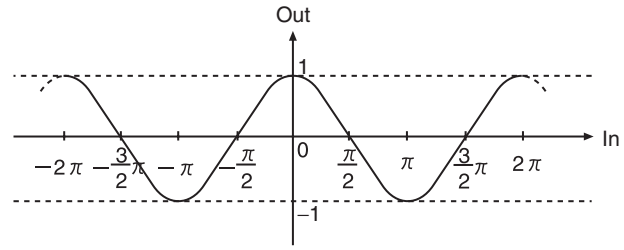
● SIN

The SIN instruction finds the sine of *In.*



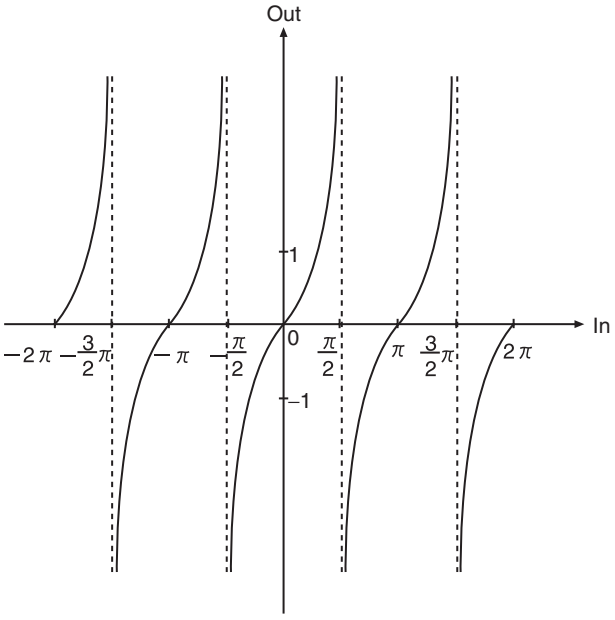
● COS

The COS instruction finds the cosine of *In.*



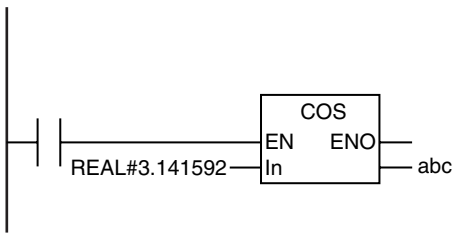
● TAN

The TAN instruction finds the tangent of *In.*



The following example for the COS instruction is for when *In* is REAL#3.141592. The value of variable *abc* will be REAL#−1.0.

LD



ST

```
abc:=COS(REAL#3.141592);
```

Additional Information

- Use the RadToDeg and DegToRad instructions (page 2-172) to convert between degrees and radians.
- If *In* for the TAN instruction is nπ/2, when n is an integer, then the value of *Out* will be positive or negative infinity. Use the CheckReal instruction (page 2-209) to see if the value of *Out* is positive infinity or negative infinity.

Precautions for Correct Use

- If the value of *In* is positive infinity, negative infinity, or nonnumeric data, the value of *Out* is nonnumeric data.
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

ASIN, ACOS, and ATAN

These instructions perform inverse trigonometric calculations on real numbers.

ASIN: Finds the arc sine of a number.

ACOS: Finds the arc cosine of a number.

ATAN: Finds the arc tangent of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ASIN	Principal Arc Sine	FUN	(@)ASIN EN ENO In Out	Out:=ASIN(In);
ACOS	Principal Arc Cosine	FUN	(@)ACOS EN ENO In Out	Out:=ACOS(In);
ATAN	Principal Arc Tangent	FUN	(@)ATAN EN ENO In Out	Out:=ATAN(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type.	---	*
Out	Calculation result	Output	Calculation result	• ASIN $-\pi/2$ to $\pi/2$ • ACOS 0 to π • ATAN $-\pi/2$ to $\pi/2$	rad	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

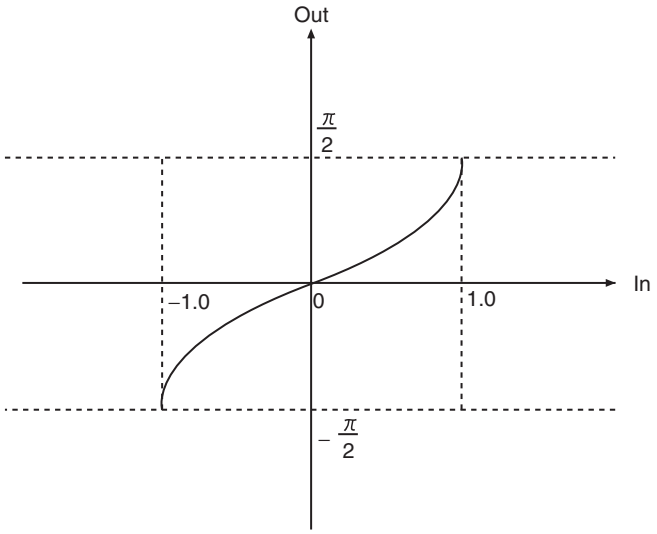
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

These instructions perform inverse trigonometric calculations on real numbers. The calculation result *Out* is an angle in radians (rad).

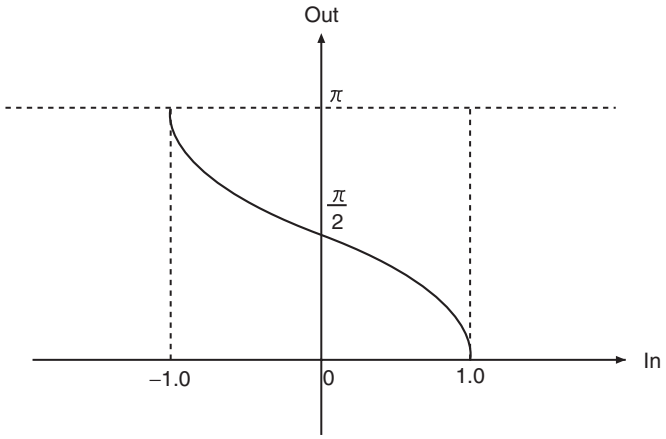
● ASIN

The ASIN instruction finds the arc sine of *In*. *Out* is between $-\pi/2$ and $\pi/2$.



ACOS

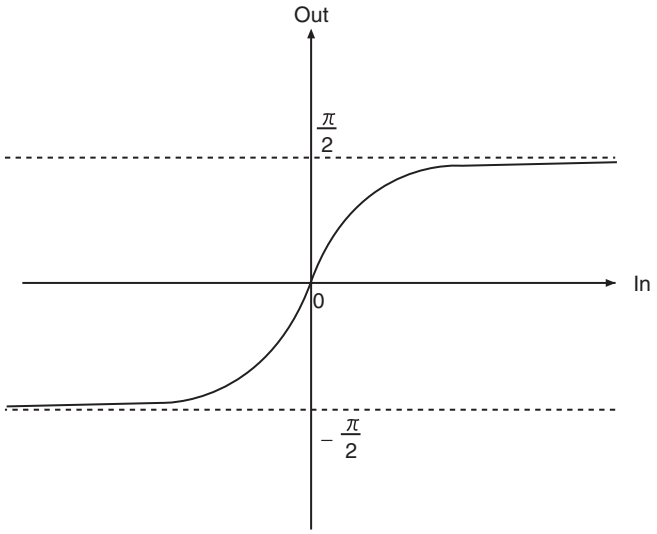
The ACOS instruction finds the arc cosine of *In*. *Out* is between 0 and π.



ATAN

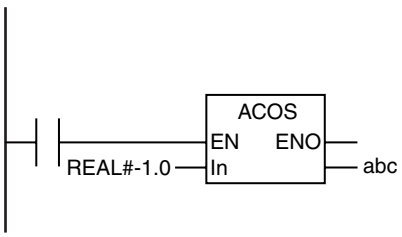
The ATAN instruction finds the arc tangent of *In*. *Out* is between −π/2 and π/2.

If the value of *In* is positive infinity, the value of *Out* is π/2. If the value of *In* is negative infinity, the value of *Out* is −π/2.



The following example for the ACOS instruction is for when *In* is REAL#−1.0. The value of variable *abc* will be REAL#3.141592.

LD



ST

```
abc:=ACOS(REAL#-1.0);
```

Additional Information

Use the RadToDeg and DegToRad instructions (page 2-172) to convert between degrees and radians.

Precautions for Correct Use

- If *In* is not between −1.0 and 1.0 for the ASIN or ACOS instruction, the value of *Out* is nonnumeric data. That also applies when the value of *In* is negative infinity, positive infinity, or nonnumeric data.
- If the value of *In* is nonnumeric data for the ATAN instruction, the value of *Out* is nonnumeric data.
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

SQRT

The SQRT instruction finds the square root of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SQRT	Square Root	FUN	(@)SQRT EN ENO In Out	Out:=SQRT(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type. *1	---	*2
Out	Square root	Output	Square root	*3	---	---

*1 Negative numbers are excluded.

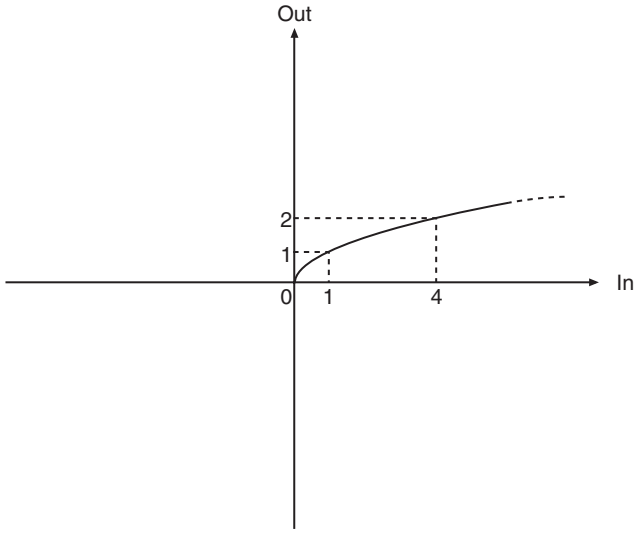
*2 If you omit an input parameter, the default value is not applied. A building error will occur.

*3 The valid range is 0.000000e+00 to 1.844674e+19 or positive infinity for REAL data. The valid range is 0.00000000000000e+000 to 1.34078079299425e+154 or positive infinity for LREAL data.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

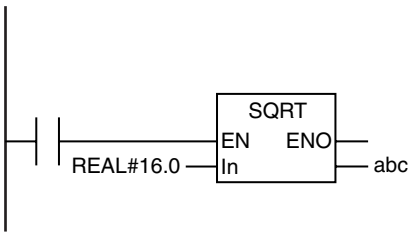
Function

The SQRT instruction finds the square root of number to process *In*. The data types of *In* and square root *Out* can have different data types.



The following example is for when *In* is REAL#16.0. The value of variable *abc* will be REAL#4.0.

LD



ST

```
abc:=SQRT(REAL#16.0);
```

Additional Information

Use the CheckReal instruction (page 2-209) to see if the value of *Out* is positive infinity.

Precautions for Correct Use

- If the value of *In* is not a positive number, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
Negative number	Nonnumeric data
0	0
+∞	+∞
−∞	Nonnumeric data
Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to In, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

LN and LOG

These instructions find the logarithm of a real number.

LN: Finds the natural logarithm of a number.

LOG: Finds the base-10 logarithm of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LN	Natural Logarithm	FUN	(@)LN EN ENO In Out	Out:=LN(In);
LOG	Logarithm Base 10	FUN	(@)LOG EN ENO In Out	Out:=LOG(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type. *1	---	*2
Out	Logarithm	Output	Logarithm	*3	---	---

*1 Negative numbers are excluded.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

*3 LN:
If *In* and *Out* are REAL data: –8.73365448e+1 to 8.87228390e+1, or −∞/+∞
If *In* is REAL and *Out* is LREAL data: –8.7336544750000000e+1 to 8.8722839050000000e+1 or −∞/+∞
If *In* is LREAL and *Out* is REAL data: –7.08384950e+2 to 7.09782712e+2 or −∞/+∞
If *In* and *Out* are LREAL data: –7.0838495021978327e+1 to 7.0978271289338399e+2 or −∞/+∞

LOG:
If *In* and *Out* are REAL data: –3.79297795e+1 to 3.85318394e+1 or −∞/+∞
If *In* is REAL and *Out* is LREAL data: –3.7929779453965430e+1 to 3.8531839419564961e+1 or −∞/+∞
If *In* is LREAL and *Out* is REAL data: –3.07652656e+2 to 3.08254716e+2 or −∞/+∞
If *In* and *Out* are LREAL data: –3.0765265556858878e+2 to 3.0825471555991674e+2 or −∞/+∞

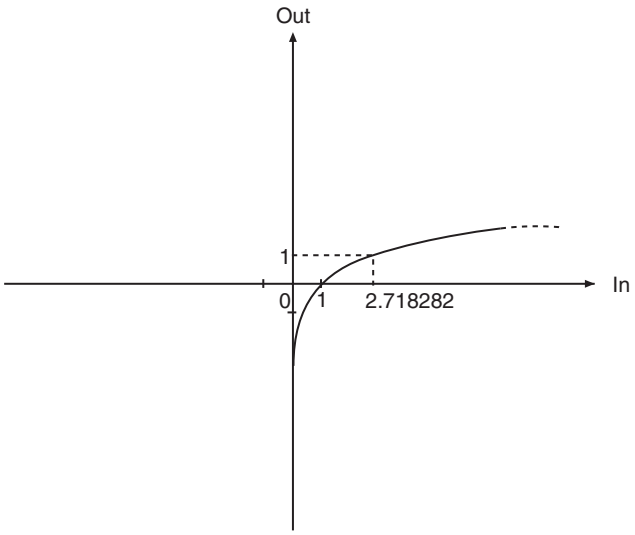
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

These instructions find the logarithm of a real number.

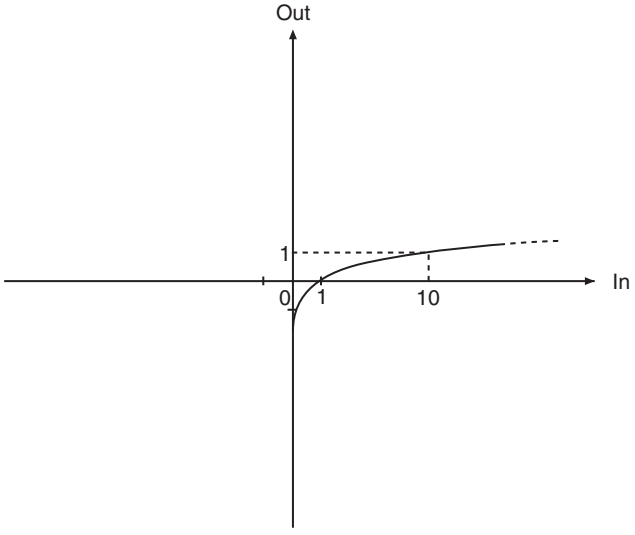
● LN

The LN instruction finds the natural logarithm (logarithm to base e, where e = 2.718282).



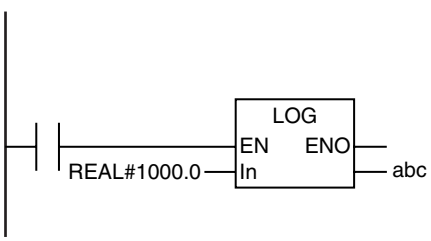
● LOG

The LOG instruction finds the base-10 logarithm.



The following example for the LOG instruction is for when *In* is REAL#1000.0. The value of variable *abc* will be REAL#3.0.

LD



ST

```
abc:=LOG(REAL#1000.0);
```

Additional Information

Use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- If the value of *In* is not a positive number, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
Negative number	Nonnumeric data
0	−∞
+∞	+∞
−∞	Nonnumeric data
Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

EXP

The EXP instruction performs calculations for the natural exponential function.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EXP	Natural Exponential Operation	FUN		Out:=EXP(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Exponent	Input	Exponent	Depends on data type.	---	*1
Out	Calculation result	Output	Calculation result	Depends on data type. *2	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 Negative numbers are excluded.

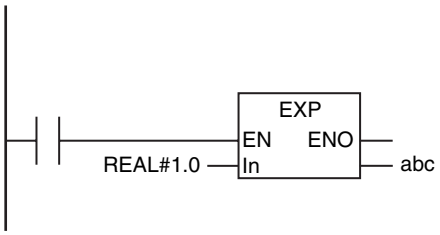
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

The EXP instruction returns the value of e^{In}, where e is Euler's constant and In is an input variable.

The following example is for when *In* is REAL#1.0. The value of variable *abc* will be REAL#2.718282.

LD



ST

```
abc:=EXP(REAL#1.0);
```

Additional Information

- Use the EXPT (**) instruction (page 2-187) to find the powers of numbers with bases other than e.
- Use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or non-numeric data.

Precautions for Correct Use

- If the value of *In* is 0.0, positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

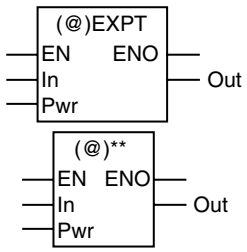
Value of *In*	Value of *Out*
0	1.0
+∞	+∞
−∞	0.0
Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

EXPT (**)

The EXPT (**) instruction raises one real number to the power of another real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EXPT (**)	Exponentiation	FUN		Out:=EXPT(In, Pwr); Out:=In ** Pwr;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Base number	Input	Base number (e.g., 5 for 5^2)	Depends on data type.	---	*
Pwr	Exponent		Exponent (e.g., 2 for 5^2)			
Out	Calculation result	Output	Calculation result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

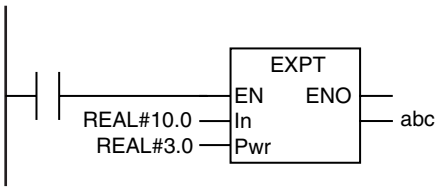
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Pwr														OK	OK					
Out														OK	OK					

Function

The EXPT (**) instruction raises base number *In* to exponent *Pwr* to find $In^{Pwr.}$

The following example is for when *In* is REAL#10.0 and *Pwr* is REAL#3.0. The value of variable *abc* will be REAL#1000.0.

LD



ST

```
abc:=EXPT(REAL#10.0, REAL#3.0);
```

The functions of the EXPT instruction and the ** instruction are exactly the same. Use the form that is easier to use.

Additional Information

- Use the EXP instruction (page 2-185) to find powers of base e.
- Use the CheckReal instruction (page 2-209) to see if *Out* is positive infinity, negative infinity, or non-numeric data.

Precautions for Correct Use

- If the absolute value of the calculation result is lower than the minimum value for a real number, the value of *Out* will be 0.

  Example: $(1.175494e{-}38)^2 \rightarrow 0$

- The following table shows the values of *Out* for different combinations of *In* and *Pwr* values.

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="10">In</th></tr>
<tr><th>+∞</th><th>1 to +∞</th><th>1</th><th>0 to 1</th><th>0</th><th>−1 to 0</th><th>−1</th><th>−1 to −∞</th><th>−∞</th><th>Nonnumeric data</th></tr>
<tr><td rowspan="10">Pwr</td><td>+∞</td><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>+∞</td><td>+∞</td><td>Nonnumeric data</td></tr>
<tr><td>Positive even number</td><td rowspan="3">+∞</td><td colspan="4" rowspan="3">Number *1, *2</td><td rowspan="3">0</td><td colspan="3">Number *1, *2</td><td>+∞</td><td rowspan="3">Nonnumeric data</td></tr>
<tr><td>Positive odd number</td><td colspan="3">Number *2, *3</td><td>−∞</td></tr>
<tr><td>Positive decimal number</td><td colspan="3">Nonnumeric data</td><td>+∞</td></tr>
<tr><td>0</td><td>1</td><td colspan="4">1</td><td>1</td><td colspan="3">1</td><td>1</td><td>1</td></tr>
<tr><td>Negative even number</td><td rowspan="3">0</td><td colspan="4" rowspan="3">Number *1, *2</td><td rowspan="3">+∞</td><td colspan="3">Number *1, *2</td><td rowspan="3">0</td><td rowspan="3">Nonnumeric data</td></tr>
<tr><td>Negative odd number</td><td colspan="3">Number *2, *3</td></tr>
<tr><td>Negative decimal number</td><td colspan="3">Nonnumeric data</td></tr>
<tr><td>−∞</td><td>0</td><td>0</td><td>1</td><td>+∞</td><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>0</td><td>Nonnumeric data</td></tr>
<tr><td>Nonnumeric data</td><td>Nonnumeric data</td><td colspan="4">Nonnumeric data</td><td>Nonnumeric data</td><td colspan="3">Nonnumeric data</td><td>Nonnumeric data</td><td>Nonnumeric data</td></tr>
</table>

*1 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be positive infinity.

*2 If the calculation result is too close to 0 to express with the data type of *Out* or if it is an unnormalized number, the value of *Out* will be 0.

*3 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be negative infinity.

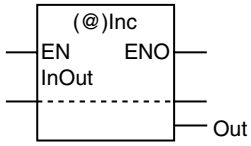
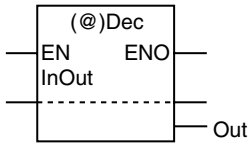
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

Inc and Dec

Inc: Increments an integer value.

Dec: Decrements an integer value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Inc	Increment	FUN		Inc(InOut);
Dec	Decrement	FUN		Dec(InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Target data	In-out	Target data	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut						OK	OK	OK	OK	OK	OK	OK	OK							
Out	OK																			

Function

Inc

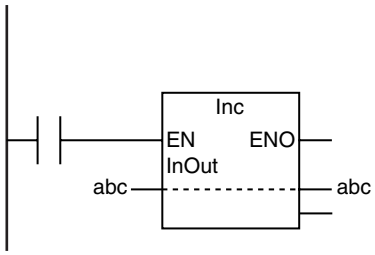
The Inc instruction increments target data *InOut*. If the result exceeds the maximum value of *InOut*, *InOut* returns to the minimum value.

Dec

The Dec instruction decrements target data *InOut*. If the result exceeds the minimum value of *InOut*, *InOut* returns to the maximum value.

The following example for the Inc instruction is for when variable *abc* is passed to *InOut*.

LD



ST

```
Inc(abc);
```

Precautions for Correct Use

Return value *Out* is not used when the instruction is used in ST.

Rand

The Rand instruction generates pseudorandom numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Rand	Random Number	FB	Rand_instance Rand Execute ENO Seed Rnd	Rand_instance(Execute, Seed, Rnd);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Seed	Random number pattern	Input	Random number pattern	Depends on data type.	---	*1
Rnd	Random number	Output	Random number	*2	---	---

*1 If you omit the input parameter, the value will be 0. It will not be the value that is specified for the Initial Value attribute.

*2 0.00000000000000e+0 to 1.00000000000000e+0

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Seed							OK													
Rnd															OK					

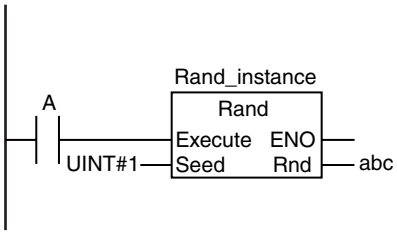
Function

The Rand instruction specifies random number *Rnd*. The value of *Rnd* is different each time the instruction is executed: Random number pattern *Seed* specifies the random number system. If the value of *Seed* is the same, the same random number series is generated each time the power supply is turned ON. This allows you to generate a repeatable series of random numbers.

If the value of *Seed* is 0, random numbers that cannot be repeated are generated. If you do not want to generate the same series of random numbers each time the power supply is turned ON, set the value of *Seed* to 0.

The following programming example is for when *Seed* is UINT#1.

LD



ST

```
Rand_instance(A, UINT#1, abc);
```

Additional Information

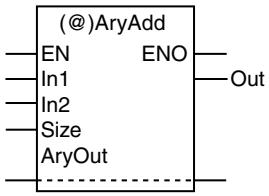
The value of *Rnd* is a real number between 0 and 1. Use the following processing to generate random numbers within a specific range.

Example: The following formula generates random numbers between 100 and 200.

```
Rand_instance(A, UINT#1, abc);
Random number:=LREAL_TO_INT((200.0-100.0)*abc)+100;
```

AryAdd

The AryAdd instruction adds corresponding elements of two arrays.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryAdd	Array Addition	FUN		AryAdd(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array) and In2[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of elements to process			1
AryOut[] (array)	Calculation results array	In-out	Calculation results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

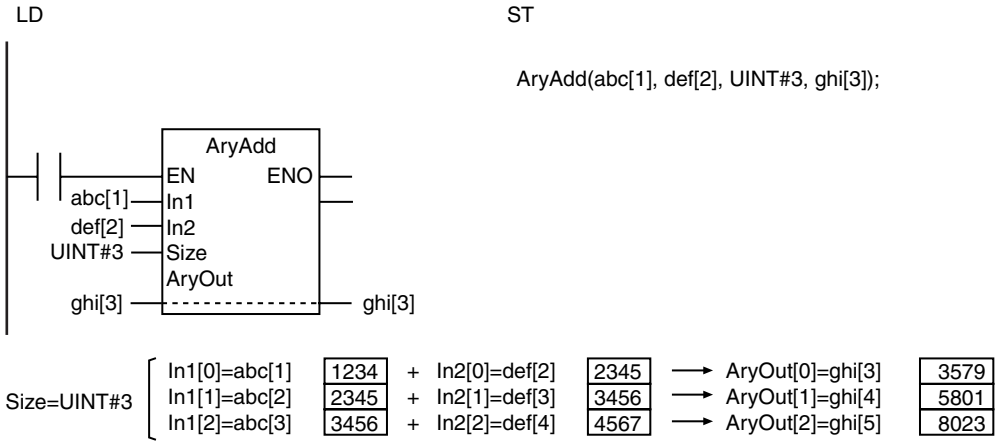
* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2[] (array)	Must be an array with the same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	Must be an array with the same data type as *In1[]*.																			
Out	OK																			

Function

The AryAdd instruction adds *Size* elements of arrays to process *In1[]* and *In2[]* starting from *In1[0]* and *In2[0]*. The results are assigned to corresponding elements of calculation results array *AryOut[]*.

The following example is for when *Size* is UINT#3.

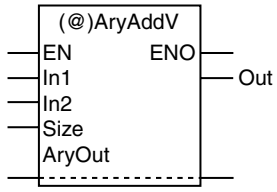


Precautions for Correct Use

- Use the same data type for *In1[]*, *In2[]*, and *AryOut[]*.
- If the calculation results exceed the valid range of *AryOut[]*, the results will be illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - *In1[], In2[],* and *AryOut[]* have different data types.
  - The value of *Size* exceeds the array range of *In1[], In2p[],* or *AryOut[].*

AryAddV

The AryAddV instruction adds the same value to specified elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryAddV	Array Value Addition	FUN		AryAddV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Addition array	Input	Addition array	Depends on data type.	---	*
In2	Value to add		Value to add			
Size	Number of elements		Number of elements of *In1[]* for addition			1
AryOut[] (array)	Addition results array	In-out	Addition results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

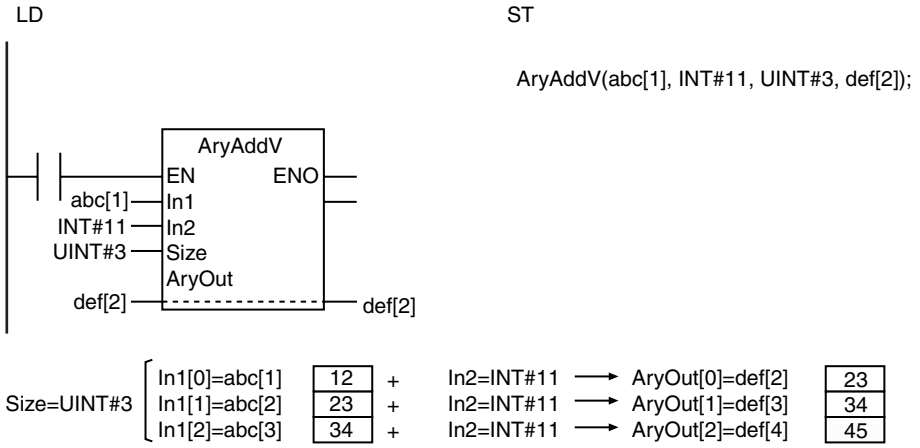
* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2	Must be same data type as *In1[].*																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[].*																			
Out	OK																			

Function

The AryAddV instruction adds value to add *In2* to *Size* elements of addition array *In1[]* starting from *In1[0].* It outputs the results to addition results array *AryOut[]*.

The following example is for when *In2* is INT#11 and *Size* is UINT#3.

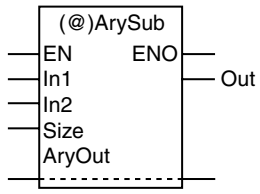


Precautions for Correct Use

- Use the same data type for *In1[]*, *In2*, and *AryOut[]*.
- If the addition results exceed the valid range of *AryOut[]*, the elements of *AryOut[]* will contain illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If *In1[], In2,* and *AryOut[]* have different data types.
  - If the value of *Size* exceeds the array area of *In1[]* or *AryOut[]*.

ArySub

The ArySub instruction subtracts corresponding elements of two arrays.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySub	Array Subtraction	FUN		ArySub(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Minuend array	Input	Minuend array	Depends on data type.	---	*
In2[] (array)	Subtrahend array		Subtrahend array			
Size	Number of elements		Number of elements for subtraction			1
AryOut[] (array)	Subtraction results array	In-out	Subtraction results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

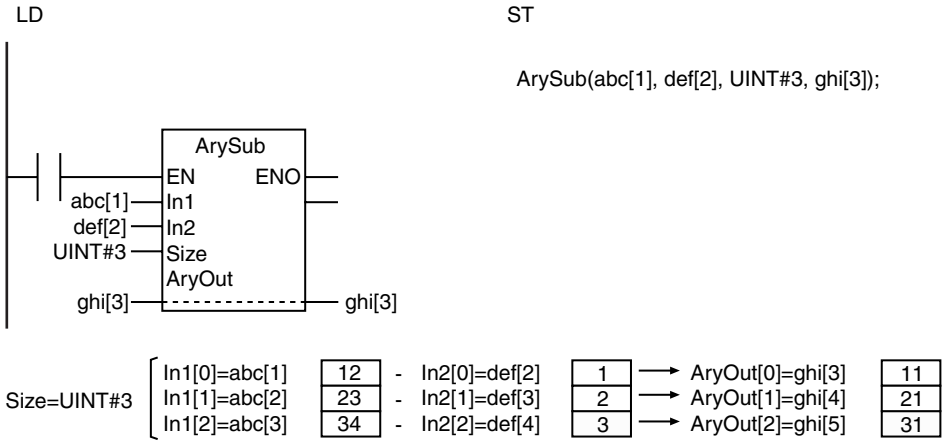
* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2[] (array)	Must be same data type as *In1[].*																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[].*																			
Out	OK																			

Function

The ArySub instruction subtracts Size elements of subtrahend array *In2[]* from corresponding elements of minuend array *In1[]* starting with *In1[0]* and *In2[0]*. It outputs the subtraction results to subtraction results array *AryOut[]*.

The following example is for when *Size* is UINT#3.

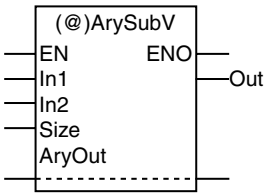


Precautions for Correct Use

- Use the same data type for *In1[]*, *In2[]*, and *AryOut[]*.
- If the subtraction results exceed the valid range of *AryOut[]*, the elements of *AryOut[]* will contain illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - *In1[], In2[],* and *AryOut[]* have different data types.
  - The value of *Size* exceeds the array range of *In1[], In2[],* or *AryOut[].*

ArySubV

The ArySubV instruction subtracts the same value from specified elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySubV	Array Value Subtraction	FUN		ArySubV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Minuend array	Input	Minuend array	Depends on data type.	---	*
In2	Subtrahend		Subtrahend			
Size	Number of elements		Number of elements of *In1[]* for subtraction			1
AryOut[] (array)	Subtraction results array	In-out	Subtraction results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

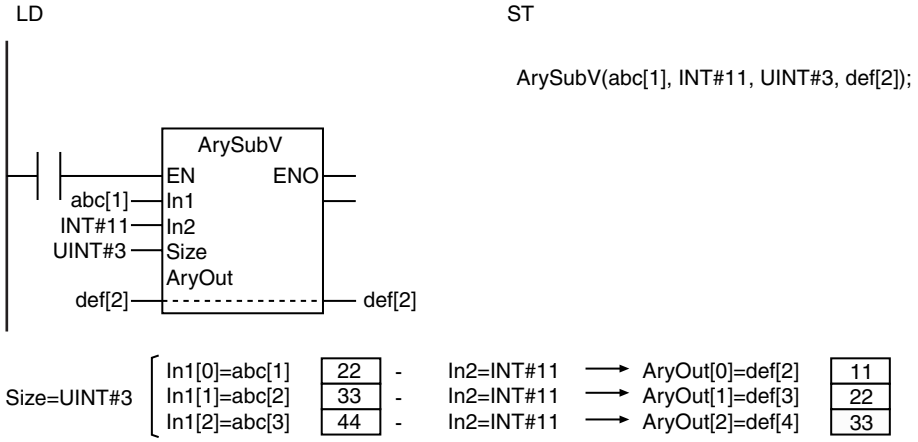
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2	Must be same data type as the elements of *In1[].*																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[].*																			
Out	OK																			

Function

The ArySubV instruction subtracts subtrahend *In2* from *Size* elements of minuend array *In1[]* starting from *In1[0].* It outputs the results to subtraction results array *AryOut[].*

The following example is for when *In2* is INT#11 and *Size* is UINT#3.



Precautions for Correct Use

- Use the same data type for *In1[]*, *In2*, and *AryOut[]*.
- If the subtraction results exceed the valid range of *AryOut[]*, the elements of *AryOut[]* will contain illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - *In1[], In2,* and *AryOut[]* have different data types.
  - The value of *Size* exceeds the array area of *In1[]* or *AryOut[]*.

AryMean

The AryMean instruction calculates the average of the elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryMean	Array Mean	FUN	(@)AryMean EN ENO In Out Size	Out := AryMean(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Out	Calculation result	Output	Calculation result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

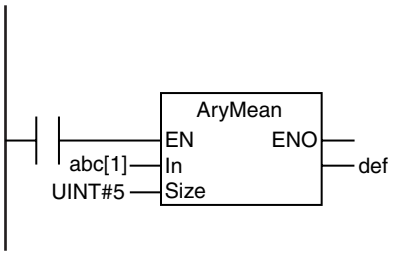
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK				
Size							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK				

Function

The AryMean instruction calculates the average of *Size* elements of array to process *In[]* starting from *In[0]*.

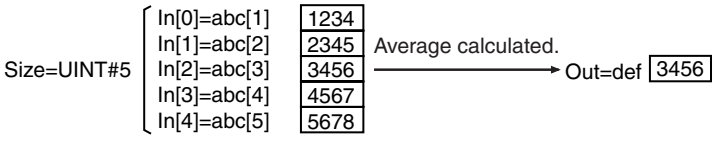
The following example is for when *Size* is UINT#5.

LD



ST

```
def:=AryMean(abc[1], UINT#5);
```

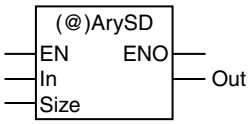


Precautions for Correct Use

- Refer to the descriptions of the functions of the ADD (+) instruction (page 2-152), SUB (–) instruction (page 2-156), MUL (*) instruction (page 2-161), and DIV (/) instruction (page 2-166) for the calculation results when the value of *In[]* is positive infinity, negative infinity, or nonnumeric data.
- If *In[]* or *Out* is an integer, the decimal portion of the average is truncated.
- If you use a different data type for *In[]* and *Out*, make sure the valid range of *Out* includes the valid range of *In[].*
- If the calculation result exceeds the valid range of *Out*, *Out* will contain an illegal value. An error will not occur.
- If an intermediate value in the calculation process exceeds the valid range of *IN[]*, *Out* will contain an illegal value. An error will not occur.
- If the value of *Size* is 0, the value of *Out* is 0.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* exceeds the array area of *In[]*.

ArySD

The ArySD instruction calculates standard deviation of the elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySD	Array Element Standard Deviation	FUN		Out:=ArySD(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements		Number of elements of *In[]* for conversion			2
Out	Standard deviation	Output	Standard deviation	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)														OK	OK					
Size							OK													
Out														OK	OK					

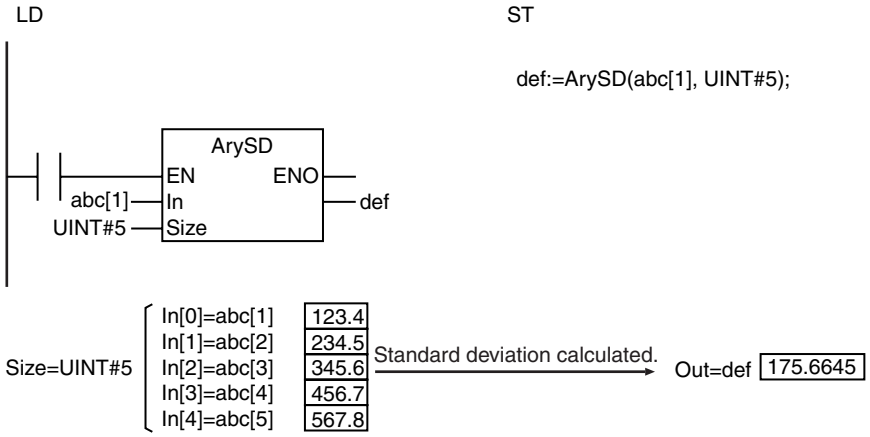
Function

The ArySD instruction calculates the standard deviation of *Size* elements of array to process *In[]* starting from *In[0]*.

$$\text{Standard deviation} = \sqrt{\frac{\sum_{i}(\text{In}[i]-\text{InM})^{2}}{\text{Size}-1}}$$

i: Subscript of *In[]*, 0 to *Size* – 1
InM: Average value of *In[0]* to *In[Size - 1]*

The following example is for when *Size* is UINT#5.

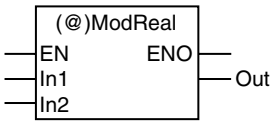


Precautions for Correct Use

- If the value of *Size* is 0 or 1, the value of *Out* is 0.
- If an intermediate value in the calculation process exceeds the valid range of *IN[]*, *Out* will contain an illegal value. An error will not occur.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* exceeds the array area of *In[]*.

ModReal

The ModReal instruction calculates the remainder of real number division.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ModReal	Real Number Modulo-division	FUN		Out:=ModReal(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Dividend	Input	Dividend	Depends on data type.	---	*
In2	Divisor		Divisor			
Out	Remainder	Output	Remainder	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

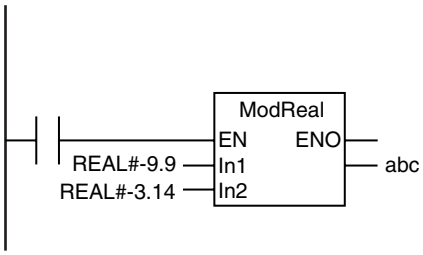
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1														OK	OK					
In2														OK	OK					
Out														OK	OK					

Function

The ModReal instruction divides dividend *In1* by divisor *In2* to find the remainder.

The following example is for when *In1* is REAL#–9.9 and *In2* is REAL#–3.14. The value of variable *abc* will be REAL#–0.48.

LD



ST

```
abc:=ModReal(REAL#-9.9, REAL#-3.14);
```

Additional Information

Use the CheckReal instruction (page 2-209) to see if the value of *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- The following table shows the values of *Out* for different combinations of *In1* and *In2* values.

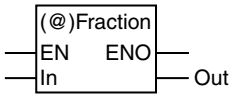
		In1				
		0	**Number**	+∞	−∞	**Nonnumeric data**
In2	**0**	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data
	Number	0	Remainder of *In1*/*In2*	Nonnumeric data	Nonnumeric data	Nonnumeric data
	+∞	0	Value of *In1*	Nonnumeric data	Nonnumeric data	Nonnumeric data
	−∞	0	Value of *In1*	Nonnumeric data	Nonnumeric data	Nonnumeric data
	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In1* or *In2*, the data type is converted as follows:

Data type of parameter that is passed to *In1* or *In2*	**Data type of *In1* or *In2***
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

Fraction

The Fraction instruction finds the fractional part of a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Fraction	Real Number Fraction	FUN		Out:=Fraction(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Real number	Input	Real number	Depends on data type.	---	*
Out	Fractional part	Output	Fractional part	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

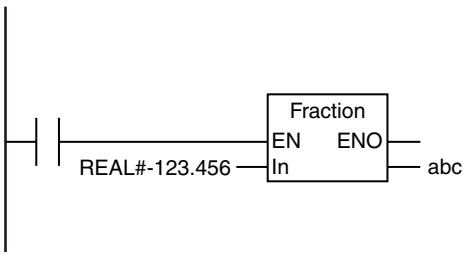
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

The Fraction instruction finds the fractional part of real number *In*.

The following example is for when *In* is REAL#–123.456. The value of variable *abc* will be REAL#–0.456.

LD



ST

```
abc:=Fraction(REAL#-123.456);
```

Additional Information

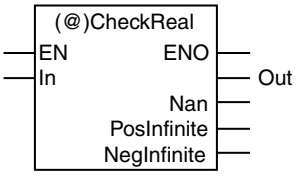
- Use the CheckReal instruction (page 2-209) to see if the value of *Out* is positive infinity, negative infinity, or nonnumeric data.

- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

CheckReal

The CheckReal instruction checks a real number to see if it is infinity or nonnumeric data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CheckReal	Real Number Check	FUN		CheckReal(In, Nan, PosInfinite, NegInfinite);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Real number	Input	Real number	Depends on data type.	---	*
Out	Return value	Output	Always TRUE	TRUE only	---	---
Nan	Nonnumeric data check result		TRUE: Nonnumeric data FALSE: Not nonnumeric data	Depends on data type.		
PosInfinite	Positive infinity check result		TRUE: Positive infinity FALSE: Not positive infinity			
NegInfinite	Negative infinity check result		TRUE: Negative infinity FALSE: Not negative infinity			

* If you omit an input parameter, the default value is not applied. A building error will occur.

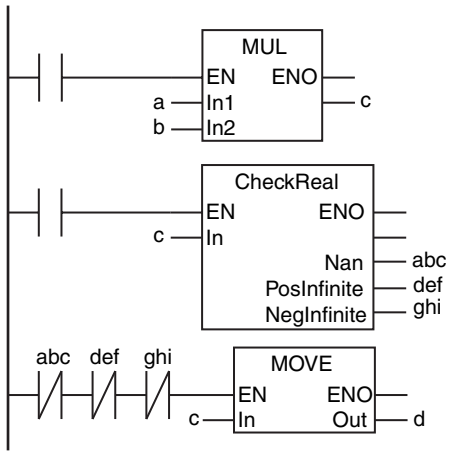
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	OK																			
Nan	OK																			
PosInfinite	OK																			
NegInfinite	OK																			

Function

The CheckReal instruction checks a real number In to see if it is nonnumeric data, positive infinity, or negative infinity. It outputs the results to *Nan*, *PosInfinite*, and *NegInfinite*.

The following figure shows a programming example. The values of REAL variables *a* and *b* are multiplied and the result is tested to see if it is a real number. If the multiplication result is a real number, it is assigned to variable *d*.

LD



ST

```
c:=a*b;
CheckReal(c, abc, def, ghi);
IF ( (abc=FALSE) AND (def=FALSE) AND (ghi=FALSE) ) THEN
    d:=c;
END_IF;
```

Additional Information

Use this instruction on the result of a math instruction that handles real numbers to see if the result is nonnumeric data, positive infinity, or negative infinity.

Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

BCD Conversion Instructions

Instruction	Name	Page
_BCD_TO_*	BCD-to-Unsigned Integer Conversion Group	2-212
_TO_BCD_*	Unsigned Integer-to-BCD Conversion Group	2-215
BCD_TO_**	BCD Data Type-to-Unsigned Integer Conversion Group	2-218
BCDsToBin	Signed BCD-to-Signed Integer Conversion	2-221
BinToBCDs_**	Signed Integer-to-BCD Conversion Group	2-224
AryToBCD	Array BCD Conversion	2-227
AryToBin	Array Unsigned Integer Conversion	2-229

_BCD_TO_*

These instructions convert BCD bit strings into unsigned integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_BCD_TO_*	BCD-to-Unsigned Integer Conversion Group	FUN	(@)**_BCD_TO_*** EN ENO In Out "**" must be a bit string data type. "****" must be an integer data type.	Out:=**_BCD_TO_*** (In); "**" must be a bit string data type. "****" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK	OK	OK	OK	OK	OK	OK	OK							

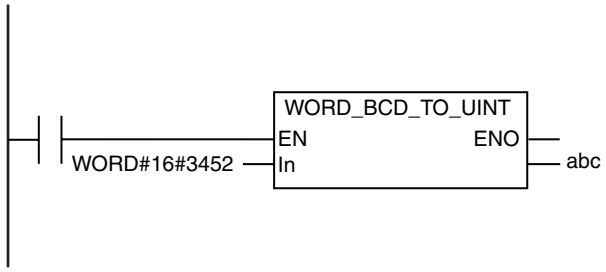
Function

These instructions convert data to convert *In* (which must be a BCD bit string) into an unsigned integer.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is UINT data, the name of the instruction is WORD_BCD_TO_UINT.

The following example for the WORD_BCD_TO_UINT instruction is for when *In* is WORD16#3452.

LD



ST

```
abc:=WORD_BCD_TO_UINT(WORD#16#3452);
```

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	USINT	16#00 to 16#99 (BCD)	0 to 99
	UINT		
	UDINT		
	ULINT		
	SINT		
	INT		
	DINT		
	LINT		
WORD	USINT	16#0000 to 16#0255 (BCD)	0 to 255
	UINT	16#0000 to 16#9999 (BCD)	0 to 9999
	UDINT		
	ULINT		
	SINT	16#0000 to 16#0127 (BCD)	0 to 127
	INT	16#0000 to 16#9999 (BCD)	0 to 9999
	DINT		
	LINT		
DWORD	USINT	16#0000_0000 to 16#0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000 to 16#0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	ULINT		
	SINT	16#0000_0000 to 16#0000_0127 (BCD)	0 to 127
	INT	16#0000_0000 to 16#0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	LINT		
LWORD	USINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000_0000_0000 to 16#0000_0000_0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000_0000_0000 to 16#0000_0042_9496_7295 (BCD)	0 to 4294967295
	ULINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999
	SINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0127 (BCD)	0 to 127
	INT	16#0000_0000_0000_0000 to 16#0000_0000_0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000_0000_0000 to 16#0000_0021_4748_3647 (BCD)	0 to 2147483647
	LINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999

Additional Information

- To convert a BCD bit string to an integer, use a BCD_TO_** instruction (page 2-218).
- To convert an integer to a BCD bit string, use a **_TO_BCD_*** instruction (page 2-215).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.
- The value in *In* is not BCD bit string data (i.e., contains A, B, C, D, E, or F hexadecimal).

_TO_BCD_*

These instructions convert unsigned integers to BCD bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_BCD_*	Unsigned Integer-to-BCD Conversion Group	FUN	(@)**_TO_BCD_*** EN ENO In Out "**" must be an integer data type. "***" must be a bit string data type.	Out:=**_TO_BCD_*** (In); "**" must be an integer data type. "***" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out		OK	OK	OK	OK															

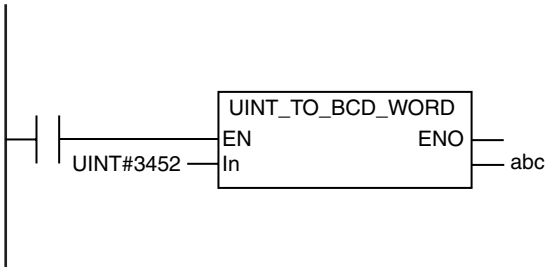
Function

These instructions convert data to convert *In* (which must be an unsigned integer) to a BCD bit string.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is UINT data and *Out* is WORD data, the name of the instruction is UINT_TO_BCD_WORD.

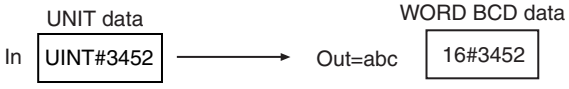
The following example for the UINT_TO_BCD_WORD instruction is for when *In* is UNIT#3452.

LD



ST

```
abc:=UINT_TO_BCD_WORD(UINT#3452);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
USINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 255	16#0000 to 16#0255 (BCD)
	DWORD		16#0000_0000 to 16#000_0255 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0000_0255 (BCD)
UINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 65535	16#0000_0000 to 16#0006_5535 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0006_5535 (BCD)
UDINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 4294967295	16#0000_0000_0000_0000 to 16#0000_0042_9496_7295 (BCD)
ULINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 9999999999999999	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)
SINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 127	16#0000 to 16#0127 (BCD)
	DWORD		16#0000_0000 to 16#0000_0127 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0000_0127 (BCD)
INT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 32767	16#0000_0000 to 16#0003_2767 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0003_2767 (BCD)
DINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 2147483647	16#0000_0000_0000_0000 to 16#0000_0021_4748_3647 (BCD)
LINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 9999999999999999	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)

Additional Information

- To convert a specific BCD bit string to an integer, use a **_BCD_TO_*** instruction (page 2-212).
- To convert a BCD bit string to an integer, use a BCD_TO_** instruction (page 2-218).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.

BCD_TO_**

The BCD_TO_** instruction converts BCD bit strings into unsigned integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BCD_TO_**	BCD Data Type-to-Unsigned Integer Conversion Group	FUN	(@)BCD_TO_** EN ENO In Out "**" must be an integer data type.	Out:=BCD_TO_** (In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

*2 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK	OK	OK	OK	OK	OK	OK	OK							

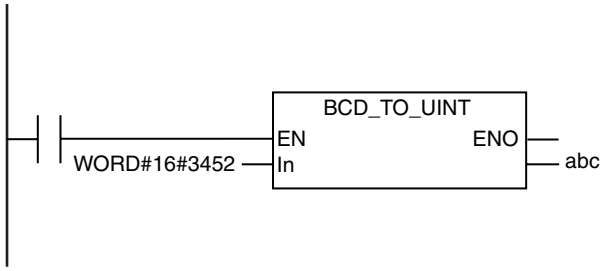
Function

These instructions convert data to convert *In* (which must be a BCD bit string) into an unsigned integer.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the UINT data type, the instruction is BCD_TO_UINT.

The following example for the BCD_TO_UINT instruction is for when *In* is WORD#16#3452.

LD



ST

```
abc:=BCD_TO_UINT(WORD#16#3452);
```

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	USINT	16#00 to 16#99 (BCD)	0 to 99
	UINT		
	UDINT		
	ULINT		
	SINT		
	INT		
	DINT		
	LINT		
WORD	USINT	16#0000 to 16#0255 (BCD)	0 to 255
	UINT	16#0000 to 16#9999 (BCD)	0 to 9999
	UDINT		
	ULINT		
	SINT	16#0000 to 16#0127 (BCD)	0 to 127
	INT	16#0000 to 16#9999 (BCD)	0 to 9999
	DINT		
	LINT		
DWORD	USINT	16#0000_0000 to 16#0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000 to 16#0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	ULINT		
	SINT	16#0000_0000 to 16#0000_0127 (BCD)	0 to 127
	INT	16#0000_0000 to 16#0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	LINT		
LWORD	USINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000_0000_0000 to 16#0000_0000_0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000_0000_0000 to 16#0000_0042_9496_7295 (BCD)	0 to 4294967295
	ULINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999
	SINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0127 (BCD)	0 to 127
	INT	16#0000_0000_0000_0000 to 16#0000_0000_0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000_0000_0000 to 16#0000_0021_4748_3647 (BCD)	0 to 2147483647
	LINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999

Additional Information

- To convert a specific BCD bit string to an integer, use a **_BCD_TO_*** instruction (page 2-212).
- To convert an integer to a BCD bit string, use a **_TO_BCD_*** instruction (page 2-215).

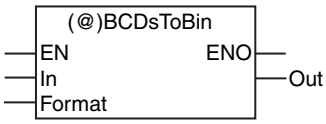
Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.
- The value in *In* is not BCD bit string data (i.e., contains A, B, C, D, E, or F hexadecimal).

BCDsToBin

The BCDsToBin instruction converts signed BCD bit strings to signed integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BCDsToBin	Signed BCD-to-Signed Integer Conversion	FUN		Out:=BCDsToBin(In, Format);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Format	Data format number		Format of BCD bit string	_BCD0 to _BCD3		_BCD0
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid range depends on the value of *Format*. Refer to *Function*, below, for details.

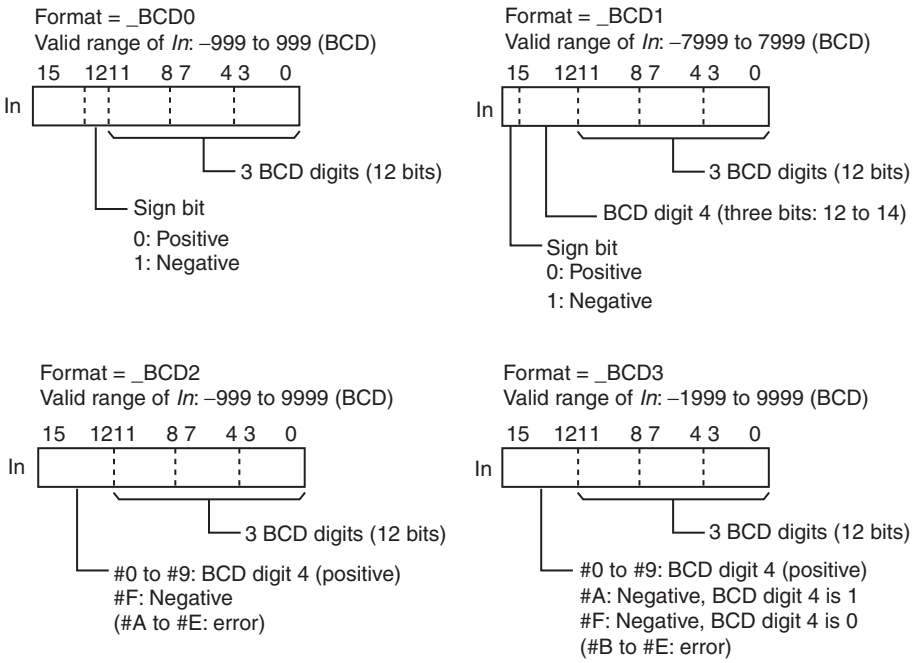
*2 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Format	Refer to *Function* for the enumerators of the enumerated type _eBCD_FORMAT.																			
Out	Must be a signed integer data type that is the same size as *In.*																			

Function

The BCDsToBin instruction converts signed BCD bit string *In* to a signed integer.

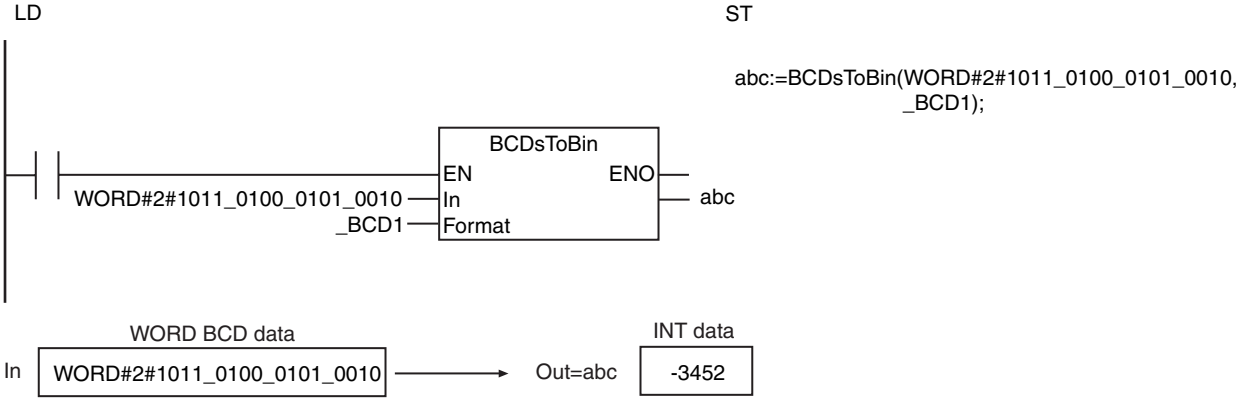
The data type of data format number *Format* is enumerated type _eBCD_FORMAT. Select one of the following: _BCD0, _BCD1, _BCD2, or _BCD3. The sign specification in the upper four bits of *In* depends on the BCD format number. The data format examples shown below use WORD data for *In.*



The same sizes of data types are used for *In* and *Out*. The valid ranges depend on the value of *Format*, as shown below.

		Value of *Format*			
		_BCD0	**_BCD1**	**_BCD2**	**_BCD3**
Data type of *In* ↓ Data type of *Out*	**BYTE ↓ SINT**	–9 to 9	–79 to 79	–9 to 99	–19 to 99
	WORD ↓ INT	–999 to 999	–7999 to 7999	–999 to 9999	–1999 to 9999
	DWORD ↓ DINT	–9999999 to 9999999	–79999999 to 79999999	–9999999 to 99999999	–19999999 to 99999999
	LWORD ↓ LINT	–999999999999999 to 999999999999999	–7999999999999999 to 7999999999999999	–999999999999999 to 9999999999999999	–1999999999999999 to 9999999999999999

The following example is for when In is WORD#2#1011_0100_0101_0010 and *Format* is _BCD1.



Precautions for Correct Use

- Use the same sizes of data types for *In* and *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Format* is _BCD0 and the upper digit of *In* is 2 to F.
  - The value of *Format* is _BCD2 and the upper digit of *In* is A to E.
  - The value of *Format* is _BCD3 and the upper digit of *In* is B to E.
  - Except for the above conditions, any digit in *In* is A to F.
  - The value of *Format* is outside of the valid range.

BinToBCDs_**

These instructions convert signed integers to signed BCD bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BinToBCDs_**	Signed Integer-to-BCD Conversion Group	FUN	(@)BinToBCDs_** EN In Format ENO Out "**" must be a bit string data type.	Out:=BinToBCDs(In, Format); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Format	Data format number		Format of BCD bit string	_BCD0 to _BCD3		_BCD0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid range depends on the value of *Format*. Refer to *Function*, below, for details.

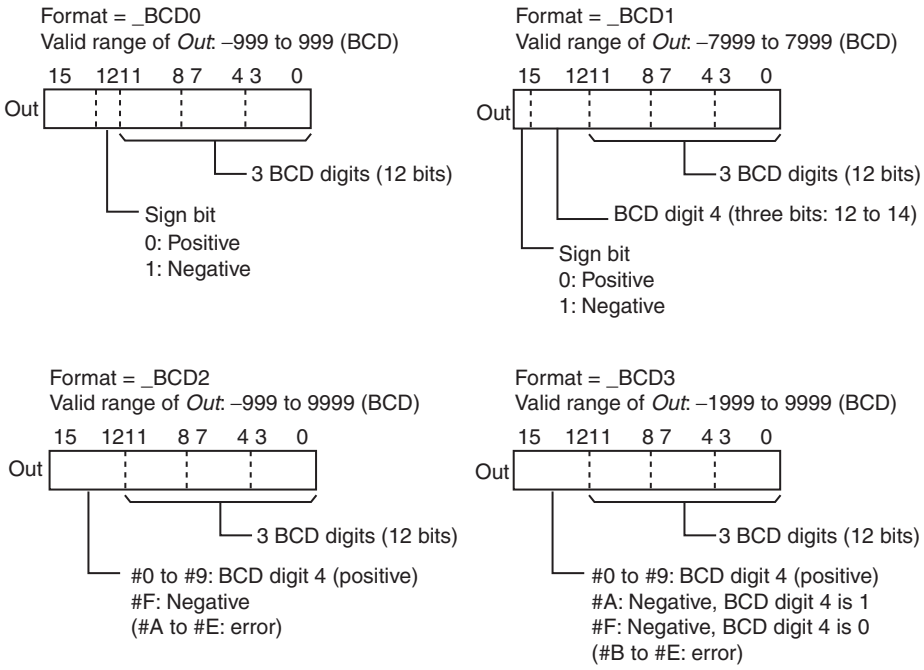
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In										OK	OK	OK	OK							
Format	Refer to *Function* for the enumerators of the enumerated type _eBCD_FORMAT.																			
Out	Must be same size of data type as *In*																			

Function

These instructions convert signed integer *In* to a signed BCD bit string.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the WORD data type, the instruction is BinToBCDs_WORD.

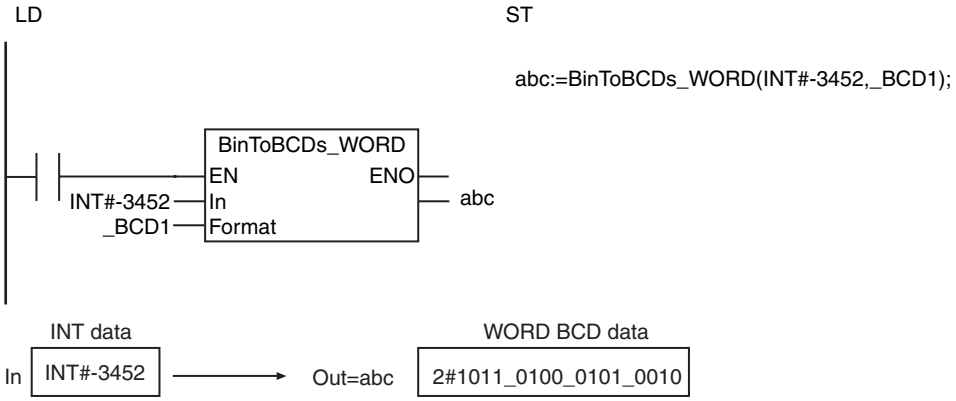
The data type of data format number *Format* is enumerated type _eBCD_FORMAT. Select one of the following: _BCD0, _BCD1, _BCD2, or _BCD3. The sign specification in the upper four bits of *Out* depends on the BCD format number. The data format examples shown below use WORD data for *Out.*



The same sizes of data types are used for *In* and *Out*. The valid ranges depend on the value of *Format*, as shown below.

		Value of *Format*			
		_BCD0	**_BCD1**	**_BCD2**	**_BCD3**
Data type of *In* ↓ Data type of *Out*	**SINT ↓ BYTE**	–9 to 9	–79 to 79	–9 to 99	–19 to 99
	INT ↓ WORD	–999 to 999	–7999 to 7999	–999 to 9999	–1999 to 9999
	DINT ↓ DWORD	–9999999 to 9999999	–79999999 to 79999999	–9999999 to 99999999	–19999999 to 99999999
	LINT ↓ LWORD	–999999999999999 to 999999999999999	–7999999999999999 to 7999999999999999	–999999999999999 to 9999999999999999	–1999999999999999 to 9999999999999999

The following example shows the BinToBCDs_WORD instruction when *In* is INT#–3452 and *Format* is _BCD1.

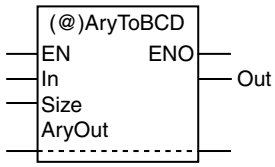


Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In* is outside of the valid range.
  - The value of *Format* is outside of the valid range.

AryToBCD

The AryToBCD instruction converts the elements of an unsigned integer array to BCD bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryToBCD	Array BCD Conversion	FUN		AryToBCD(In, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Unsigned integer array	Input	Unsigned integer array	*1	---	*2
Size	Number of elements		Number of elements of *In[]* for conversion	Depends on data type.		1
AryOut[] (array)	BCD array	In-out	BCD array	*1	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 The valid ranges depend on the data types of the elements of *In[]* and *AryOut[]*. Refer to *Function* for details.

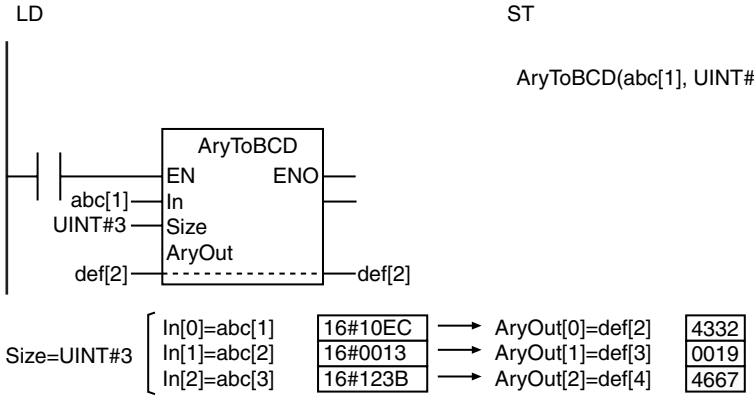
*2 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)						OK	OK	OK	OK											
Size							OK													
AryOut[] (array)	Must be a bit string array. The data type must be the same size as the elements of *In[]*.																			
Out	OK																			

Function

The AryToBCD instruction converts *Size* elements of unsigned integer array *In[]* starting from *In[0]* to a BCD bit string. It outputs the BCD bit string to BCD array *AryOut[]*.

The following example is for when *Size* is UINT#3.



The following table shows the valid ranges for *In[]* and *AryOut[]* according to the data types of their elements.

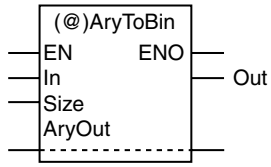
Data type of the elements of *In[]*	Data type of the elements of *AryOut[]*	Valid range of *In[]*	Valid range of *AryOut[]*
USINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
UINT	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
UDINT	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
ULINT	LWORD	0 to 9999999999999999	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)

Precautions for Correct Use

- Use the same data type and size for *In[]* and *AryOut[]*. For example, if the elements of *In[]* are UINT data, use WORD as the data type of the elements of *AryOut[]*.
- This instruction does not convert signed binary to signed BCD. Use an unsigned integer (USINT, UINT, UDINT, or ULINT) as the data type of *In[]*.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *In[]* is outside of the valid range.
  - The data type sizes of *In[]* and *AryOut[]* are different.
  - The value of *Size* exceeds the array area of *In[]* or *AryOut[]*.

AryToBin

The AryToBin instruction converts the elements of an array of BCD bit strings into unsigned integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryToBin	Array Unsigned Integer Conversion	FUN		AryToBin(In, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array of BCD bit strings	Input	Array of BCD bit strings	*1	---	*2
Size	Number of elements		Number of elements of *In[]* for conversion	Depends on data type.		1
AryOut[] (array)	Unsigned integer array	In-out	Unsigned integer array	*1	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 The valid ranges depend on the data types of the elements of *In[]* and *AryOut[]*. Refer to *Function* for details.

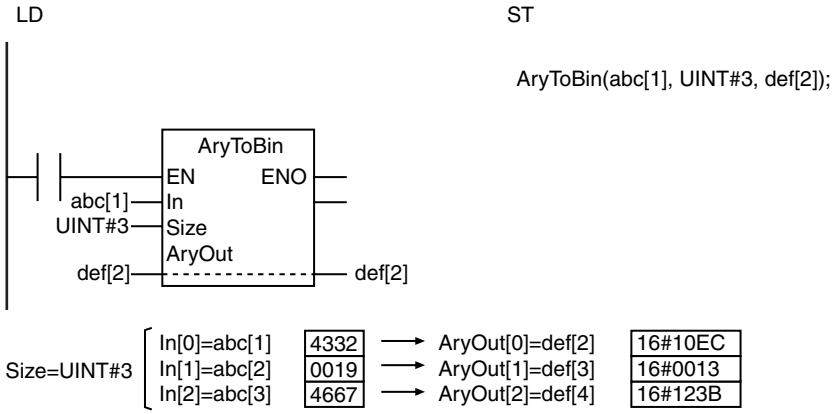
*2 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK															
Size							OK													
AryOut[] (array)	Must be an unsigned integer array. The data type must be the same size as the elements of *In[]*.																			
Out	OK																			

Function

The AryToBin instruction converts *Size* elements of array of BCD bit strings *In[]* starting from *In[0]* to unsigned integers. It outputs the unsigned integers to unsigned integer array *AryOut[]*.

The following example is for when *Size* is UINT#3.



The following table shows the valid ranges for *In[]* and *AryOut[]* according to the data types of their elements.

Data type of the elements of *In[]*	Data type of the elements of *AryOut[]*	Valid range of *In[]*	Valid range of *AryOut[]*
BYTE	USINT	16#00 to 16#99 (BCD)	0 to 99
WORD	UINT	16#0000 to 16#9999 (BCD)	0 to 9999
DWORD	UDINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
LWORD	ULINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999

Precautions for Correct Use

- Use the same data type and size for *In[]* and *AryOut[]*. For example, if the elements of *In[]* are WORD data, use USINT as the data type of the elements of *AryOut[]*.
- This instruction does not convert signed BCD to signed binary. Use an unsigned integer (USINT, UINT, UDINT, or ULINT) as the data type of AryOut *[].*
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The data type sizes of *In[]* and *AryOut[]* are different.
  - The value of *Size* exceeds the array area of *In[]* or *AryOut[]*.
  - A value in In[] is not a BCD bit string (i.e., contains A, B, C, D, E, or F hexadecimal).

Data Type Conversion Instructions

Instruction	Name	Page	Instruction	Name	Page
_TO_* (Integer-to-Integer Conversion Group)	Integer-to-Integer Conversion Group	2-232	**_TO_STRING (Real Number-to-Text String Conversion Group)	Real Number-to-Text String Conversion Group	2-257
_TO_* (Integer-to-Bit String Conversion Group)	Integer-to-Bit String Conversion Group	2-235	RealToFormatString	REAL-to-Formatted Text String	2-259
_TO_* (Integer-to-Real Number Conversion Group)	Integer-to-Real Number Conversion Group	2-237	LrealToFormatString	LREAL-to-Formatted Text String	2-264
_TO_* (Bit String-to-Integer Conversion Group)	Bit String-to-Integer Conversion Group	2-239	STRING_TO_** (Text String-to-Integer Conversion Group)	Text String-to-Integer Conversion Group	2-270
_TO_* (Bit String-to-Bit String Conversion Group)	Bit String-to-Bit String Conversion Group	2-242	STRING_TO_** (Text String-to-Bit String Conversion Group)	Text String-to-Bit String Conversion Group	2-272
_TO_* (Bit String-to-Real Number Conversion Group)	Bit String-to-Real Number Conversion Group	2-244	STRING_TO_** (Text String-to-Real Number Conversion Group)	Text String-to-Real Number Conversion Group	2-274
_TO_* (Real Number-to-Integer Conversion Group)	Real Number-to-Integer Conversion Group	2-246	TO_** (Integer Conversion Group)	Integer Conversion Group	2-277
_TO_* (Real Number-to-Bit String Conversion Group)	Real Number-to-Bit String Conversion Group	2-249	TO_** (Bit String Conversion Group)	Bit String Conversion Group	2-279
_TO_* (Real Number-to-Real Number Conversion Group)	Real Number-to-Real Number Conversion Group	2-251	TO_** (Real Number Conversion Group)	Real Number Conversion Group	2-281
**_TO_STRING (Integer-to-Text String Conversion Group)	Integer-to-Text String Conversion Group	2-253	TRUNC, Round, and RoundUp	Truncate/Round Off Real Number/Round Up Real Number	2-283
**_TO_STRING (Bit String-to-Text String Conversion Group)	Bit String-to-Text String Conversion Group	2-255			

_TO_* (Integer-to-Integer Conversion Group)

These instructions convert integers to integers with different data types.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Integer-to-Integer Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" and "***" must be different integer data types.	Out:=**_TO_*** (In); "**" and "***" must be different integer data types.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out						OK	OK	OK	OK	OK	OK	OK	OK							

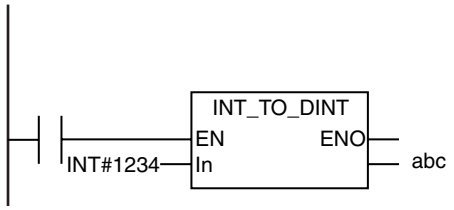
Function

These instructions convert an integer, *In*, to an integer with a different data type.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is INT data and *Out* is DINT data, the name of the instruction is INT_TO_DINT.

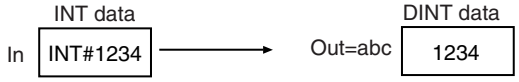
The following example for the INT_TO_DINT instruction is for when *In* is INT#1234.

LD



ST

```
abc:=INT_TO_DINT(INT#1234);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
USINT	UINT	0 to 255
	UDINT	
	ULINT	
	SINT	0 to 127
	INT	0 to 255
	DINT	
	LINT	
UINT	USINT	0 to 255
	UDINT	0 to 65535
	ULINT	
	SINT	0 to 127
	INT	0 to 32767
	DINT	0 to 65535
	LINT	
UDINT	USINT	0 to 255
	UINT	0 to 65535
	ULINT	0 to 4294967295
	SINT	0 to 127
	INT	0 to 32767
	DINT	0 to 2147483647
	LINT	0 to 4294967295
ULINT	USINT	0 to 255
	UINT	0 to 65535
	UDINT	0 to 4294967295
	SINT	0 to 127
	INT	0 to 32767
	DINT	0 to 2147483647
	LINT	0 to 9223372036854775807
SINT	USINT	0 to 127
	UINT	
	UDINT	
	ULINT	
	INT	−128 to 127
	DINT	
	LINT	
INT	USINT	0 to 255
	UINT	0 to 32767
	UDINT	
	ULINT	
	SINT	−128 to 127
	DINT	−32768 to 32767
	LINT	
DINT	USINT	0 to 255
	UINT	0 to 65535
	UDINT	0 to 2147483647
	ULINT	
	SINT	−128 to 127
	INT	−32768 to 32767
	LINT	−2147483648 to 2147483647

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
LINT	USINT	0 to 255
	UINT	0 to 65535
	UDINT	0 to 4294967295
	ULINT	0 to 9223372036854775807
	SINT	−128 to 127
	INT	−32768 to 32767
	DINT	−2147483648 to 2147483647

Additional Information

To convert data with any data type to integer data, use a TO_** (Integer Conversion Group) instruction (page 2-277).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If *In* is a signed integer and the data size of *Out* is larger than the data size of *In*, sign extension is performed.
- If *In* is an unsigned integer and the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.

_TO_* (Integer-to-Bit String Conversion Group)

These instructions convert integers to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Integer-to-Bit String Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be an integer data type. "***" must be a bit string data type.	Out:=**_TO_*** (In); "**" must be an integer data type. "***" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out		OK	OK	OK	OK															

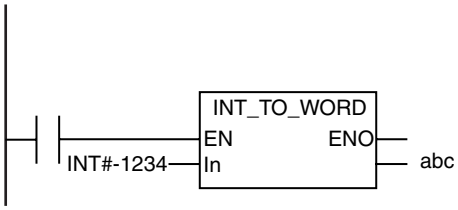
Function

These instructions convert an integer, *In*, to a bit string.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is INT data and *Out* is WORD data, the name of the instruction is INT_TO_WORD.

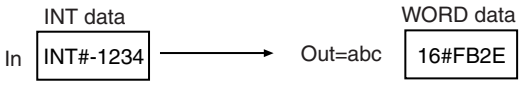
The following example for the INT_TO_WORD instruction is for when *In* is INT#–1234.

LD



ST

```
abc:=INT_TO_WORD(INT#-1234);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
USINT	BYTE	0 to 255	16#00 to 16#FF
	WORD		
	DWORD		
	LWORD		
UINT	BYTE	0 to 255	16#00 to 16#FF
	WORD	0 to 65535	16#0000 to 16#FFFF
	DWORD		
	LWORD		
UDINT	BYTE	0 to 255	16#00 to 16#FF
	WORD	0 to 65535	16#0000 to 16#FFFF
	DWORD	0 to 4294967295	16#0000_0000 to 16#FFFF_FFFF
	LWORD		
ULINT	BYTE	0 to 255	16#00 to 16#FF
	WORD	0 to 65535	16#0000 to 16#FFFF
	DWORD	0 to 4294967295	16#0000_0000 to 16#FFFF_FFFF
	LWORD	0 to 18446744073709551615	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF
SINT	BYTE	−128 to 127	16#00 to 16#FF
	WORD		
	DWORD		
	LWORD		
INT	BYTE	−128 to 127	16#00 to 16#FF
	WORD	−32768 to 32767	16#0000 to 16#FFFF
	DWORD		
	LWORD		
DINT	BYTE	−128 to 127	16#00 to 16#FF
	WORD	−32768 to 32767	16#0000 to 16#FFFF
	DWORD	−2147483648 to 2147483647	16#0000_0000 to 16#FFFF_FFFF
	LWORD		
LINT	BYTE	−128 to 127	16#00 to 16#FF
	WORD	−32768 to 32767	16#0000 to 16#FFFF
	DWORD	−2147483648 to 2147483647	16#0000_0000 to 16#FFFF_FFFF
	LWORD	−9223372036854775808 to 9223372036854775807	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF

Additional Information

- To convert a bit string to an integer, use a **_TO_*** (Bit String-to-Integer Conversion Group) instruction (page 2-239).
- To convert data with any data type to a bit string, use a TO_** (Bit String Conversion Group) instruction (page 2-279).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If *In* is a signed integer and the data size of *Out* is larger than the data size of *In*, sign extension is performed.
- If *In* is an unsigned integer and the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.

_TO_* (Integer-to-Real Number Conversion Group)

These instructions convert integers to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Integer-to-Real Number Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be an integer data type. "***" must be a real number data type.	Out:=**_TO_*** (In); "**" must be an integer data type. "***" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out														OK	OK					

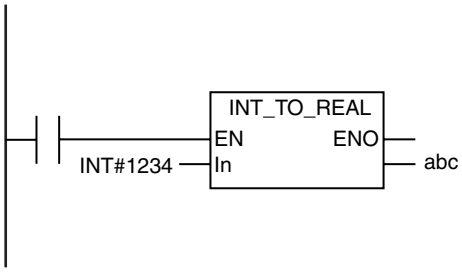
Function

These instructions convert an integer, *In*, to a real number.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is INT data and *Out* is REAL data, the name of the instruction is INT_TO_REAL.

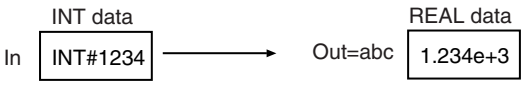
The following example for the INT_TO_REAL instruction is for when *In* is INT#1234.

LD



ST

```
abc:=INT_TO_REAL(INT#1234);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
USINT	REAL	0 to 255	0 to 2.55e+2
	LREAL		
UINT	REAL	0 to 65535	0 to 6.5535e+4
	LREAL		
UDINT	REAL	0 to 4294967295	0 to 4.294967e+9
	LREAL		0 to 4.294967295e+9
ULINT	REAL	0 to 18446744073709551615	0 to 1.844674e+19
	LREAL		0 to 1.84467440737095e+19
SINT	REAL	−128 to 127	−1.28e+2 to 1.27e+2
	LREAL		
INT	REAL	−32768 to 32767	−3.2768e+4 to 3.2767e+4
	LREAL		
DINT	REAL	−2147483648 to 2147483647	−2.147483e+9 to 2.147483e+9
	LREAL		−2.147483648e+9 to 2.147483647e+9
LINT	REAL	−9223372036854775808 to 9223372036854775807	−9.223372e+18 to 9.223372e+18
	LREAL		−9.22337203685477e+18 to 9.22337203685477e+18

Additional Information

- To convert a real number to an integer, use a **_TO_*** (Real Number-to-Integer Conversion Group) instruction (page 2-246).
- To convert data with any data type to a real number, use a TO_** (Real Number Conversion Group) instruction (page 2-281).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- Depending on the data types of *In* and *Out*, rounding will be performed for the effective digits of the real number. This will cause error between the values before and after conversion. The following table lists the data types that result in error.

Data type of *In*	Data type of *Out*	Values for which error occurs
DINT	REAL	−16777216 or lower, or 16777216 or higher
LINT		
UDINT	REAL	16777216 or higher
ULINT		
LINT	LREAL	−9007199254740992 or lower, or 9007199254740992 or higher
ULINT	LREAL	9007199254740992 or higher

_TO_* (Bit String-to-Integer Conversion Group)

These instructions convert bit strings to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Bit String-to-Integer Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be a bit string data type. "***" must be an integer data type.	Out:=**_TO_*** (In); "**" must be a bit string data type. "***" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK	OK	OK	OK	OK	OK	OK	OK							

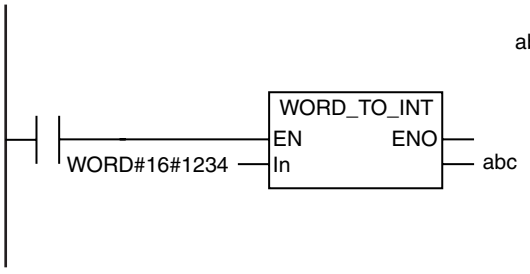
Function

These instructions convert a bit string, *In*, to an integer.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is INT data, the name of the instruction is WORD_TO_INT.

The following example for the WORD_TO_INT instruction is for when *In* is WORD #16#1234.

LD



ST

```
abc:=WORD_TO_INT(WORD#16#1234);
```

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	USINT	16#00 to 16#FF	0 to 255
	UINT		
	UDINT		
	ULINT		
	SINT		–128 to 127
	INT		
	DINT		
	LINT		
WORD	USINT	16#00 to 16#FF	0 to 255
	UINT	16#0000 to 16#FFFF	0 to 65535
	UDINT		
	ULINT		
	SINT	16#00 to 16#FF	–128 to 127
	INT	16#0000 to 16#FFFF	–32768 to 32767
	DINT		
	LINT		
DWORD	USINT	16#00 to 16#FF	0 to 255
	UINT	16#0000 to 16#FFFF	0 to 65535
	UDINT	16#0000_0000 to 16#FFFF_FFFF	0 to 4294967295
	ULINT		
	SINT	16#00 to 16#FF	–128 to 127
	INT	16#0000 to 16#FFFF	–32768 to 32767
	DINT	16#0000_0000 to 16#FFFF_FFFF	–2147483648 to 2147483647
	LINT		
LWORD	USINT	16#00 to 16#FF	0 to 255
	UINT	16#0000 to 16#FFFF	0 to 65535
	UDINT	16#0000_0000 to 16#FFFF_FFFF	0 to 4294967295
	ULINT	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF	0 to 18446744073709551645
	SINT	16#00 to 16#FF	–128 to 127
	INT	16#0000 to 16#FFFF	–32768 to 32767
	DINT	16#0000_0000 to 16#FFFF_FFFF	–2147483648 to 2147483647
	LINT	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF	-9223372036854775808 to 9223372036854775807

Additional Information

- To convert an integer to a bit string, use a **_TO_*** (Integer-to-Bit String Conversion Group) instruction (page 2-235).
- To convert data with any data type to a bit string, use a TO_** (Bit String Conversion Group) instruction (page 2-279).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.

- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.

_TO_* (Bit String-to-Bit String Conversion Group)

These instructions convert bit strings to bit strings with different data types.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Bit String-to-Bit String Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" and "***" must be different bit string data types.	Out:=**_TO_*** (In); "**" and "***" must be different bit string data types.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out		OK	OK	OK	OK															

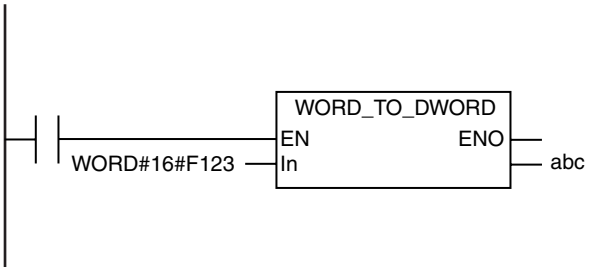
Function

These instructions convert a bit string, *In*, to a bit string with a different data type.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is DWORD data, the name of the instruction is WORD_TO_DWORD.

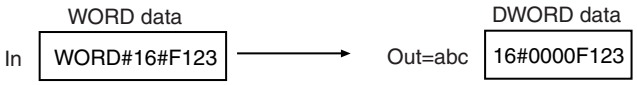
The following example for the WORD_TO_DWORD instruction is for when *In* is WORD#16#F123.

LD



ST

```
abc:=WORD_TO_DWORD(WORD#16#F123);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
BYTE	WORD	16#00 to 16#FF
	DWORD	
	LWORD	
WORD	BYTE	16#00 to 16#FF
	DWORD	16#0000 to 16#FFFF
	LWORD	
DWORD	BYTE	16#00 to 16#FF
	WORD	16#0000 to 16#FFFF
	LWORD	16#0000_0000 to 16#FFFF_FFFF
LWORD	BYTE	16#00 to 16#FF
	WORD	16#0000 to 16#FFFF
	DWORD	16#0000_0000 to 16#FFFF_FFFF

Additional Information

To convert data with any data type to a bit string, use a TO_** (Bit String Conversion Group) instruction (page 2-279).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated when the data is output to *Out*.

_TO_* (Bit String-to-Real Number Conversion Group)

These instructions convert bit strings to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Bit String-to-Real Number Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be a bit string data type. "***" must be a real number data type.	Out:=**_TO_*** (In); "**" must be a bit string data type. "***" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out														OK	OK					

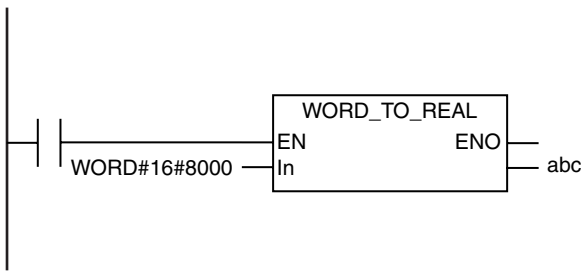
Function

These instructions take a bit string, *In*, as an unsigned integer of the same size and convert it to a real number.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is REAL data, the name of the instruction is WORD_TO_REAL.

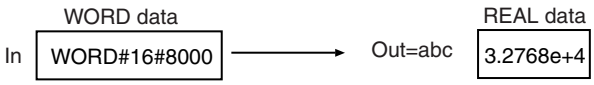
The following example for the WORD_TO_REAL instruction is for when *In* is WORD#16#8000.

LD



ST

```
abc:=WORD_TO_REAL(WORD#16#8000);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	REAL	16#00 to 16#FF	0 to 2.55e+2
	LREAL		
WORD	REAL	16#0000 to 16#FFFF	0 to 6.5535e+4
	LREAL		
DWORD	REAL	16#0000_0000 to 16#FFFF_FFFF	0 to 4.294967e+9
	LREAL		0 to 4.294967295e+9
LWORD	REAL	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF	0 to 1.844674e+19
	LREAL		0 to 1.84467440737095e+19

Additional Information

- To convert a real number to a bit string, use a **_TO_*** (Real Number-to-Bit String Conversion Group) instruction (page 2-249).
- To convert data with any data type to a real number, use a TO_** (Real Number Conversion Group) instruction (page 2-281).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- Depending on the data types of *In* and *Out*, rounding will be performed for the effective digits of the real number. This will cause error between the values before and after conversion. The following table lists the data types that result in error.

Data type of *In*	Data type of *Out*	Values for which error occurs
DWORD	REAL	16#0100_0000 or higher
LWORD	LREAL	16#0002_0000_0000_0000 or higher

_TO_* (Real Number-to-Integer Conversion Group)

These instructions convert real numbers to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Real Number-to-Integer Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be a real number data type. "***" must be an integer data type.	Out:=**_TO_*** (In); "**" must be a real number data type. "***" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK							

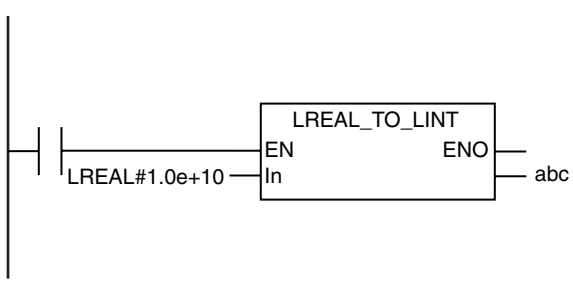
Function

These instructions convert a real number, *In*, to an integer.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is LREAL data and *Out* is LINT data, the name of the instruction is LREAL_TO_LINT.

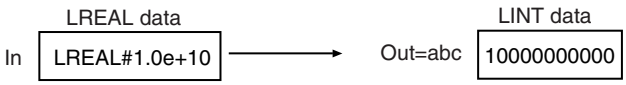
The following example for the LREAL_TO_LINT instruction is for when *In* is LREAL#1.0e+10.

LD



ST

```
abc:=LREAL_TO_LINT(LREAL#1.0e+10);
```



The fractional part of the value of *In* is rounded off to the closest integer. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 −1.49 → −1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 −1.50 → −2 −2.50 → −2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 −1.51 → −2

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
REAL	USINT	0 to 2.55e+2	0 to 255
	UINT	0 to 6.5535e+4	0 to 65535
	UDINT	0 to 4.294967e+9	0 to 4294967295
	ULINT	0 to 1.844674e+19	0 to 18446744073709551615
	SINT	−1.28e+2 to 1.27e+2	−128 to 127
	INT	−3.2768e+4 to 3.2767e+4	−32768 to 32767
	DINT	−2.147483e+9 to 2.147483e+9	−2147483648 to 2147483647
	LINT	−9.223372e+18 to 9.223372e+18	−9223372036854775808 to 9223372036854775807
LREAL	USINT	0 to 0.255e+3	0 to 255
	UINT	0 to 6.5535e+4	0 to 65535
	UDINT	0 to 4.294967295e+9	0 to 4294967295
	ULINT	0 to 1.84467440737095e+19	0 to 18446744073709551615
	SINT	−1.28e+2 to 1.27e+2	−128 to 127
	INT	−3.2768e+4 to 3.2767e+4	−32768 to 32767
	DINT	−2.147483648e+9 to 2.147483647e+9	−2147483648 to 2147483647
	LINT	−9.22337203685477e+18 to 9.22337203685477e+18	−9223372036854775808 to 9223372036854775807

Additional Information

- To convert an integer to a real number, use an Integer-to-Real Number Conversion Group Instruction.
- To convert data with any data type to an integer, use an Integer Conversion Group Instruction.
- You can use the following instructions to convert a real number to an integer: TRUNC (Truncate), Round (Round Off Real Number), and RoundUp (Round Up Real Number). All of these instructions have a REAL input and DINT output, or a LREAL input and LINT output. The differences between these instructions are shown in the following table.

Input value	Output value			
	REAL_TO_INT	TRUNC	Round	RoundUp
REAL#1.6	INT#2	DINT#1	DINT#2	DINT#2
REAL#1.5	INT#2	DINT#1	DINT#2	DINT#2
REAL#1.5	INT#1	DINT#1	DINT#1	DINT#2
REAL#2.5	INT#2	DINT#2	DINT#2	DINT#3
REAL#-1.6	INT#-2	DINT#-1	DINT#-2	DINT#-2
REAL#-1.5	INT#-2	DINT#-1	DINT#-2	DINT#-2
REAL#-1.4	INT#-1	DINT#-1	DINT#-1	DINT#-2
REAL#-2.5	INT#-2	DINT#-2	DINT#-2	DINT#-3

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an illegal value.

_TO_* (Real Number-to-Bit String Conversion Group)

These instructions convert real numbers to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Real Number-to-Bit String Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be a real number data type. "***" must be a bit string data type.	Out:=**_TO_*** (In); "**" must be a real number data type. "***" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out		OK	OK	OK	OK															

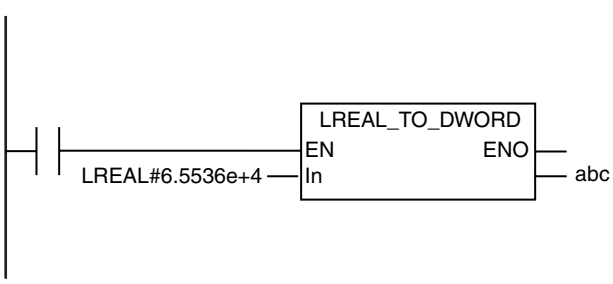
Function

These instructions convert a real number, *In*, to a bit string.

The name of the instruction is determined by the data types of *In* and conversion output *Out*. For example, if *In* is LREAL data and *Out* is DWORD data, the name of the instruction is LREAL_TO_DWORD.

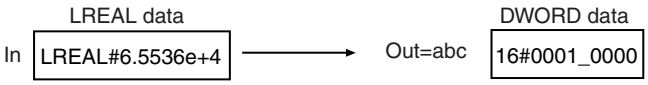
The following example for the LREAL_TO_DWORD instruction is for when *In* is LREAL#6.5536e+4.

LD



ST

```
abc:=LREAL_TO_DWORD(LREAL#6.5536e+4);
```



Conversion is performed using the following procedure.

1 The fractional part of the value of *In* is rounded off to the closest integer as described below.

2 The resulting integer is taken as an unsigned integer and output as a bit string.

The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 –1.49 → –1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 –1.50 → –2 –2.50 → –2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 –1.51 → –2

The following table gives some conversion examples.

Value of *In*	Integer	Value of *Out*
1.6	2	16#0002
3.5	4	16#0004
–1.6	–2	16#FFFE

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
REAL	BYTE	–1.285999e+2 to 1.274999e+2	16#00 to 16#FF
	WORD	–3.276859e+4 to 3.276749e+4	16#0000 to 16#FFFF
	DWORD	–2.147483e+9 to 2.147483e+9	16#0000_0000 to 16#FFFF_FFFF
	LWORD	–9.223372e+18 to 9.223372e+18	16#0000_0000_0000_0000 to 16#FFFF_FFEE_FFFF_FFFF
LREAL	BYTE	–1.28599999999999e+2 to 1.27499999999999e+2	16#00 to 16#FF
	WORD	–3.27685999999999e+4 to 3.27674999999999e+4	16#0000 to 16#FFFF
	DWORD	–2.14748364859999e+9 to 2.14748364749999e+9	16#0000_0000 to 16#FFFF_FFFF
	LWORD	–9.22337203685477e+18 to 9.22337203685477e+18	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF

Additional Information

To convert a bit string to a real number, use a **_TO_*** (Bit String-to-Real Number Conversion Group) instruction (page 2-244).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an illegal value.

_TO_* (Real Number-to-Real Number Conversion Group)

These instructions convert real numbers to real numbers with different data types.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Real Number-to-Real Number Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "***" and "****" must be different real number data types.	Out:=**_TO_*** (In); "***" and "****" must be different real number data types.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

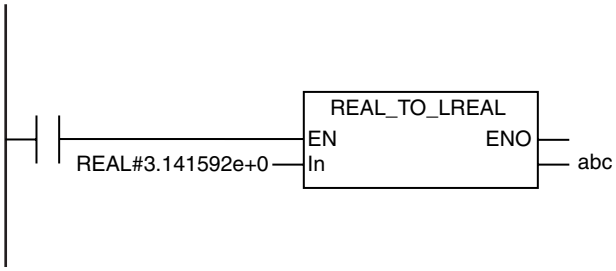
Function

These instructions convert a real number, *In*, to a real number with a different data type.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is REAL data and *Out* is LREAL data, the name of the instruction is REAL_TO_LREAL.

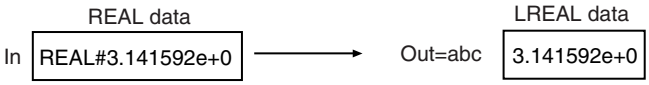
The following example for the REAL_TO_LREAL instruction is for when *In* is REAL#3.141592e+0.

LD



ST

```
abc:=REAL_TO_LREAL(REAL#3.141592e+0);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
REAL	LREAL	–3.402823e+38 to 3.402823e+38
LREAL	REAL	or +∞/–∞

Additional Information

To convert data with any data type to a real number, use a TO_** (Real Number Conversion Group) instruction (page 2-281).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the value of *In* is positive or negative infinity, the value of *Out* is positive or negative infinity.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the conversion result exceeds the valid range of *Out*, the value of *Out* will be infinity with the same sign as the value of *In*.
- For the LREAL_TO_REAL instruction, if the value of *In* is closer to 0 than ±1.175494e–38, the value of *Out* will be 0.

**_TO_STRING (Integer-to-Text String Conversion Group)

These instructions convert integers to text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
**_TO_STRING	Integer-to-Text String Conversion Group	FUN	(@)**_TO_STRING EN ENO In Out "**" must be an integer data type.	Out:=**_TO_STRING(In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid range depends on the data type of *In*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out																				OK

Function

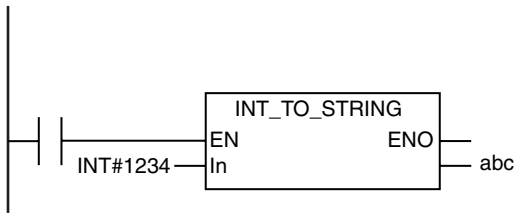
These instructions convert an integer, *In*, to a text string. The number given in *In* is output to conversion result *Out* as a text string. A NULL character (16#00) is placed at the end of *Out*.

The text in *Out* is left-aligned. If the value in *In* requires fewer digits than provided by the data type of *In*, zeros will not be output to the upper digits of *Out*. In other words, leading zeros are suppressed. If *In* contains a negative value, a minus sign (–) is added to the front of the text string.

The name of the instruction is determined by the data type of *In*. For example, if *In* is the INT data type, the instruction is INT_TO_STRING.

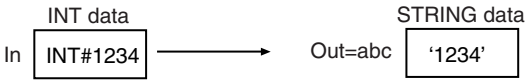
The following example for the INT_TO_STRING instruction is for when *In* is INT#1234.

LD



ST

```
abc:=INT_TO_STRING(INT#1234);
```



The valid range of *Out* depends on the data type of *In* as shown below:

Data type of *In*	Valid range of *Out* (maximum number of bytes)
USINT	4 bytes (three single-byte alphanumeric characters plus the final NULL character)
UINT	6 bytes (five single-byte alphanumeric characters plus the final NULL character)
UDINT	11 bytes (10 single-byte alphanumeric characters plus the final NULL character)
ULINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)
SINT	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
INT	7 bytes (six single-byte alphanumeric characters plus the final NULL character)
DINT	12 bytes (11 single-byte alphanumeric characters plus the final NULL character)
LINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)

Additional Information

To convert a text string number to an integer, use a STRING_TO_** (Text String-to-Integer Conversion Group) instruction (page 2-270).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *In*.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The number of bytes in the conversion result exceeds the size of the output parameter that is connected to *Out*.

**_TO_STRING (Bit String-to-Text String Conversion Group)

These instructions convert bit strings to text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
**_TO_STRING	Bit String-to-Text String Conversion Group	FUN	(@)**_TO_STRING EN ENO In Out "**" must be a bit string data type.	Out:=**_TO_STRING(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid range depends on the data type of *In*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out																				OK

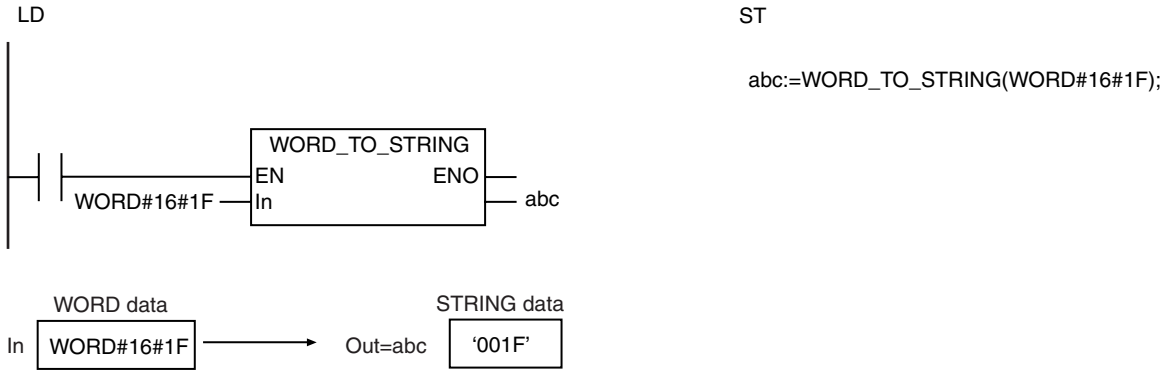
Function

These instructions convert a bit string, *In*, to a text string. The hexadecimal number given in *In* is output to conversion result *Out* as a text string. The #16 prefix of the hexadecimal number is not output to *Out*. A NULL character (16#00) is placed at the end of *Out*.

The text in *Out* is left-aligned. If the value in *In* requires fewer digits than provided by the data type of *In*, the upper digits of *Out* will contain 0. In other words, the unused digits are padded with zeros. The number of bytes in *Out* (including the NULL character) will always be one greater than twice the number of bytes in *In*.

The name of the instruction is determined by the data type of *In*. For example, if *In* is the WORD data type, the instruction is WORD_TO_STRING.

The following example for the WORD_TO_STRING instruction is for when *In* is WORD#16#1F.



The valid range of *Out* depends on the data type of *In* as shown below:

Data type of *In*	Valid range of *Out* (maximum number of bytes)
BYTE	3 bytes (two single-byte alphanumeric characters plus the final NULL character)
WORD	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
DWORD	9 bytes (eight single-byte alphanumeric characters plus the final NULL character)
LWORD	17 bytes (16 single-byte alphanumeric characters plus the final NULL character)

Additional Information

To convert *In* to a signed text string, first convert it to a signed integer using a **_TO_*** (Bit String-to-Integer Conversion Group) instruction (page 2-239) and then use a **_TO_STRING (Integer-to-Text String Conversion Group) instruction (page 2-253).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *In*.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The number of bytes in the conversion result exceeds the size of the output parameter that is connected to *Out*.

**_TO_STRING (Real Number-to-Text String Conversion Group)

These instructions convert real numbers to text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
**_TO_STRING	Real Number-to-Text String Conversion Group	FUN	(@)**_TO_STRING EN ENO In Out "**" must be a real number data type.	Out:=**_TO_STRING(In); "**" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0.0
Out	Conversion result	Output	Conversion result	*	---	---

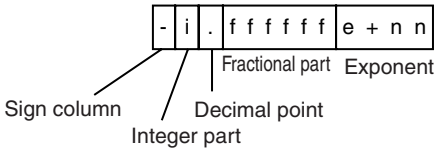
* The valid range depends on the data type of *In*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out																				OK

Function

These instructions convert a real number, *In*, to a text string. *In* is expressed as an alphanumeric text string and output to conversion result *Out*.

The format of *Out* is as follows:



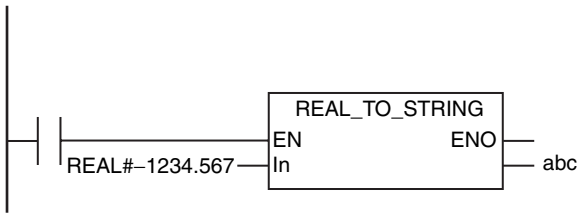
Item	Description
Sign column	If *In* contains a negative value, a minus sign (–) is added. If *In* contains a positive value, a plus sign (+) is not added.
Integer part	The integer part is always only one digit.
Decimal point	The decimal point is always given even if *In* is not a decimal number.
Fractional part	If *In* is REAL data, 6 digits are given. If *In* is LREAL data, 14 digits are given.
Exponent	The exponent is always given. “nn” is 2 or 3 digits. The sign of “nn” is positive (+) if the absolute value of *In* is 1.0 or higher and negative (–) if it is less than 1.0.

A NULL character (16#00) is placed at the end of *Out*.

The name of the instruction is determined by the data type of *In*. For example, if *In* is the REAL data type, the instruction is REAL_TO_STRING.

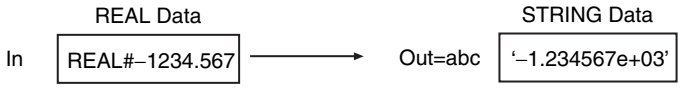
The following example shows the REAL_TO_STRING instruction when *In* is REAL#–1234.567.

LD



ST

```
abc:=REAL_TO_STRING(REAL#–1234.567);
```



If the value of *In* is 0, infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
0	0
+∞	inf
–∞	–inf
Nonnumeric data	‘nan’ or ‘–nan’

Additional Information

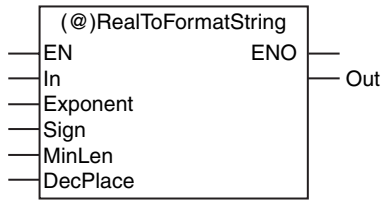
- To convert a text string to a real number, use a STRING_TO_** (Text String-to-Real Number Conversion Group) instruction (page 2-274).
- To specify the format when you convert a real number to a text string, use the RealToFormatString instruction (page 2-259) or the LrealToFormatString instruction (page 2-264).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *In*.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The number of bytes in the conversion result exceeds the size of the output parameter that is connected to *Out*.

RealToFormatString

The RealToFormatString instruction converts a REAL variable to a text string with the specified format.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RealToFormat-String	REAL-to-Format-ted Text String	FUN		Out:=RealToFormat-String(In, Exponent, Sign, MinLen, DecPlace);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0.0
Exponent	Exponent		TRUE: Exponent FALSE: No exponent			FALSE
Sign	Sign column		TRUE: Sign column FALSE: No sign column			
MinLen	Minimum number of digits		Minimum number of digits in *Out*			6
DecPlace	Precision		Number of decimal digits in *Out*	0 to 15		
Out	Conversion result	Output	Conversion result	327 bytes max. (326 single-byte alphanumeric characters plus the final NULL character)	---	---

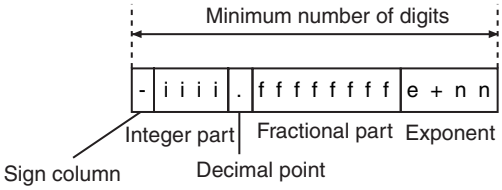
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK						
Exponent	OK																			
Sign	OK																			
MinLen						OK														
DecPlace						OK														
Out																				OK

Function

The RealToFormatString instruction converts REAL variable *In* to a text string. *In* is expressed as an alphanumeric text string and output to conversion result *Out.* A NULL character (16#00) is placed at the end of *Out*.

If *In* contains a negative value, a minus sign (–) is added to the front of the text string. If *In* contains a positive value, a plus sign (+) is not added to the front of the text string.

The format of *Out* is determined by exponent *Exponent*, sign column *Sign*, minimum number of digits *MinLen*, and precision *DecPlace*.



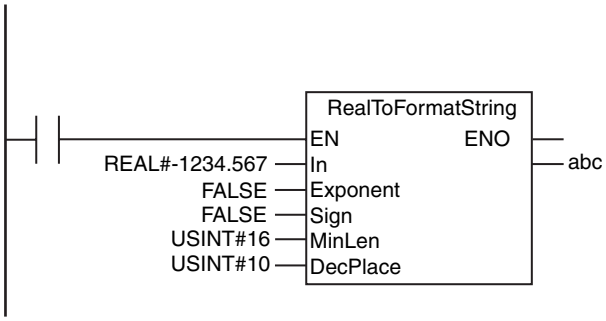
Input variable	Description
Exponent	*Exp* specifies whether an exponent is given. TRUE: Exponent FALSE: No exponent
Sign	*Sign* specifies whether there is a sign column. TRUE: Sign column FALSE: No sign column The sign column is used only for a minus sign (–). If the number is positive when the sign column is specified, the sign column will contain a blank character. If the number is negative when no sign column is specified, a minus sign (–) will be added to the front of the integer part. However, if the number of digits in the conversion result exceeds the value of *MinLen* and the conversion result is positive, the highest digit is placed in the sign column.
MinLen	*MinLen* is the minimum number of total digits for the sign column, integer part, decimal point, fractional part, and exponent. If the conversion result has fewer digits than the value of *MinLen*, the text string will be right-aligned (except for the sign column) and remaining digits will contain blank characters. If the number of digits in the conversion result exceeds the value of *MinLen*, the text string is left-aligned and the text string for the digits that exceed the value of *MinLen* is assigned to *Out*.
DecPlace	*DecPlace* is the number of digits in the fractional part. If the number of digits exceeds the value of *DecPlace*, the extra digits in the fractional portion are rounded off as described below. If the value of *DecPlace* is 0, the fractional part and decimal point are not given.

The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is REAL#–1234.567.

Example 1: Exponent: FALSE
Sign: FALSE
MinLen: USINT#16
DecPlace: USINT#10

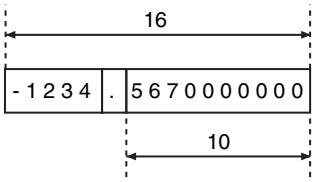
Here, no sign column is specified for a negative number, so a minus sign (–) is added to the front of the integer part.

LD



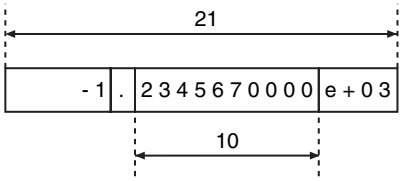
ST

```
abc:=RealToFormatString(REAL#-1234.567, FALSE,
                        FALSE, USINT#16,
                        USINT#10);
```



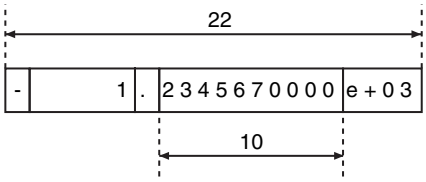
Example 2: Exponent: TRUE
Sign: FALSE
MinLen: USINT#21
DecPlace: USINT#10

Here, the value of *MinLen* exceeds the number of digits in the text string, so the text string is right-aligned and blank characters are added before it.



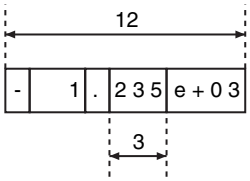
Example 3: Exponent: TRUE
Sign: TRUE
MinLen: USINT#22
DecPlace: USINT#10

The sign column is always on the left. Blank characters are added to the front of the integer part.



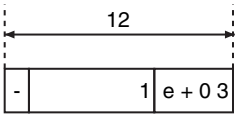
Example 4: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#3

The fourth decimal place is rounded off because *DecPlace* is USINT#3.



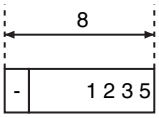
Example 5: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#0

The first decimal place is rounded off because *DecPlace* is USINT#0. The decimal point is also not given.



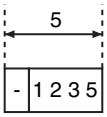
Example 6: Exponent: FALSE
Sign: TRUE
MinLen: USINT#8
DecPlace: USINT#0

Here, no exponent is given and the integer part is only four digits. The first decimal place is rounded off.



Example 7: Exponent: FALSE
Sign: TRUE
MinLen: USINT#2
DecPlace: USINT#0

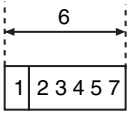
Here, the number of digits in the integer part of *In* (four digits) is larger than the value of *MinLen* (USINT#2). The four digits of the integer part are given.



The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is REAL#123456.7.

Example 8: Exponent: FALSE
Sign: TRUE
MinLen: USINT#4
DecPlace: USINT#0

Here, the number of digits in the integer part of *In* (six digits) is larger than the value of *MinLen* (USINT#4). The six digits of the integer part are given. The value of *In* is positive, so the highest digit is placed in the sign column.



If the value of *In* is positive infinity, the value of *Out* is 'Inf'. If the value of *In* is negative infinity, the value of *Out* is '–Inf'. If the value of *In* is nonnumeric data, the value of Out is "–nan".

If the value of *In* is infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
+∞	'inf'
−∞	'–inf'
Nonnumeric data	'nan' or '–nan'

The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

Additional Information

- *Exponent, Sign, MinLen,* and *DecPlace* can be omitted. The defaults are applied for any omitted input variables.
- To convert a LREAL variable to a text string, use the LrealToFormatString instruction (page 2-264).
- To convert a text string to a real number, use a STRING_TO_** (Text String-to-Real Number Conversion Group) instruction (page 2-274).

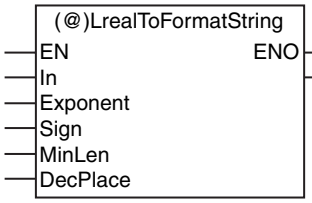
Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *DecPlace* is outside of the valid range.
- The value of *DecPlace* is greater than the value of *MinLen.*
- The number of bytes in the conversion result exceeds the size of the output parameter that is connected to *Out*.

LrealToFormatString

The LrealToFormatString instruction converts a LREAL variable to a text string with the specified format.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LrealToFormat-String	LREAL-to-Format-ted Text String	FUN		Out:=LrealToFormat-String(In, Exponent, Sign, MinLen, DecPlace);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0.0
Exponent	Exponent		TRUE: Exponent FALSE: No exponent			FALSE
Sign	Sign column		TRUE: Sign column FALSE: No sign column			
MinLen	Minimum number of digits		Minimum number of digits in *Out*			6
DecPlace	Precision		Number of decimal digits in *Out*	0 to 15		
Out	Conversion result	Output	Conversion result	327 bytes max. (326 single-byte alphanumeric characters plus the final NULL character)	---	---

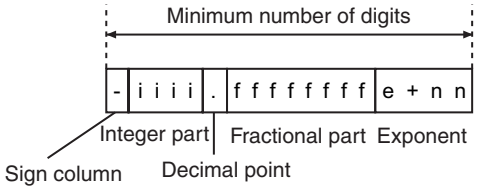
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In															OK					
Exponent	OK																			
Sign	OK																			
MinLen						OK														
DecPlace						OK														
Out																				OK

Function

The LrealToFormatString instruction converts LREAL variable *In* to a text string. *In* is expressed as an alphanumeric text string and output to conversion result *Out.* A NULL character (16#00) is placed at the end of *Out*.

If *In* contains a negative value, a minus sign (–) is added to the front of the text string. If *In* contains a positive value, a plus sign (+) is not added to the front of the text string.

The format of *Out* is determined by exponent *Exponent*, sign column *Sign*, minimum number of digits *MinLen*, and precision *DecPlace*.



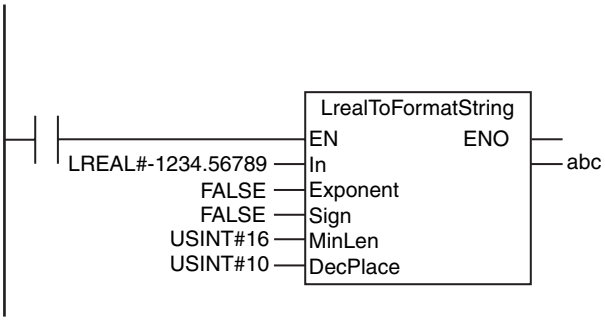
Input variable	Description
Exponent	*Exp* specifies whether an exponent is given. TRUE: Exponent FALSE: No exponent
Sign	*Sign* specifies whether there is a sign column. TRUE: Sign column FALSE: No sign column The sign column is used only for a minus sign (–). If the number is positive when the sign column is specified, the sign column will contain a blank character. If the number is negative when no sign column is specified, a minus sign (–) will be added to the front of the integer part. However, if the number of digits in the conversion result exceeds the value of *MinLen* and the conversion result is positive, the highest digit is placed in the sign column.
MinLen	*MinLen* is the minimum number of total digits for the sign column, integer part, decimal point, fractional part, and exponent. If the conversion result has fewer digits than the value of *MinLen*, the text string will be right-aligned (except for the sign column) and remaining digits will contain blank characters. If the number of digits in the conversion result exceeds the value of *MinLen*, the text string is left-aligned and the text string for the digits that exceed the value of *MinLen* is assigned to *Out*.
DecPlace	*DecPlace* is the number of digits in the fractional part. If the number of digits exceeds the value of *DecPlace*, the extra digits in the fractional portion are rounded off as described below. If the value of *DecPlace* is 0, the fractional part and decimal point are not given.

The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is LREAL#–1234.56789.

Example 1: Exponent: FALSE
Sign: FALSE
MinLen: USINT#16
DecPlace: USINT#10

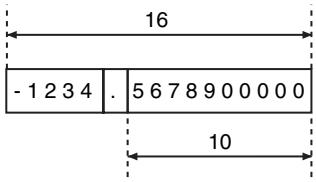
Here, no sign column is specified for a negative number, so a minus sign (–) is added to the front of the integer part.

LD



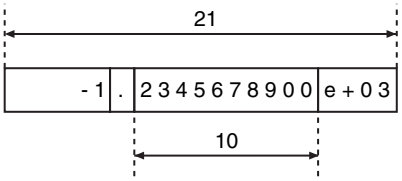
ST

```
abc:=LrealToFormatString(LREAL#-1234.56789, FALSE,
                         FALSE, USINT#16,
                         USINT#10);
```



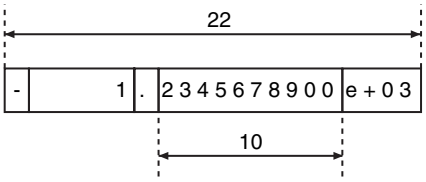
Example 2: Exponent: TRUE
Sign: FALSE
MinLen: USINT#21
DecPlace: USINT#10

Here, the value of *MinLen* exceeds the number of digits in the text string, so the text string is right-aligned and blank characters are added before it.



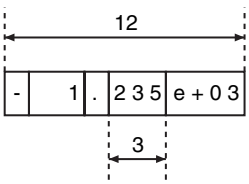
Example 3: Exponent: TRUE
Sign: TRUE
MinLen: USINT#22
DecPlace: USINT#10

The sign column is always on the left. Blank characters are added to the front of the integer part.



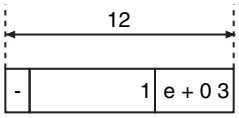
Example 4: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#3

The fourth decimal place is rounded off because *DecPlace* is USINT#3.



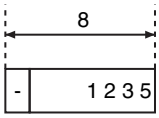
Example 5: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#0

The first decimal place is rounded off because *DecPlace* is USINT#0. The decimal point is also not given.



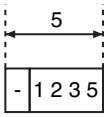
Example 6: Exponent: FALSE
Sign: TRUE
MinLen: USINT#8
DecPlace: USINT#0

Here, no exponent is given and the integer part is only four digits. The first decimal place is rounded off.



Example 7: Exponent: FALSE
Sign: TRUE
MinLen: USINT#2
DecPlace: USINT#0

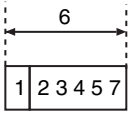
Here, the number of digits in the integer part of *In* (four digits) is larger than the value of *MinLen* (USINT#2). The four digits of the integer part are given.



The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is LREAL#123456.789.

Example 8: Exponent: FALSE
Sign: TRUE
MinLen: USINT#4
DecPlace: USINT#0

Here, the number of digits in the integer part of *In* (six digits) is larger than the value of *MinLen* (USINT#4). The six digits of the integer part are given. The value of *In* is positive, so the highest digit is placed in the sign column.



If the value of *In* is positive infinity, the value of *Out* is 'Inf'. If the value of *In* is negative infinity, the value of *Out* is '–Inf'. If the value of *In* is nonnumeric data, the value of Out is "–nan".

If the value of In is infinity, or nonnumeric data, the value of Out is as shown below.

Value of *In*	Value of *Out*
+∞	'inf'
−∞	'–inf'
Nonnumeric data	'nan' or '–nan'

The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

Additional Information

- *Exponent, Sign, MinLen,* and *DecPlace* can be omitted. The defaults are applied for any omitted input variables.
- To convert a REAL variable to a text string, use the RealToFormatString instruction (page 2-259).
- To convert a text string to a real number, use a STRING_TO_** (Text String-to-Real Number Conversion Group) instruction (page 2-274).

Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *DecPlace* is outside of the valid range.
- The value of *DecPlace* is greater than the value of *MinLen.*
- The number of bytes in the conversion result exceeds the size of the output parameter that is connected to *Out*.

STRING_TO_** (Text String-to-Integer Conversion Group)

These instructions convert text strings to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
STRING_TO_**	Text String-to-Integer Conversion Group	FUN	(@)STRING_TO_** EN ENO In Out "**" must be an integer data type.	Out:=STRING_TO_** (In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	"
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* The valid range depends on the data type of *Out*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out						OK	OK	OK	OK	OK	OK	OK	OK							

Function

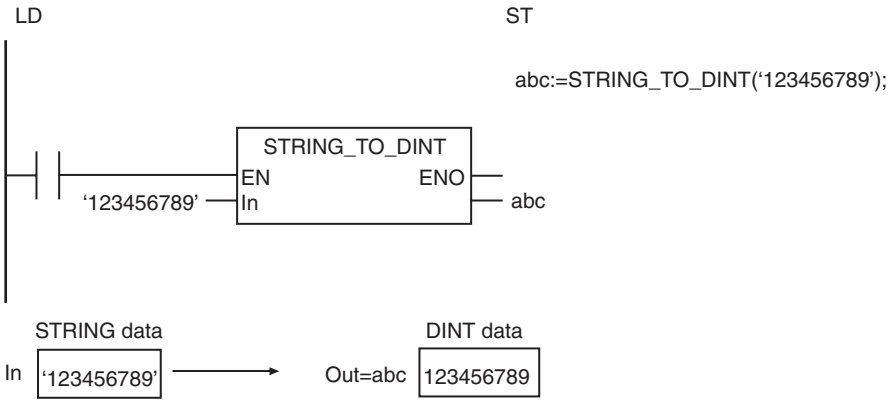
These instructions convert a text string, *In*, to an integer.

Basically, the text string in *In* must consist only of numbers 0 to 9. The following exceptions are possible.

- If the first character in *In* is a single minus sign (–) or a single plus sign (+), it is processed as the sign.
- Any blank characters at the beginning of *In* are ignored.
- Any blank characters between an initial minus sign (–) or plus sign (+) and a number are ignored.
- Any single underbars ('_') at any location are ignored.
- An error occurs if there are two or more consecutive underbars ('_') at any location.
- An error occurs if there are any underbars ('_') at the beginning or end.
- An error occurs if there are any underbars ('_') between the minus signs ('–') or plus sign ('+') and the number at the beginning.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the DINT data type, the instruction is STRING_TO_DINT.

The following example for the STRING_TO_DINT instruction is for when *In* is '123456789'.



The valid range of *In* depends on the data type of *Out* as shown below:

Data type of *Out*	Valid range of *In* (maximum number of bytes)*
USINT	4 bytes (three single-byte alphanumeric characters plus the final NULL character)
UINT	6 bytes (five single-byte alphanumeric characters plus the final NULL character)
UDINT	11 bytes (10 single-byte alphanumeric characters plus the final NULL character)
ULINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)
SINT	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
INT	7 bytes (six single-byte alphanumeric characters plus the final NULL character)
DINT	12 bytes (11 single-byte alphanumeric characters plus the final NULL character)
LINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)

* Any blank characters (' ') at the beginning of the text string, any zeros at the beginning of the text string, and any underbars ('_') in the text string are not included in the number of bytes.

Additional Information

- To convert a text string to a hexadecimal number, use a STRING_TO_** (Text String-to-Bit String Conversion Group) instruction (page 2-272).
- To convert an integer to a text string, use a **_TO_STRING (Integer-to-Text String Conversion Group) instruction (page 2-253).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the value of *In* is '–0', the value of *Out* is 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The text string in *In* does not express a number.
  - The conversion result exceeds the valid range of the data type of *Out.*
  - The text string in *In* does not end in a NULL character.

STRING_TO_** (Text String-to-Bit String Conversion Group)

These instructions convert text strings to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
STRING_TO_**	Text String-to-Bit String Conversion Group	FUN	(@)STRING_TO_** EN ENO In Out "**" must be a bit string data type.	Out:=STRING_TO_** (In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	"
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* The valid range depends on the data type of *Out*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out		OK	OK	OK	OK															

Function

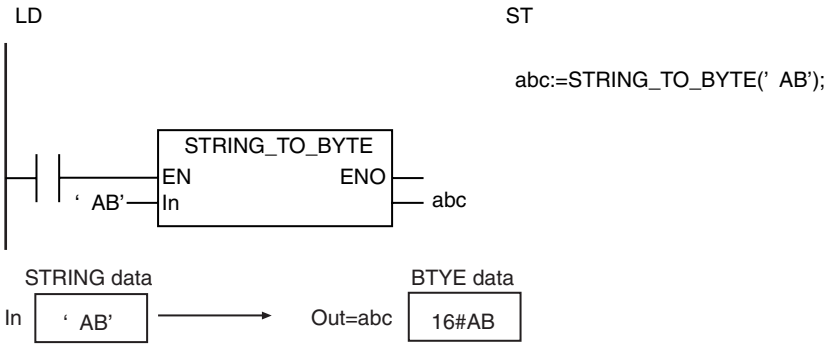
These instructions interpret the content of a text string, *In*, as a hexadecimal number and convert it to a bit string.

Basically, the text string in *In* must consist only of '0' to '9', 'a' to 'f', and 'A' to 'F'. The following exception is possible.

- Any continuous blank characters or zeros at the beginning of *In* are ignored.
- Any single underbars ('_') at any location are ignored.
- An error occurs if there are two or more consecutive underbars ('_') at any location.
- An error occurs if there are any underbars ('_') at the beginning or end.
- An error occurs if there are any underbars ('_') between the minus signs ('–') or plus sign ('+') and the number at the beginning.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the BYTE data type, the instruction is STRING_TO_BYTE.

The following example for the STRING_TO_BYTE instruction is for when *In* is ‘ AB’. Any blank characters at the beginning are ignored.



The valid range of *In* depends on the data type of *Out* as shown below:

Data type of *Out*	Valid range of *In* (maximum number of bytes)*
BYTE	3 bytes (two single-byte alphanumeric characters plus the final NULL character)
WORD	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
DWORD	9 bytes (eight single-byte alphanumeric characters plus the final NULL character)
LWORD	17 bytes (16 single-byte alphanumeric characters plus the final NULL character)

* Any blank characters (‘ ’) at the beginning of the text string, any zeros at the beginning of the text string, and any underbars (‘_’) in the text string are not included in the number of bytes.

Additional Information

- To treat a signed number as a text string, use a STRING_TO_** (Text String-to-Integer Conversion Group) instruction (page 2-270).
- To convert a bit string to a text string, use a **_TO_STRING (Bit String-to-Text String Conversion Group) instruction (page 2-255).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The text string in *In* does not express a number.
  - The conversion result exceeds the valid range of the data type of *Out.*
  - The text string in *In* does not end in a NULL character.

STRING_TO_** (Text String-to-Real Number Conversion Group)

These instructions convert text strings to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
STRING_TO_**	Text String-to-Real Number Conversion Group	FUN	(@)STRING_TO_** EN ENO In Out "**" must be a real number data type.	Out:=STRING_TO_** (In); "**" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	311 bytes max. (310 single-byte alphanumeric characters plus the final NULL character)	---	"
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

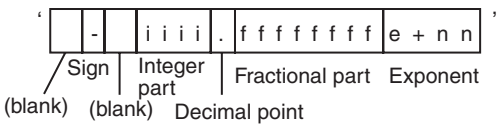
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out														OK	OK					

Function

These instructions convert a text string, *In*, to a real number.

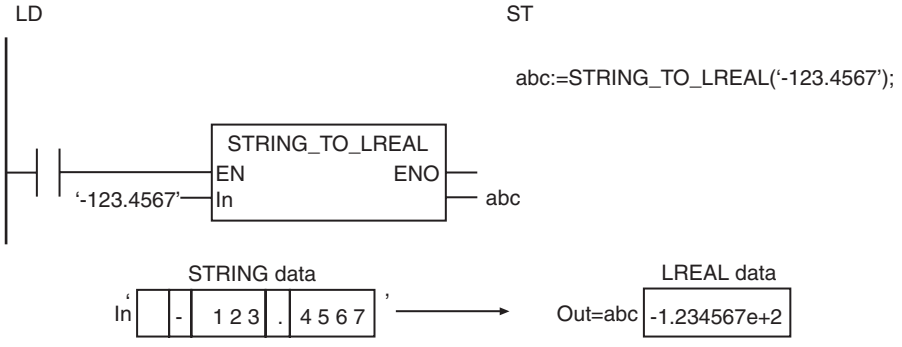
The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the LREAL data type, the instruction is STRING_TO_LREAL.

The format of the text sting in *In* is given below.

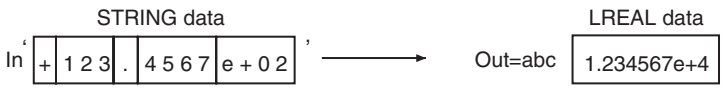


Name	Format
Sign	• Any consecutive blank characters at the beginning of the text string are ignored. Any following single plus or minus sign is treated as the sign. • The plus sign can be omitted. • Any consecutive blank characters after the sign are ignored.
Integer part	• The characters after the sign and up to the decimal point are taken as the integer part. Any consecutive blank characters after the sign are not included in the integer part. The sign may sometimes be omitted. • If the decimal point and fractional part are omitted, the characters up to the exponent are taken as the integer part. • If the decimal point, fractional part, and exponent are omitted, the characters up to the end of the text string are taken as the integer part. • The integer part consists of '0' to '9'. • The integer part cannot be omitted. • The maximum number of digits in the integer part is the maximum text string length of 1985 minus the total number of bytes in the following: the sign, decimal point, fractional part, exponent, and blank characters before and after the sign.
Decimal point	• A single period ('.') following the integer part is taken as the decimal point. • Omit the decimal point if there is no fractional part.
Fractional part	• The characters after the decimal point and up to the exponent are taken as the fractional part. • If the exponent is omitted, the characters up to the end of the text string are taken as the fractional part. • The fractional part consists of '0' to '9'. • The fractional part can be omitted. • The fractional part can consist of a maximum of 15 digits. • If there is no decimal point, then there is no fractional part.
Exponent	• The exponent consists of a single 'e' or 'E' after the fractional part, a following single plus or minus sign, and the remaining characters to the end of the text string. • If there is no fractional part, then the above text string after the decimal point is taken as the exponent. • If there is no decimal point or fractional part, then the above text string after the integer part is taken as the exponent. • The numeric part of the exponent consists of '0' to '9'. • The exponent can be omitted. • The numeric part of the exponent can consist of a maximum of three digits.

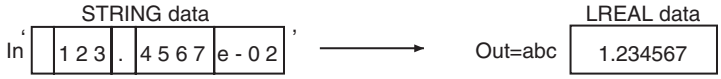
Example 1: The following example uses the sign, decimal point, and fractional part, but does not use an exponent.



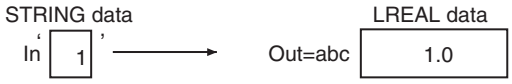
Example 2: The following example uses the sign, decimal point, fractional part, and exponent.



Example 3: The following example does not use the sign, but uses the decimal point, fractional part, and exponent.



Example 4: The following example does not use the sign, fractional part, decimal point, and exponent.



If the value of *In* is '+Inf', the value of *Out* is positive infinity. If the value of *In* is '–Inf', the value of *Out* is negative infinity. In either case, characters are not case sensitive.

Additional Information

To convert a real number to a text string, use a **_TO_STRING (Real Number-to-Text String Conversion Group) instruction (page 2-257).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- Any single underbars ('_') at any location in In are ignored.
- An error occurs if there are any underbars ('_') at the beginning or end of *In*.
- An error occurs if there are two or more consecutive underbars ('_') at any location in *In*.
- An error occurs if there are any underbars ('_') between the minus signs ('–') or plus sign ('+') and the number at the beginning of *In*.
- If the content of *In* exceeds the precision of the data type of *Out*, the value is rounded.
- If the content of *In* is closer to 0 than the minimum value of the data type of *Out*, the value of *Out* will be 0.
- If the content of *In* exceeds the valid range of *Out*, *Out* will be positive infinity for a positive number or negative infinity for a negative number.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The text string in *In* does not express a number.
  - The text string in *In* does not end in a NULL character.
  - The text string in *In* has a decimal point but not a fractional part.

TO_** (Integer Conversion Group)

These instructions convert integers, bit strings, real numbers, and text strings to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TO_**	Integer Conversion Group	FUN	(@)TO_** EN ENO In Out "**" must be an integer data type.	Out:=TO_** (In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*.

*2 If you omit the input parameter, the default value is not applied. A building error will occur.

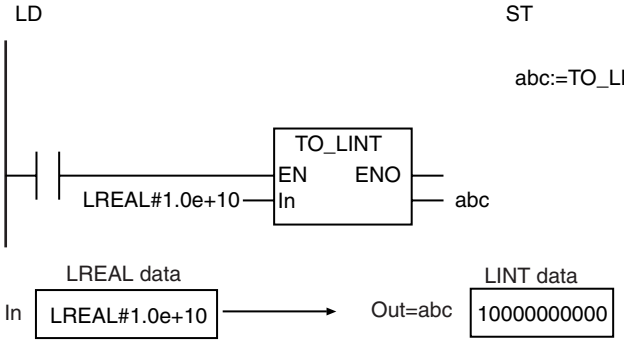
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out						OK	OK	OK	OK	OK	OK	OK	OK							

Function

These instructions convert the integer, bit string, real number, or text string in *In* to an integer.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the LINT data type, the instruction is TO_LINT.

The following example for the TO_LINT instruction is for when *In* is LREAL#1.0e+10.



- Conversion is performed to within the effective digits of the data type of *In.* If *In* is a real number, the fractional part is rounded off to the closest integer. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

The valid ranges for *In* and *Out* depend on their data types. Refer to the descriptions of the functions of the following instructions for the valid ranges: **_TO_*** (Integer-to-Integer Conversion Group) (page 2-232), **_TO_*** (Bit String-to-Integer Conversion Group) (page 2-239), and **_TO_*** (Real Number-to-Integer Conversion Group) (page 2-246).

For detailed specifications when *In* is STRING data, refer to Function for the STRING_TO_** (Text String-to-Integer Conversion Group) instructions (page 2-270).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the data type of *In* is for a bit string and the sizes of the data types of *In* and *Out* are different, the following processing is performed.
  - If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
  - If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.
- Observe the following precautions if *In* is STRING data.
  - If the first character in *In* is a minus sign (–) or a plus sign (+), it is processed as the sign.
  - Except for a minus sign (–) or a plus sign (+) at the beginning, *In* must consist of consecutive ‘0’ to ‘9’ characters. Underbars (‘_’) and blank characters before or after the ‘–’ or ‘+’ are allowed in the text string.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an illegal value.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* is STRING data, but the text sting in *In* does not express a number.
  - *In* is STRING data, but it does not end in a NULL character.

TO_** (Bit String Conversion Group)

These instructions convert integers, bit strings, real numbers, and text strings to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TO_**	Bit String Conversion Group	FUN	(@)TO_** EN ENO In Out "**" must be a bit string data type.	Out:=TO_**(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*.

*2 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out		OK	OK	OK	OK															

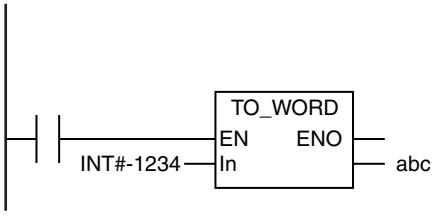
Function

These instructions convert the integer, bit string, real number, or text string in *In* to a bit string.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the WORD data type, the instruction is TO_WORD.

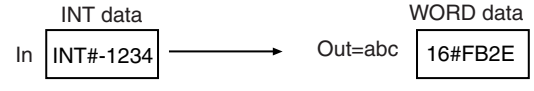
The following example for the TO_WORD instruction is for when *In* is INT#−1234.

LD



ST

```
abc:=TO_WORD(INT#-1234);
```



The valid ranges for *In* and *Out* depend on their data types. Refer to the descriptions of the functions of the following instructions for the valid ranges: **_TO_*** (Integer-to-Bit String Conversion Group) (page 2-235), **_TO_*** (Bit String-to-Bit String Conversion Group) (page 2-242), and **_TO_*** (Real Number-to-Bit String Conversion Group) (page 2-249).

For detailed specifications when *In* is STRING data, refer to Function for the STRING_TO_** (Text String-to-Bit String Conversion Group) instructions (page 2-272).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an illegal value.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* is STRING data, but the text sting in *In* does not express a number.
  - *In* is STRING data, but it does not end in a NULL character.

TO_** (Real Number Conversion Group)

These instructions convert integers, bit strings, real numbers, and text strings to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TO_**	Real Number Conversion Group	FUN	(@)TO_** EN ENO In Out "**" must be a real number data type.	Out:=TO_**(In); "**" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1, *2	---	*3
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*.

*2 For STRING data, the valid range is 311 bytes max. (310 single-byte alphanumeric characters plus the final NULL character).

*3 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out														OK	OK					

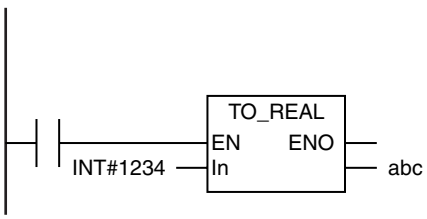
Function

These instructions convert the integer, bit string, real number, or text string in *In* to a real number.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the REAL data type, the instruction is TO_REAL. If the value of *In* is positive or negative infinity, the value of *Out* is positive or negative infinity.

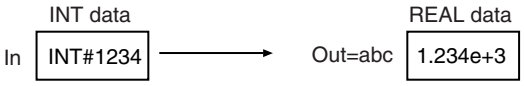
The following example for the TO_REAL instruction is for when *In* is INT#1234.

LD



ST

```
abc:=TO_REAL(INT#1234);
```



The valid ranges for *In* and *Out* depend on their data types. Refer to the descriptions of the functions of the following instructions for the valid ranges: **_TO_*** (Integer-to-Real Number Conversion Group) (page 2-237), **_TO_*** (Bit String-to-Real Number Conversion Group) (page 2-244), and **_TO_*** (Real Number-to-Real Number Conversion Group) (page 2-251).

For detailed specifications when *In* is STRING data, refer to Function for the STRING_TO_** (Text String-to-Real Number Conversion Group) instructions (page 2-274).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* is STRING data, but the text sting in *In* does not express a number.
  - *In* is STRING data, but it does not end in a NULL character.

TRUNC, Round, and RoundUp

These instructions change real numbers to integers.

TRUNC:	Truncates the number at the first decimal digit.
Round:	Rounds the number at the first decimal digit.
RoundUp:	Rounds up the number at the first decimal digit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TRUNC	Truncate	FUN	(@)TRUNC EN ENO In Out	Out:=TRUNC(In);
Round	Round Off Real Number	FUN	(@)Round EN ENO In Out	Out:=Round(In);
RoundUp	Round Up Real Number	FUN	(@)RoundUp EN ENO In Out	Out:=RoundUp(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out												OK	OK							

Function

These instructions change the real number in *In* to an integer by eliminating the fractional part.

● TRUNC

The TRUNC instruction truncates the number at the first decimal digit.

● Round

The Round instruction rounds the number at the first decimal digit. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 –1.49 → –1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 –1.50 → –2 –2.50 → –2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 –1.51 → –2

● RoundUp

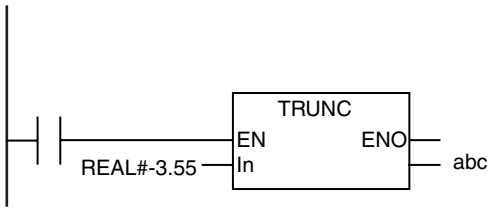
The RoundUp instruction rounds up the number at the first decimal digit.

The differences in these three instructions are shown by the following examples.

Input value	Output value		
	TRUNC	Round	RoundUp
REAL#1.6	DINT#1	DINT#2	DINT#2
REAL#1.5	DINT#1	DINT#2	DINT#2
REAL#1.5	DINT#1	DINT#1	DINT#2
REAL#2.5	DINT#2	DINT#2	DINT#3
REAL#-1.6	DINT#-1	DINT#-2	DINT#-2
REAL#-1.5	DINT#-1	DINT#-2	DINT#-2
REAL#-1.4	DINT#-1	DINT#-1	DINT#-2
REAL#-2.5	DINT#-2	DINT#-2	DINT#-3

The following example for the TRUNC instruction is for when *In* is REAL#–3.55. The value of variable *abc* will be DINT#–3.

LD



ST

```
abc:=TRUNC(REAL#-3.55);
```

Additional Information

If the data type of *In* is REAL, the data type of *Out* is DINT. If the data type of *In* is LREAL, the data type of *Out* is LINT.

Precautions for Correct Use

If the conversion result exceeds the valid range of *Out*, *Out* will contain an illegal value.

Bit String Processing Instructions

Instruction	Name	Page
AND (&), OR, and XOR	Logical AND/Logical OR/ Logical Exclusive OR	2-286
XORN	Logical Exclusive NOR	2-289
NOT	Bit Reversal	2-291
AryAnd, AryOr, AryXor, and AryXorN	Array Logical AND/ Array Logical OR/ Array Logical Exclusive OR/ Array Logical Exclusive NOR	2-293

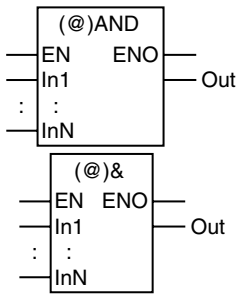
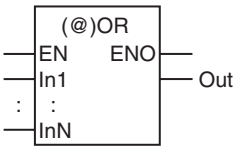
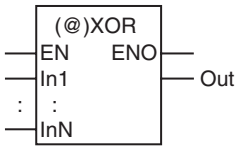
AND (&), OR, and XOR

These instructions perform processing on Boolean variables or individual bits in bit stings.

AND (&): Logical AND

OR: Logical OR

XOR: Logical Exclusive OR

Instruction	Name	FB/FUN	Graphic expression	ST expression
AND (&)	Logical AND	FUN		Out:=In1 AND ··AND InN; Out:=In1 & ··& InN;
OR	Logical OR	FUN		Out:=In1 OR ··OR InN;
XOR	Logical Exclusive OR	FUN		Out:=In1 XOR ··XOR InN;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Data to process	Input	Data to process, where N is 2 to 5	Depends on data type.	---	0*
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	OK	OK	OK	OK	OK															
Out	Must be same data type as *In1* to *InN*																			

Function

These instructions perform processing on Boolean variables or corresponding bits in bit strings. The data to process is in *In1* to *InN*. *In1* to *InN* and *Out* must be the same data types.

If there are more than two data to process, processing is performed with the following procedure.

1 Processing is performed for *In1* and *In2*.

2 Processing is performed for the results of step 1 and *In3*.

3 Processing is performed for the results of step 2 and *In4*.

⋮ ⋮

The relationships between input and output variables are given in the following tables.

● AND (&)

If both bits are TRUE, then the processing result is TRUE. Otherwise, the processing result is FALSE.

***In1* bit**	***In2* bit**	***Out* bit**
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

● OR

If both bits are FALSE, then the processing result is FALSE. Otherwise, the processing result is TRUE.

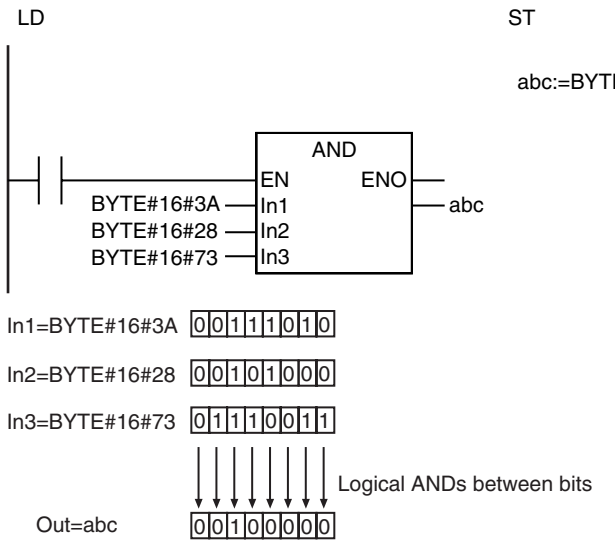
***In1* bit**	***In2* bit**	***Out* bit**
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

● XOR

If both bits are the same, then the processing result is FALSE. If one bit is TRUE and the other is FALSE, then the processing result is TRUE.

***In1* bit**	***In2* bit**	***Out* bit**
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

The following example shows the AND instruction when *In1* is BYTE#16#3A, *In2* is BYTE#16#28 and *In3* is BYTE#16#73.



The functions of the AND instruction and the & instruction are exactly the same. Use the form that is easier to use.

Additional Information

In ST, there is no limit to the number of input variables if you use the following notation.

Out:=In1 AND In2 AND In3 AND In4 AND In5 AND In6 ···

Out:=In1 & In2 & In3 & In4 & In5 & In6 ···

Out:=In1 OR In2 OR In3 OR In4 OR In5 OR In6 ···

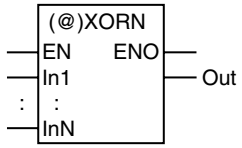
Out:=In1 XOR In2 XOR In3 XOR In4 XOR In5 XOR In6 ···

Precautions for Correct Use

The data types of *In1* to *InN* and *Out* must all be the same. Otherwise, a building error will occur.

XORN

The XORN instruction performs a logical exclusive NOR operation on Boolean variables or individual bits in bit stings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
XORN	Logical Exclusive NOR	FUN		Out:=In1 XOR NOT ·· XOR NOT InN;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Data to process	Input	Data to process, where N is 2 to 5	Depends on data type.	---	0*
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	OK	OK	OK	OK	OK															
Out	Must be same data type as *In1* to *InN*																			

Function

The XORN instruction performs processing on Boolean variables or corresponding bits in bit strings. The data to process is in *In1* to *InN. In1* to *InN* and *Out* must be the same data types.

If there are more than two data to process, processing is performed with the following procedure.

1 Processing is performed for *In1* and *In2*.

2 Processing is performed for the results of step 1 and *In3*.

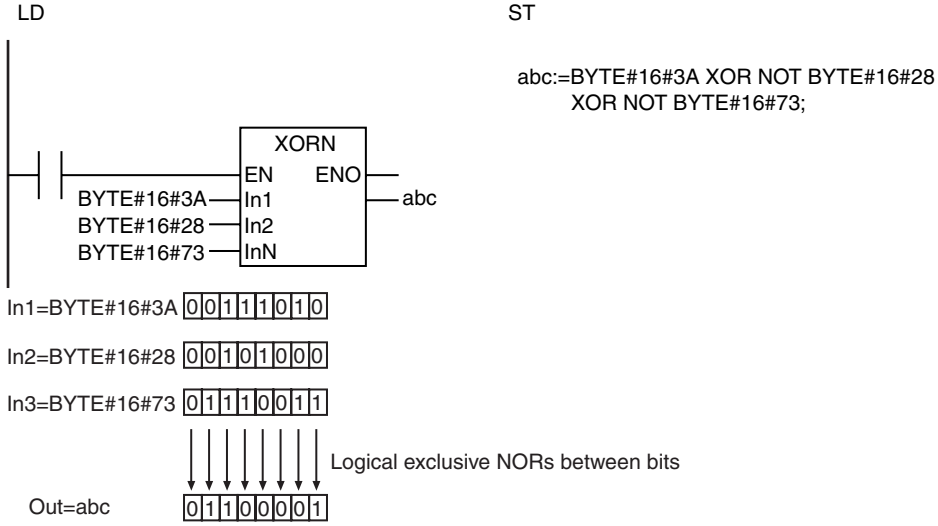
3 Processing is performed for the results of step 2 and *In4*.

⋮ ⋮

The relationships between input and output variables are given in the following table. If both values are the same, then the processing result is TRUE. Otherwise, the processing result is FALSE.

In1 bit	*In2* bit	*Out* bit
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

The following example is for when *In1* is BYTE#16#3A, *In2* is BYTE#16#28, and *In3* is BYTE#16#73.



Precautions for Correct Use

The data types of *In1* to *InN* and *Out* must all be the same. Otherwise, a building error will occur.

NOT

The NOT instruction reverses the value of a Boolean variable or the individual bits in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NOT	Bit Reversal	FUN	(@)NOT EN ENO In Out	Out:=NOT(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to process	Input	Data to process	Depends on data type.	---	*
Out	Processing result	Output	Processing result	Depends on data type.	---	---

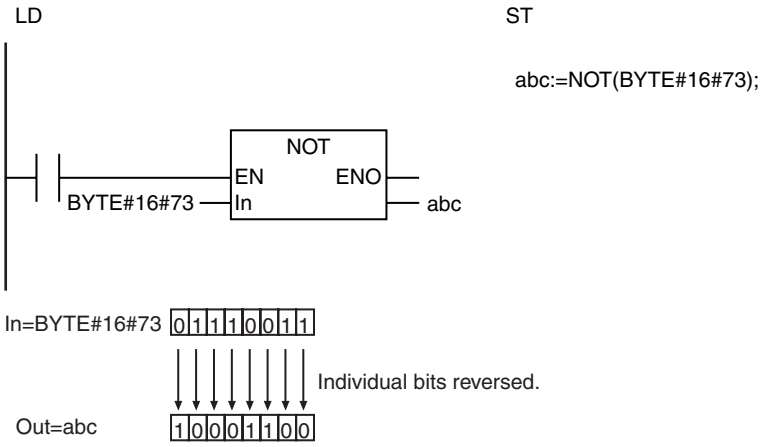
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK															
Out	Must be same data type as *In*																			

Function

The NOT instruction reverses the value of a Boolean variable or the values of individual bits in a bit string. The data to process is in *In*. *In* and processing result *Out* must have the same number of bits, i.e., they must be the same data type.

The following example is for when *In* is BYTE#16#73.



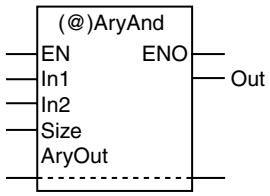
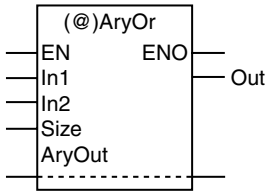
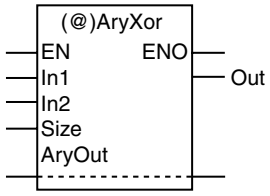
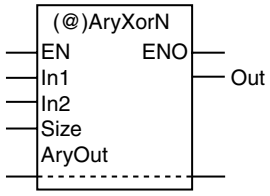
Precautions for Correct Use

The data types of *In* and *Out* must be the same. Otherwise, a building error will occur.

AryAnd, AryOr, AryXor, and AryXorN

These instructions process Boolean variables or individual bits in bit stings between arrays.

AryAnd: Logical AND
AryOr: Logical OR
AryXor: Logical Exclusive OR
AryXorN: Logical Exclusive NOR

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryAnd	Array Logical AND	FUN		AryAnd(In1, In2, Size, AryOut);
AryOr	Array Logical OR	FUN		AryOr(In1, In2, Size, AryOut);
AryXor	Array Logical Exclusive OR	FUN		AryXor(In1, In2, Size, AryOut);
AryXorN	Array Logical Exclusive NOR	FUN		AryXorN(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] and In2[] (arrays)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements		Number of elements to process			1
AryOut[] (array)	Processing results array	In-out	Processing results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	OK	OK	OK	OK	OK															
In2[] (array)	Must be same data type as *In1[]*																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[]*																			
Out	OK																			

Function

These instructions process *Size* elements from the beginning of arrays to process *In1[]* and *In2[]*. Processing is performed for corresponding bits of corresponding elements. The processing results are stored in corresponding elements of *AryOut[]*. *In1[]* to *In2[]* and *AryOut[]* must be the same data types.

The relationships between input and output variables are given in the following tables.

● AryAnd

If both bits are TRUE, then the processing result is TRUE. Otherwise, the processing result is FALSE.

Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

● AryOr

If both bits are FALSE, then the processing result is FALSE. Otherwise, the processing result is TRUE.

Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

AryXor

If both bits are the same, then the processing result is FALSE. If one bit is TRUE and the other is FALSE, then the processing result is TRUE.

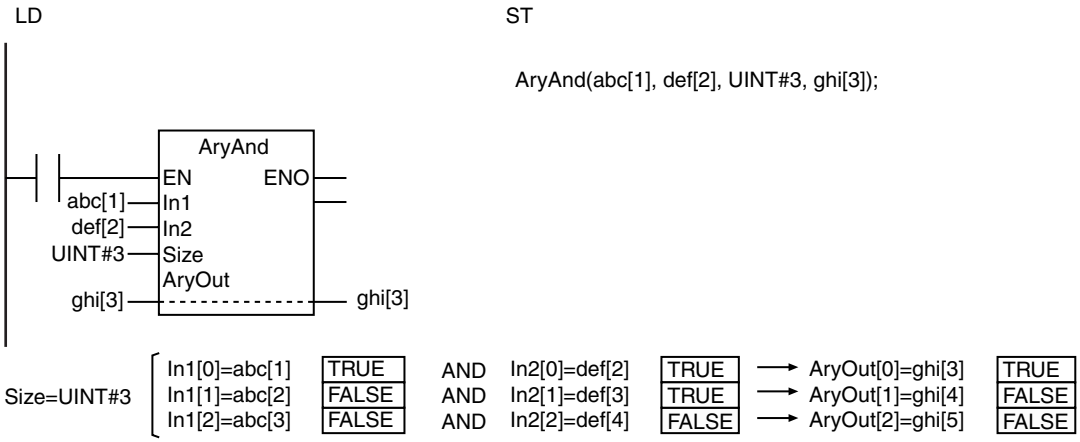
Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

AryXorN

If both bits are the same, then the processing result is TRUE. If one bit is TRUE and the other is FALSE, then the processing result is FALSE.

Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

The following example shows the AryAnd instruction when *Size* is UINT#3.



Precautions for Correct Use

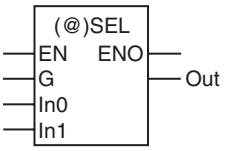
- The data types of *In1[]*, *In2[]*, and *AryOut[]* must be the same.
- Use an *AryOut[]* array that has at least as many elements as the value of *Size.*
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - *In1[], In2[],* and *AryOut[]* have different data types.
  - The value of *Size* exceeds the number of elements in *In1[], In2[],* or *AryOut[].*

Selection Instructions

Instruction	Name	Page
SEL	Binary Selection	2-298
MUX	Multiplexer	2-300
LIMIT	Limiter	2-302
Band	Deadband Control	2-304
Zone	Dead Zone Control	2-307
MAX and MIN	Maximum/Minimum	2-310
AryMax and AryMin	Array Maximum/Array Minimum	2-312
ArySearch	Array Search	2-314

SEL

The SEL instruction selects one of two selections.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SEL	Binary Selection	FUN		Out:=SEL(G, In0, In1);

Variables

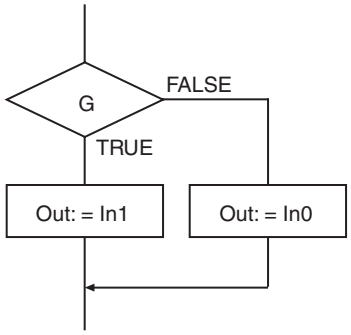
Name	Meaning	I/O	Description	Valid range	Unit	Default
G	Gate	Input	FALSE: Selects *In0.* TRUE: Selects *In1.*	Depends on data type.	---	FALSE
In0 and In1	Selections		Selections			*
Out	Selection result	Output	Selection result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
G	OK																			
In0 and In1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

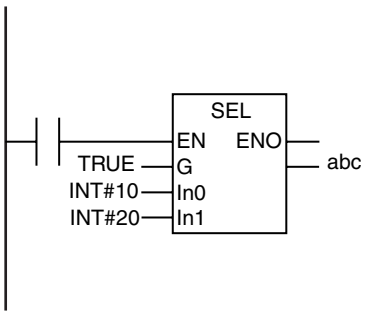
Function

The SEL instruction selects one of two selections, *In0* and *In1*. Gate *G* specifies which of *In0* and *In1* to select. If *G* is FALSE, *In0* is assigned to *Out*. If *G* is TRUE, *In1* is assigned to *Out*.



The following example is for when *In0* is INT#10, *In1* is INT#20, and *G* is TRUE. The value of variable *abc* will be INT#20.

LD



ST

```
abc:=SEL(TRUE, INT#10, INT#20);
```

Additional Information

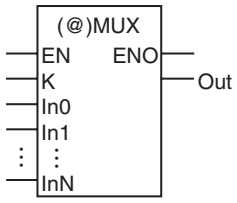
Use the MUX instruction (page 2-300) to select one of two or more selections.

Precautions for Correct Use

- *In0, In1,* and *Out* may be different data types, but observe the following precautions.
  - Set the valid range of *Out* to include the valid ranges of *In0* and *In1*.
  - *In0, In1,* and *Out* cannot be different varieties of data types (such as a bit string and an integer, or an integer and a text string).
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In0* or *In1* is STRING data and the number of bytes in the selection result exceeds the size of the output parameter that is connected to *Out*.
  - *In0* or *In1* is STRING data and it does not end in a NULL character.

MUX

The MUX instruction selects one of three to five selections.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MUX	Multiplexer	FUN		Out:=MUX(K, In0, In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
K	Selector	Input	0: Selects *In0.* 1: Selects *In1.* 2: Selects *In2.* 3: Selects *In3.* 4: Selects *In4.*	0 to N	---	*1
In0 to InN	Selections		Selections N is 2 to 4.	Depends on data type.		0*2
Out	Selection result	Output	Selection result	Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 2 and the input parameters that connect to *In0* and *In1* are omitted, the default values are applied, but if the input parameter that connects to *In2* is omitted, a building error will occur.

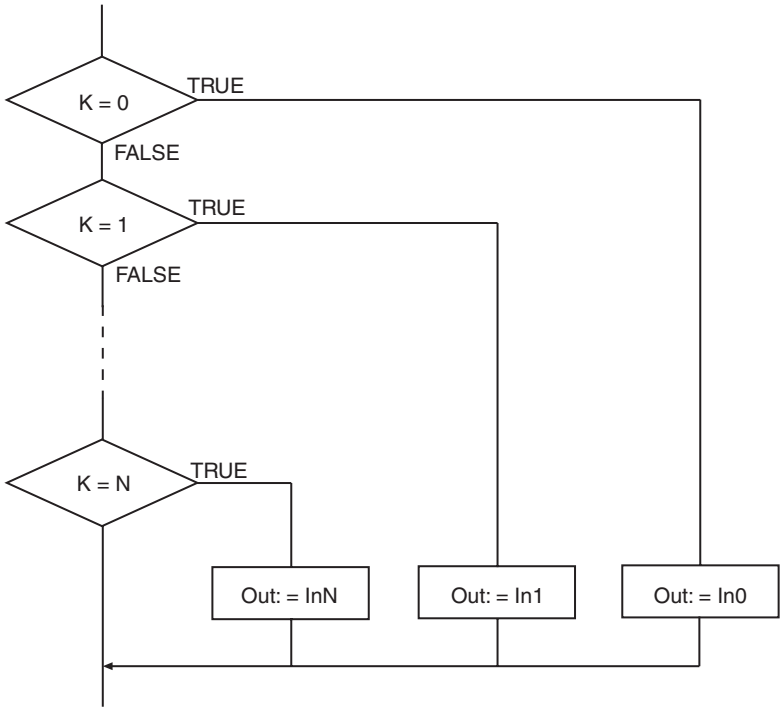
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
K						OK														
In0 to InN	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

Function

The MUX instruction selects one of three to five selections, *In0* to *InN*.

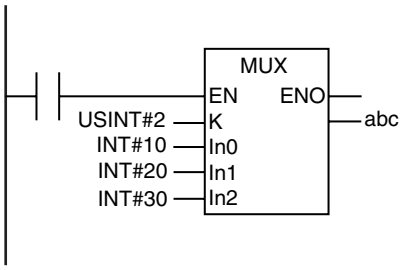
Selector *K* specifies which of *In0* to *InN* to select.

The value of one of the input variables is assigned to *Out* according to the value of *K*. *In0* is assigned if *K* is 0, *In1* is assigned if *K* is 1, *etc.*



The following example is for when *In0* is INT#10, *In1* is INT#20, *In2* is INT#30, and *K* is USINT#2. The value of variable *abc* will be INT#30.

LD



ST

```
abc:=MUX(USINT#2, INT#10, INT#20, INT#30);
```

Additional Information

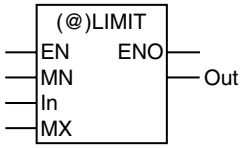
Use the SEL instruction (page 2-298) to select one of two selections.

Precautions for Correct Use

- *In0* to *InN* and *Out* may be different data types, but observe the following precautions.
  - Set the valid range of *Out* to include the valid ranges of *In0* to *InN*.
  - *In0* to *InN* and *Out* cannot be different varieties of data types (such as a bit string and an integer, or an integer and a text string).
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *K* is outside the valid range (i.e., less than 0 or greater than N).
  - A variable between *In0* and *InN* is STRING data and the number of bytes in the selection result exceeds the size of the output parameter that is connected to *Out*.
  - One of the variables between *In0* and *InN* is STRING data and it does not end in a NULL character.

LIMIT

The LIMIT instruction limits the value of the input variable to the specified minimum and maximum values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LIMIT	Limiter	FUN		Out:=LIMIT(MN, In, MX);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
MN	Minimum value	Input	Minimum value of limiter	Depends on data type.	---	*
In	Data to limit		Data to limit			
MX	Maximum value		Maximum value of limiter			
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
MX						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

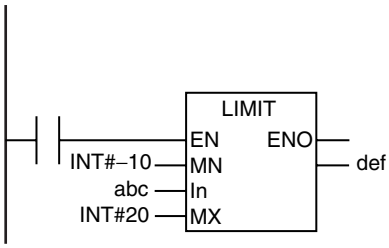
Function

The LIMIT instruction limits the value of data to limit *In* according to the maximum value, *MX*, and the minimum value, *MN*. The value of processing result *Out* is as shown below.

Value of *In*	Value of *Out*
In < MN	MN
MN ≤ In ≤ MX	In
MX < In	MX

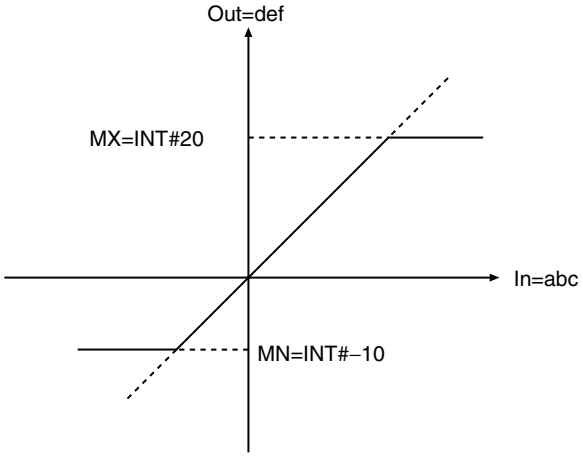
The following example is for when *MN* is INT#−10 and *MX* is INT#20.

LD



ST

```
def:=LIMIT(INT#−10, abc, INT#20);
```

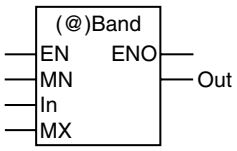


Precautions for Correct Use

- *In, MN, MX,* and *Out* may be different data types, but observe the following precautions.
  - Set the valid range of *Out* to include the valid ranges of *In, MN*, and *MX*.
  - Do not combine signed integers (SINT, INT, DINT, and LINT) together with unsigned integers (USINT, UINT, UDINT, and ULINT) for *In*, *MN*, and *MX*.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *MX* is smaller than the value of *MN.*

Band

The Band instruction performs deadband control.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Band	Deadband Control	FUN		Out:=Band(MN, In, MX);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
MN	Minimum value	Input	Minimum value of deadband	Depends on data type.	---	*
In	Data to control		Data to control			
MX	Maximum value		Maximum value of deadband			
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN										OK	OK	OK	OK	OK	OK					
In										OK	OK	OK	OK	OK	OK					
MX										OK	OK	OK	OK	OK	OK					
Out										OK	OK	OK	OK	OK	OK					

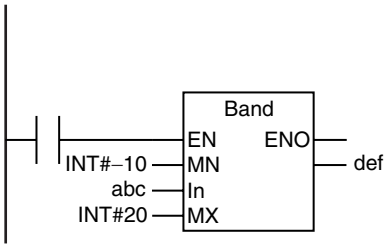
Function

The Band instruction controls the value of data to control *In* according to the maximum value, *MX*, and the minimum value, *MN*. The value of processing result *Out* is as shown below.

Value of *In*	Value of *Out*
In < MN	In − MN
MN ≤ In ≤ MX	0
MX < In	In − MX

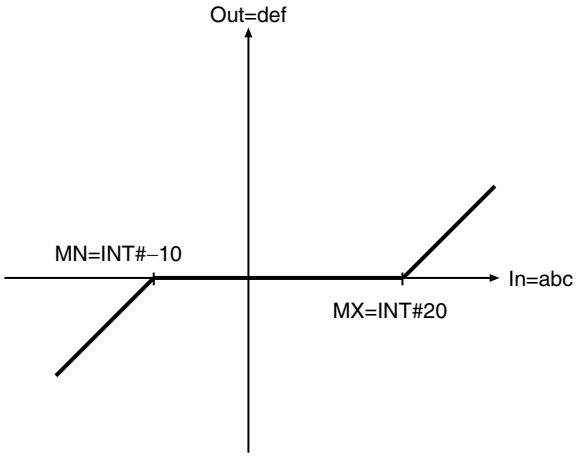
The following example is for when *MN* is INT#−10 and *MX* is INT#20.

LD



ST

```
def:=Band(INT#−10, abc, INT#20);
```



Precautions for Correct Use

- *In, MN, MX,* and *Out* may be different data types, but observe the following precaution.
  - Set the valid range of *Out* to include the valid ranges of *In, MN*, and *MX*.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the value of *In*, *MN*, or *MX* is positive infinity or negative infinity, the value of *Out* is as shown below.

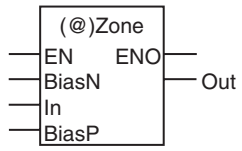
Value of *In*	Value of *MN*	Value of *MX*	Value of *Out*
+∞	+∞	+∞	0
		−∞	Error
	−∞	+∞	0
		−∞	+∞

Value of *In*	Value of *MN*	Value of *MX*	Value of *Out*
−∞	+∞	+∞	−∞
		−∞	Error
	−∞	+∞	0
		−∞	0

- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *MX* is smaller than the value of *MN.*
  - Either *MX* or *MN* contains nonnumeric data.
  - The processing result exceeds the valid range of *Out*.

Zone

The Zone instruction adds a bias value to the input value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Zone	Dead Zone Control	FUN		Out:=Zone(BiasN, In, BiasP);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
BiasN	Negative bias	Input	Negative bias	Depends on data type.	---	*
In	Data to control		Data to control			
BiasP	Positive bias		Positive bias			
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
BiasN										OK	OK	OK	OK	OK	OK					
In										OK	OK	OK	OK	OK	OK					
BiasP										OK	OK	OK	OK	OK	OK					
Out										OK	OK	OK	OK	OK	OK					

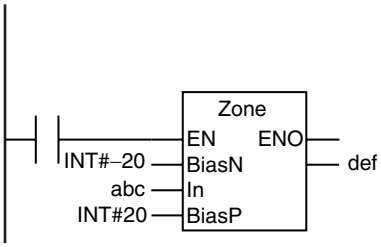
Function

The Zone instruction controls the value of data to control *In* according to the positive bias, *BiasP*, and the negative bias, *BiasN*. The value of processing result *Out* is as shown below.

Value of *In*	Value of *Out*
In < 0	In + BiasN
In = 0	0
0 < In	In + BiasP

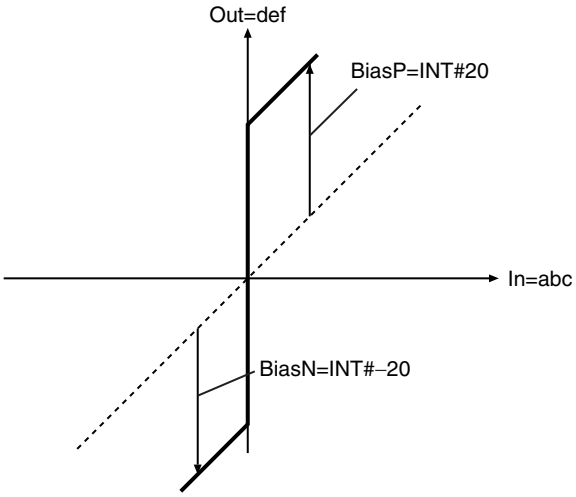
The following example is for when *BiasP* is INT#20 and *BiasN* is INT#−20.

LD



ST

```
def:=Zone(INT#−20, abc, INT#20);
```



Precautions for Correct Use

- *In, BiasP, BiasN,* and *Out* may be different data types, but observe the following precaution.
  - Set the valid range of *Out* to include the valid ranges of *In, BiasP*, and *BiasN*.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the value of *In*, *BiasP*, or *BiasN* is positive infinity or negative infinity, the value of *Out* is as shown below.

Value of *In*	Value of *BiasP*	Value of *BiasN*	Value of *Out*
+∞	+∞	+∞	+∞
		−∞	+∞
	−∞	+∞	Error
		−∞	0

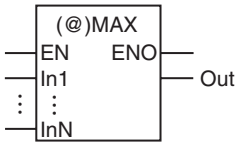
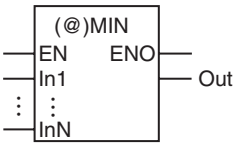
Value of *In*	Value of *BiasP*	Value of *BiasN*	Value of *Out*
−∞	+∞	+∞	0
		−∞	−∞
	−∞	+∞	Error
		−∞	−∞

- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *BiasP* is less than *BiasN*.
  - Either *BiasP* or *BiasN* contains nonnumeric data.
  - The processing result exceeds the valid range of *Out*.

MAX and MIN

MAX: Finds the largest of two to five values.

MIN: Finds the smallest of two to five values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MAX	Maximum	FUN		Out:=MAX(In1, In2, ···, InN);
MIN	Minimum	FUN		Out:=MIN(In1, In2, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Data to process	Input	Data to process, where N is 2 to 5	Depends on data type.	---	0*
Out	Search result	Output	Search result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

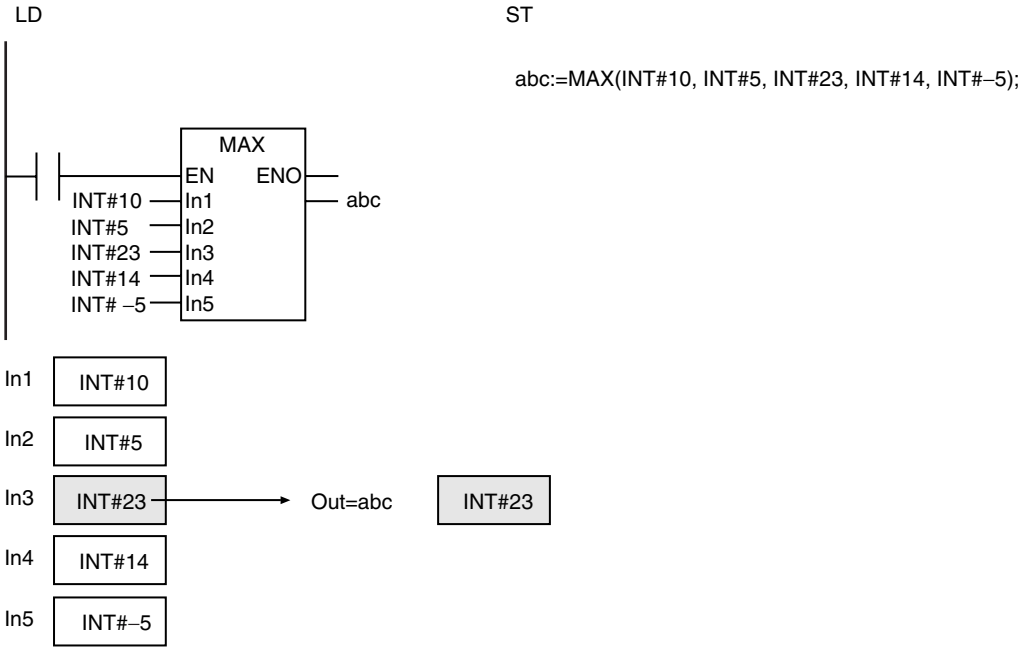
● MAX

The MAX instruction finds the largest value of two to five data to process, *In1* to *InN*.

● MIN

The MIN instruction finds the smallest value of two to five data to process, *In1* to *InN*.

The following example is for the MAX instruction when *In1* is INT#10, *In2* is INT#5, *In3* is INT#23, *In4* is INT#14, and *In5* is INT#−5.



Additional Information

To find the largest or smallest of six or more values, use the AryMax or AryMin instruction (page 2-312).

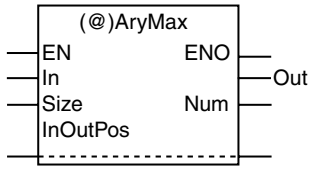
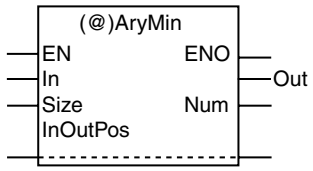
Precautions for Correct Use

- *In1* to *InN* and *Out* may be different data types, but observe the following precaution.
  - Set the valid range of *Out* to include the valid ranges of *In1* to *InN*.
  - Do not combine signed integers (SINT, INT, DINT, and LINT) together with unsigned integers (USINT, UINT, UDINT, and ULINT) for *In1* to *InN*.
- If *In1* to *InN* are real numbers, the desired results may not be achieved due to error.

AryMax and AryMin

AryMax: Finds the elements with the largest value in a one-dimensional array.

AryMin: Finds the elements with the smallest value in a one-dimensional array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryMax	Array Maximum	FUN		Out:=AryMax(In, Size, InOutPos, Num);
AryMin	Array Minimum	FUN		Out:=AryMin(In, Size, InOut-Pos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array to search	Depends on data type.	---	*
Size	Number of elements to search		Number of elements in *In[]* to search			1
InOutPos	Found element number	In-out	Array element number where value was found	Depends on data type.	---	---
Out	Search result	Output	Search result	Depends on data type.	---	---
Num	Number found		Number found			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Size							OK													
InOutPos							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Num							OK													

Function

These instructions search *Size* elements in array to search *In[]* starting from *In[0].* The value that is found is assigned to *Out*, the element number where it was found is assigned to *InOutPos*, and the number of times the value was found is assigned to *Num*. If *Num* is greater than 1, the value in *InOut-Pos* is the number of the lowest element that contains the value that was found.

● AryMax

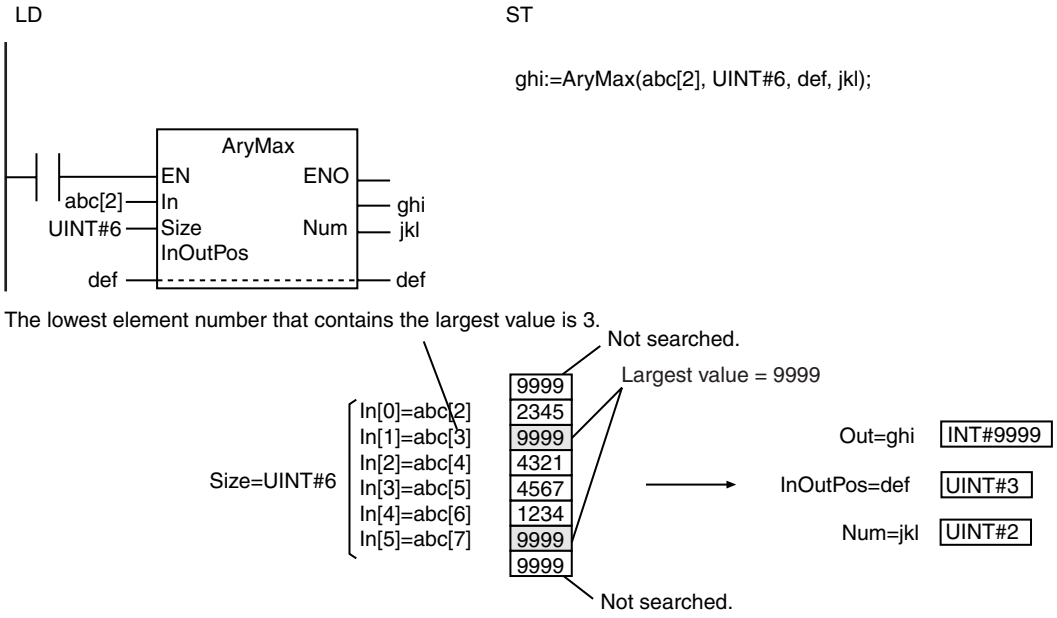
The AryMax instruction finds the largest value.

● AryMin

The AryMin instruction finds the smallest value.

The following example shows the AryMax instruction when *Size* is UINT#6.

The input parameter that is passed to *In[]* is *abc[2]*, so the search starts from *abc[2]*.

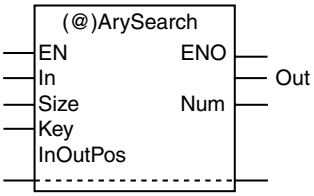


Precautions for Correct Use

- If you use a different data type for *In[]* and *Out*, make sure the valid range of *Out* includes the valid range of *In[].*
- If *In[]* contains real numbers, the desired results may not be achieved due to error.
- Always used a one-dimensional array for *In[].*
- If the value of *Size* is 0, the values of *Out* and *Num* are 0. The value of *InOutPos* does not change.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - *Size* exceeds the array area of *In[]*.
  - *In[]* is not a one-dimensional array.

ArySearch

The ArySearch instruction searches for the specified value in a one-dimensional array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySearch	Array Search	FUN		Out:=ArySearch(In, Size, Key, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array to search	Depends on data type.	---	*
Size	Number of elements to search		Number of elements in *In[]* to search	1 to 65535		1
Key	Search key		Value to search for	Depends on data type.		---
InOutPos	Found element number	In-out	Array element number where value was found	Depends on data type.	---	---
Out	Search result	Output	Search result	Depends on data type.	---	---
Num	Number found		Number found			

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations can also be specified.																			
Size							OK													
Key	Must be same data type as the elements of *In[]*.																			
InOutPos							OK													
Out	OK																			
Num							OK													

Function

The ArySearch instruction searches *Size* elements of one-dimensional array to search *In[]* for elements with the same value as search key *Key*. The search starts from *In[0]*.

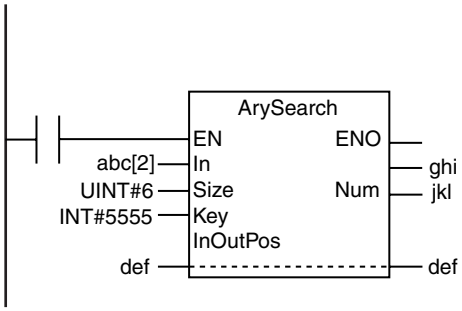
The values of search result *Out*, found element number *InOutPos*, and number found *Num* are as follows:

Element with same value as *Key*	Out	InOutPos	Num
Exists.	TRUE	Lowest element number that contains the same value as *Key*	Number of elements with same value as *Key*
Does not exist.	FALSE	Does not change.	0

The following example is for when *Size* is UINT#6 and *Key* is INT#5555.

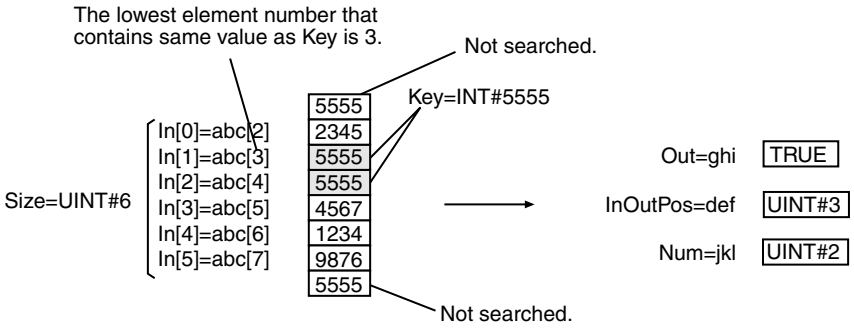
The input parameter that is passed to *In[]* is *abc[2]*, so the search starts from *abc[2]*.

LD



ST

```
ghi:=ArySearch(abc[2], UINT#6, INT#5555, def, jkl);
```



Precautions for Correct Use

- Always use a one-dimensional array for *In[]*.
- Make sure that *Key* has the same data type as the elements of *In[]*.
- If the value of *Size* is 0, the values of *Out* and *Num* are 0. The value of *InOutPos* does not change.
- When *Key* is an enumeration, always use a variable for the input parameter to pass to *Key*. A building error will occur if a constant is passed.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *Num*, and *InOutPos* will not change.
  - *Size* exceeds the array area of *In[]*.
  - *In[]* or *Key* is STRING data and it does not end in a NULL character.
  - *In[]* is not a one-dimensional array.

Data Movement Instructions

Instruction	Name	Page
MOVE	Move	2-318
MoveBit	Move Bit	2-321
MoveDigit	Move Digit	2-323
TransBits	Move Bits	2-325
MemCopy	Memory Copy	2-327
SetBlock	Block Set	2-329
Exchange	Data Exchange	2-331
AryExchange	Array Data Exchange	2-333
AryMove	Array Move	2-335
Clear	Initialize	2-337
Copy**ToNum (Bit String to Signed Integer)	Bit Pattern Copy (Bit String to Signed Integer) Group	2-339
Copy**To*** (Bit String to Real Number)	Bit Pattern Copy (Bit String to Real Number) Group	2-341
CopyNumTo** (Signed Integer to Bit String)	Bit Pattern Copy (Signed Integer to Bit String) Group	2-343
CopyNumTo** (Signed Integer to Real Number)	Bit Pattern Copy (Signed Integer to Real Number) Group	2-345
Copy**To*** (Real Number to Bit String)	Bit Pattern Copy (Real Number to Bit String) Group	2-347
Copy**ToNum (Real Number to Signed Integer)	Bit Pattern Copy (Real Number to Signed Integer) Group	2-349

MOVE

The MOVE instruction moves the value of a constant or variable to another variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MOVE	Move	FUN	(@)MOVE EN ENO In Out	Out:=In;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*
Out	Move destination	Output	Move destination	Depends on data type.	---	*

* If you omit the input parameter, the default value is not applied. A building error will occur.

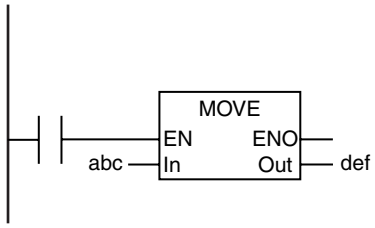
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Must be the same data type as *In* if *In* is an enumeration, array element, structure, or structure member. Must be an array with the same data type, size, and subscripts if *In* is an array,																			

Function

The MOVE instruction moves the value in move source *In* to move destination *Out*. The input parameter that is passed to *In* can be a variable or constant. You can specify an enumeration, array, array element, structure, or structure member for *In*.

The following figure shows a programming example. The content of variable *abc* is moved to variable *def.*

LD



ST

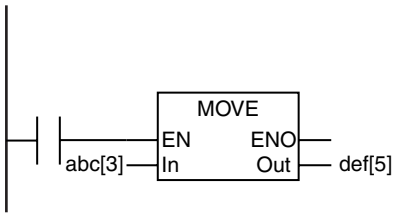
```
def:=abc;
```

Additional Information

- When moving an array, you can move either one element or all of the elements in the array. To move only one element, add the subscript to the array variable name. To move the entire array, do not add the subscript to the array variable name.

Moving One Array Element

LD

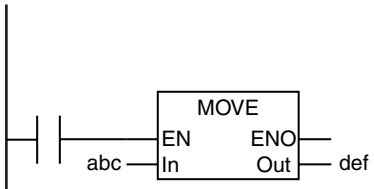


ST

```
def[5]:=abc[3];
```

Moving All Array Elements

LD



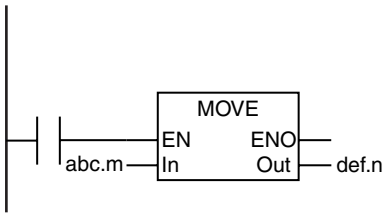
ST

```
def:=abc;
```

- When moving a structure, you can move either one member or all of the members in the structure. To move only one member, specify the member. To move the entire structure, give only the structure name.

Moving One Member of a Structure

LD

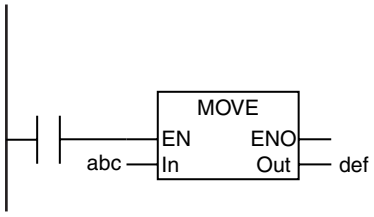


ST

```
def.n:=abc.m;
```

Moving the Entire Structure

LD



ST

```
def:=abc;
```

- You can use the MemCopy instruction to move an entire array faster than with the MOVE instruction.

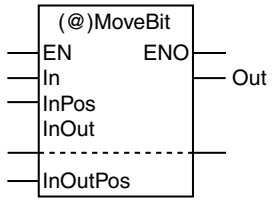
Precautions for Correct Use

- The data types of *In* and *Out* can be different as long as they are both in one of the following groups. The valid range of *Out* must include the valid range of *In*.

    - BYTE, WORD, DWORD, and LWORD
    - USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, and LREAL
- If *In* is an enumeration, array element, structure, or structure member, then *Out* must have the same data type as *In*.
- If *In* is an array, an array of the same data type, size, and subscripts must be used for *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
    - *In* is STRING data and it does not end in a NULL character.
    - *Out* is STRING data, but the text string that is moved exceeds the size of *Out*.

MoveBit

The MoveBit instruction moves one bit in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MoveBit	Move Bit	FUN		MoveBit(In, InPos, InOut, InOutPos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*
InPos	Move source bit		Position of bit in *In* to move	0 to No. of bits in *In* – 1		0
InOutPos	Move destination bit		Position of bit in *Out* to receive the bit	0 to No. of bits in *InOut* – 1		
InOut	Move destination	In-out	Move destination	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

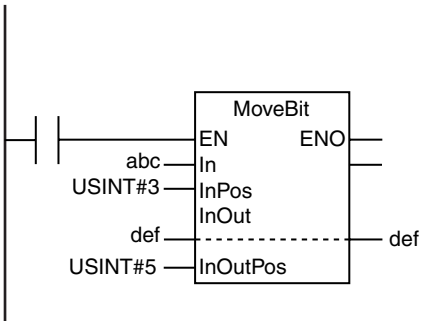
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
InPos						OK														
InOutPos						OK														
InOut		OK	OK	OK	OK															
Out	OK																			

Function

The MoveBit instruction moves one bit from the bit position *InPos* in move source *In* to the bit position *InOutPos* in move destination *InOut*.

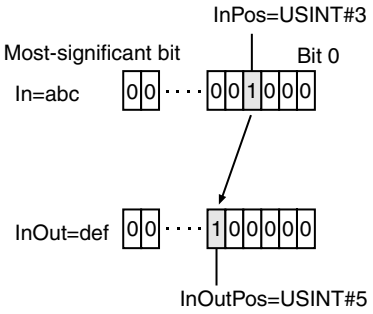
The following example is for when *InPos* is USINT#3 and *InOutPos* is USINT#5.

LD



ST

```
MoveBit(abc, USINT#3, def, USINT#5);
```

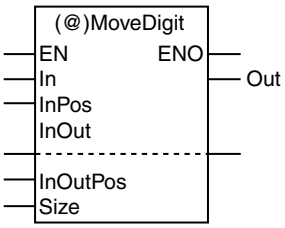


Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *InPos* is outside of the valid range.
  - The value of *InOutPos* is outside of the valid range.

MoveDigit

The MoveDigit instruction moves digits (4 bits per digit) in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MoveDigit	Move Digit	FUN		MoveDigit(In, InPos, InOut, InOutPos, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*1
InPos	Move source digit		Position of digit in *In* to move	*2		0
InOutPos	Move destination digit		Position of digit in *Out* to receive the digit	*3		
Size	Number of digits		Number of digits to move	*4		1
InOut	Move destination	In-out	Move destination	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 If you omit the input parameter, the default value is not applied. A building error will occur.

*2 0 to No. of bits in *In*/4 − 1

*3 0 to No. of bits in *InOut*/4 − 1

*4 0 to No. of bits in *In*/4

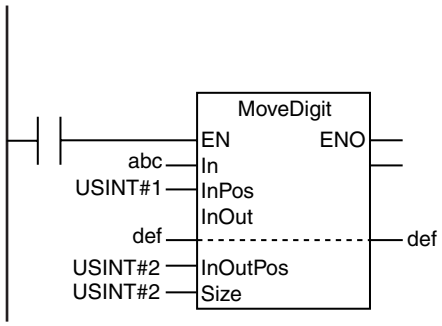
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
InPos						OK														
InOutPos						OK														
Size						OK														
InOut		OK	OK	OK	OK															
Out	OK																			

Function

The MoveDigit instruction moves *Size* digits from the *InPos* digit in move source *In* to the *InOutPos* digit in move destination *InOut*. One digit is four bits.

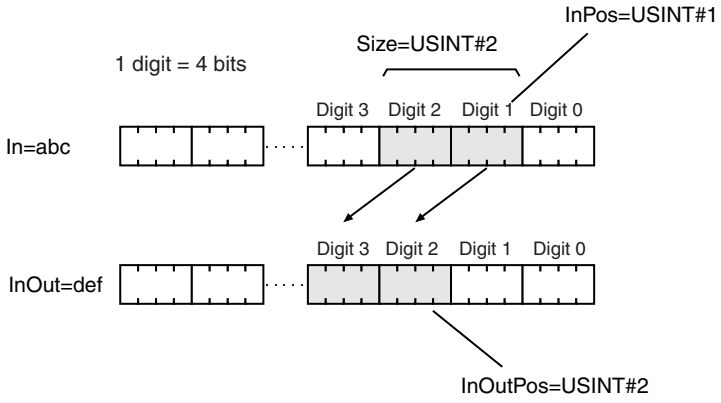
The following example is for when *InPos* is USINT#1, *InOutPos* is USINT#2, and *Size* is USINT#2.

LD



ST

```
MoveDigit(abc, USINT#1, def, USINT#2, USINT#2);
```

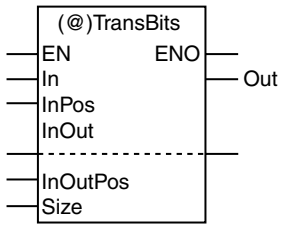


Precautions for Correct Use

- If the position of the digit at the destination exceeds the most-significant digit of *InOut*, the remaining digits are stored the least-significant digits of *InOut*.
- If the position of the digit at the source exceeds the most-significant digit of *In*, the remaining digits are moved to the least-significant digits of *In*.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *InOut* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *InPos* is outside of the valid range.
  - The value of *InOutPos* is outside of the valid range.
  - The value of *Size* is outside of the valid range.

TransBits

The TransBits instruction moves one or more bits in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TransBits	Move Bits	FUN		TransBits(In, InPos, InOut, InOutPos, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*1
InPos	Move source bit		Position of bit in *In* to move	*2		0
InOutPos	Move destination bit		Position of bit in *Out* to receive the bit	*3		
Size	Number of bits		Number of bits to move	*4		1
InOut	Move destination	In-out	Move destination	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 0 to No. of bits in *In* – 1

*3 0 to No. of bits in *InOut* – 1

*4 0 to No. of bits in *In*

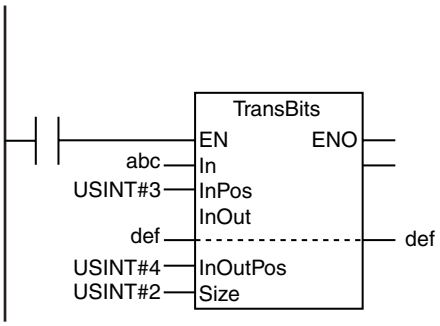
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
InPos						OK														
InOutPos						OK														
Size						OK														
InOut		OK	OK	OK	OK															
Out	OK																			

Function

The TransBis instruction moves *Size* bits from the *InPos* bit in move source *In* to the *InOutPos* bit in move destination *InOut*.

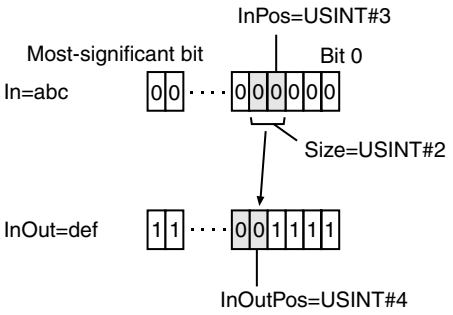
The following example is for when *InPos* is USINT#3, *InOutPos* is USINT#4, and *Size* is USINT#2.

LD



ST

```
TransBits(abc, USINT#3, def, USINT#4, USINT#2);
```



Additional Information

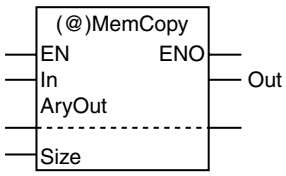
The bits in the move source and move destination can overlap.

Precautions for Correct Use

- Set the instruction so that the positions of the bits at the source and destination do not exceed the most-significant bit in *In* or *InOut*. An error will occur and the instruction will not operate.
- Nothing is moved if the value of *Size* is 0.
- The bits in *InOut* that are not involved in the move operation do not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *InPos* is outside of the valid range.
  - The value of *InOutPos* is outside of the valid range.
  - The value of *Size* is outside of the valid range.
  - The value of *InPos* or *Size* exceeds the number of bits in *In*.
  - The value of *InOutPos* or *Size* exceeds the number of bits in *InOut*.

MemCopy

The MemCopy instruction moves one or more array elements. The move source and move destination must have the same data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MemCopy	Memory Copy	FUN		MemCopy(In, AryOut, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Move source array	Input	Move source array	Depends on data type.	---	*
Size	Number of elements		Number of array elements to move			1
AryOut[] (array)	Move destination array	In-out	Move destination array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

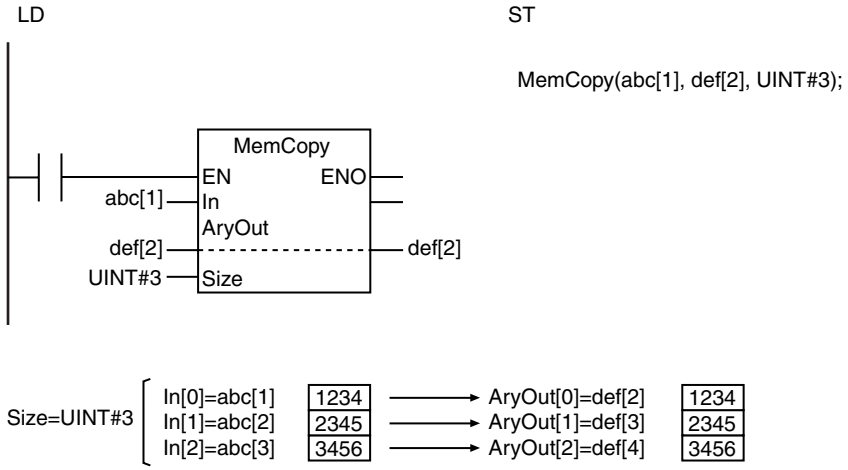
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Size							OK													
AryOut[] (array)	Must be an array with the same data type as *In[]*.																			
Out	OK																			

Function

The MemCopy instruction moves *Size* elements of move source array *In[]* starting from *In[0]* to move destination array *AryOut[]* starting from *AryOut[0]*.

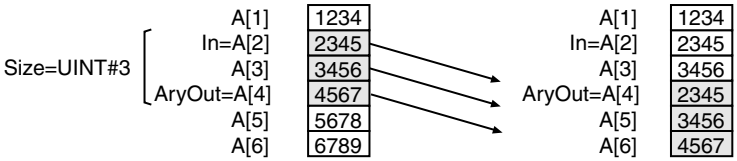
The following example is for when *Size* is UINT#3.



Additional Information

- You can specify different positions in the same array for *In[]* and *AryOut[]*. The source and destination data can overlap.

The following example is for when *In* is *A[2]*, *AryOut* is *A[4],* and *Size* is UINT#3.



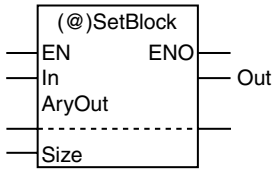
- Use the AryMove instruction (page 2-335) if the source and destination have different data types.
- If the data types of *In[]* and *AryOut[]* are the same, this instruction is faster than the AryMove instruction.
- Use the MOVE instruction (page 2-318) to move variables that are not arrays.

Precautions for Correct Use

- Use the same data type for *In[]* and *AryOut[].* If they are different, a building error will occur.
- If *In[]* and *AryOut[]* are STRING arrays, their sizes must be the same.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - *Size* exceeds the array area of *In[]*.
  - *Size* exceeds the array area of *AryOut[]*.

SetBlock

The SetBlock instruction moves the value of a variable or constant to one or more array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetBlock	Block Set	FUN		SetBlock(In, AryOut, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*
Size	Number of elements		Number of array elements to move			1
AryOut[] (array)	Move destination array	In-out	Move destination array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

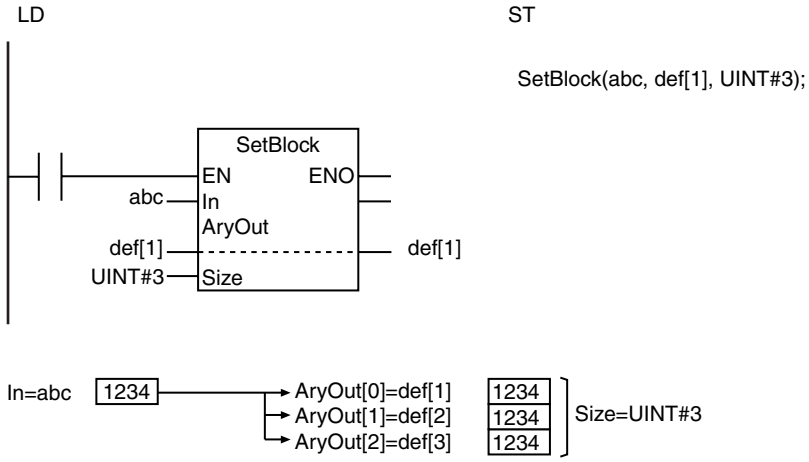
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
Size							OK													
AryOut[] (array)	Must be an array with elements that have the same data type as *In*.																			
Out	OK																			

Function

The SetBlock instruction moves the value of move source *In* to *Size* locations in move destination array *AryOut[]* starting from *AryOut[0]*.

The following example is for when *Size* is UINT#3.

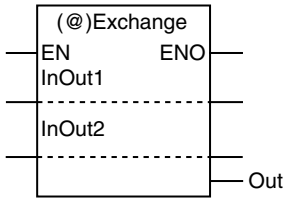


Precautions for Correct Use

- Use the same data type for *In* and *AryOut[].* If they are different, a building error will occur.
- If *In* and *AryOut[]* are STRING data, their sizes must be the same.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Size* exceeds the array area of *AryOut[].*

Exchange

The Exchange instruction exchanges the values of two variables.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Exchange	Data Exchange	FUN		Exchange(InOut1, InOut2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut1 and InOut2	Data to exchange	In-out	Data to exchange	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

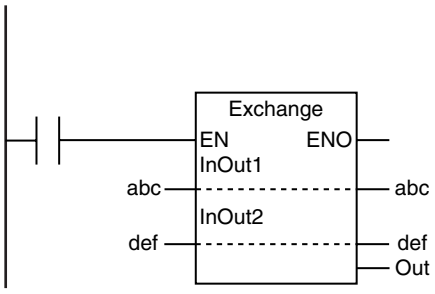
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
InOut2	Must be same data type as *InOut1.*																			
Out	OK																			

Function

The Exchange instruction exchanges the values of data to exchange *InOut1* and *InOut2.* You can specify enumerations, structures, or structure members for *InOut1* and *InOut2*.

The following figure shows a programming example. The values in variables *abc* and *def* are exchanged.

LD



ST

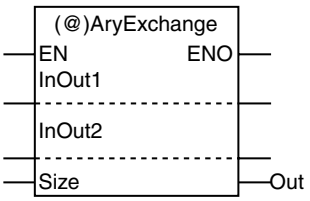
```
Exchange(abc, def);
```

Precautions for Correct Use

- The data types of *InOut1* and *InOut2* must be the same. If they are different, a building error will occur.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut1* and *InOut2* will not change.
  - Both *InOut1* and *InOut2* are STRING data and the length of the text string in one of them does not fit into the other.
  - *InOut1* and *InOut2* contain different data types.

AryExchange

The AryExchange instruction exchanges the elements of two arrays.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryExchange	Array Data Exchange	FUN		AryExchange(InOut1, InOut2, Size);

Variables

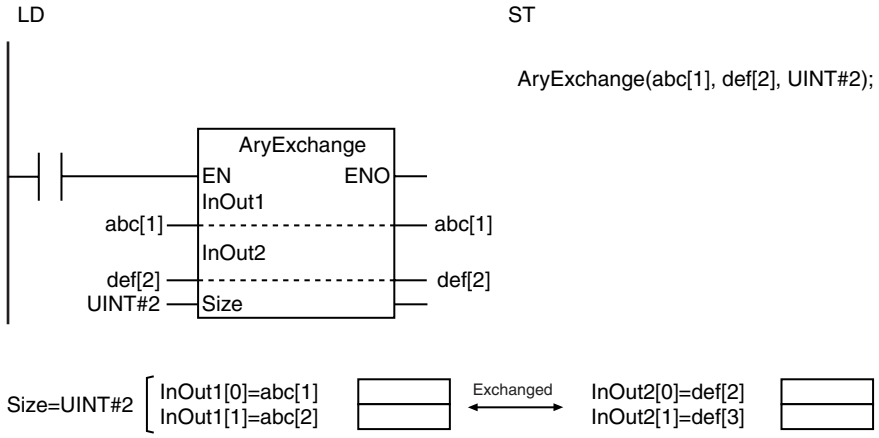
Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of elements	Input	Number of elements to exchange	Depends on data type.	---	1
InOut1[] and InOut2[] (arrays)	Arrays to exchange	In-out	Arrays to exchange	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
InOut1[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
InOut2[] (array)	Must be an array with the same data type as *InOut1[].*																			
Out	OK																			

Function

The AryExchange instruction exchanges *Size* elements from *InOut1[0]* of array to exchange *InOut1[]* with *Size* elements from *InOut2[0]* of array to exchange *InOut2[]*.

The following example is for when *Size* is UINT#2.



Additional Information

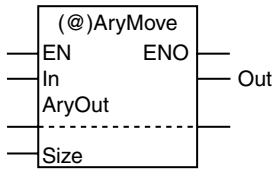
- Use the MOVE instruction (page 2-318) to assign constants to variables.
- Use the MemCopy instruction (page 2-327) to copy the values of variables to other variables.

Precautions for Correct Use

- Use the same data type for the elements of *InOut1[]* and *InOut2[]*. If they are different, a building error will occur.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *InOut1[]* and *InOut2[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut1[]* and *InOut2[]* will not change.
  - The value of *Size* exceeds the array range of *InOut1[]* or *InOut2[]*.
  - *InOut1[]* and *InOut2[]* are STRING arrays and there is an element with a text string that exceeds the size of the element in the other array.
  - *InOut1[]* and *InOut2[]* are STRING arrays and there is an element that does not end in a NULL character.

AryMove

The AryMove instruction moves one or more array elements. The data types of the move source and move destination can be different.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryMove	Array Move	FUN		AryMove(In, AryOut, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Move source array	Input	Array to move	Depends on data type.	---	*
Size	Number of elements		Number of elements to move			1
AryOut[] (array)	Move result array	In-out	Move result array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

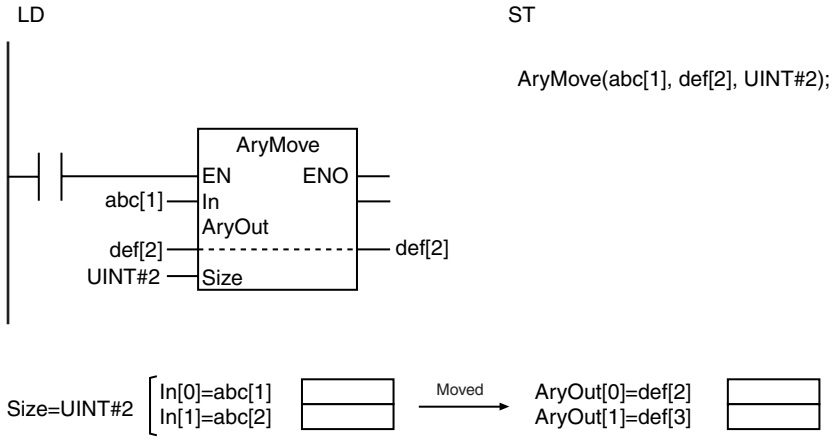
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Size							OK													
AryOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Out	OK																			

Function

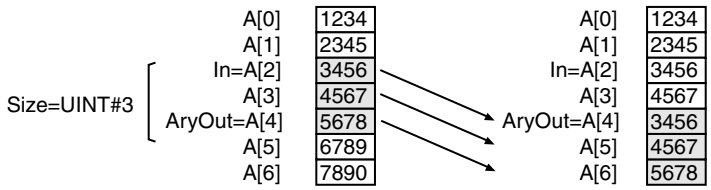
The *AryMove* instruction moves *Size* elements of move source array *In[]* starting from *In[0]* to move result array *AryOut[]* starting from *AryOut[0]*. The data types of *In[]* and *AryOut[]* can be different.

The following example is for when *Size* is UINT#2.



Additional Information

- If the data types of *In[]* and *AryOut[]* are the same, the MemCopy instruction is faster.
- You can specify the same array for *In[]* and *AryOut[]*. Also, the move source and destination data can overlap. The following example is for when *In[0]* is *A[2]*, *AryOut[0]* is *A[4],* and *Size* is UINT#3.

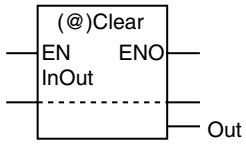


Precautions for Correct Use

- The data types of *In[]* and *AryOut[]* can be different as long as they are both in one of the following groups. The valid range of *AryOut[]* must include the valid range of *In[]*.
  - BYTE, WORD, DWORD, and LWORD
  - USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, and LREAL
- If *In[]* is an array of structures, use the same data types for *In[]* and *AryOut[]*.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Size* exceeds the size of *In[]* or Ary*Out[]*.
  - *In[]* and *AryOut[]* are STRING arrays and one of the elements to move does not end in a NULL character.
  - *In[]* or *AryOut[]* is a STRING array and the length of a text string in an element to move exceeds the size of the element in *AryOut[]*.

Clear

The Clear instruction initializes a variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Clear	Initialize	FUN		Clear(InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Data to initialize	In-out	Data to initialize	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Out	OK																			

Function

The Clear instruction initializes the value of data to initialize *InOut*.
If an initial value attribute is set for a variable, the specified initial value is used. If an initial value attribute is not set, the default value for the data type is used.

The default values for the data types are given below.

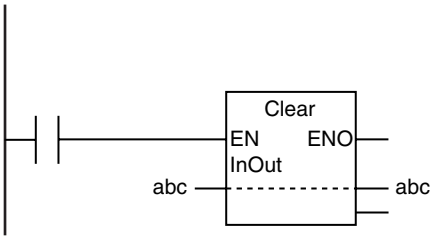
Data type	Default
BOOL	FALSE
BYTE, WORD, DWORD, or LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, or LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	''

If *InOut* is an array, array element, structure, or structure member, the following processing is performed.

InOut	Processing
Array	All elements in the array are initialized.
Array element	Only the specified element is initialized.
Structure	All members in the structure are initialized.
Structure member	Only the specified member is initialized.

The following figure shows a programming example. The value of variable *abc* is initialized.

LD



ST

```
Clear(abc);
```

Additional Information

- If *InOut* is an array that is used as a stack, execute this instruction and also set the variable that manages the number of items stored in the stack to 0.
- If you initialize a cam data variable with this instruction, it will not contain the data that was saved with the MC_SaveCamTable instruction. It will contain all zeros.

Precautions for Correct Use

Return value *Out* is not used when the instruction is used in ST.

Copy**ToNum (Bit String to Signed Integer)

The Copy**ToNum instruction copies the content of a bit string directly to a signed integer.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**ToNum	Bit Pattern Copy (Bit String to Signed Integer) Group	FUN	(@)Copy**ToNum EN ENO In Out "**" must be a bit string data type.	Out:=Copy**ToNum(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

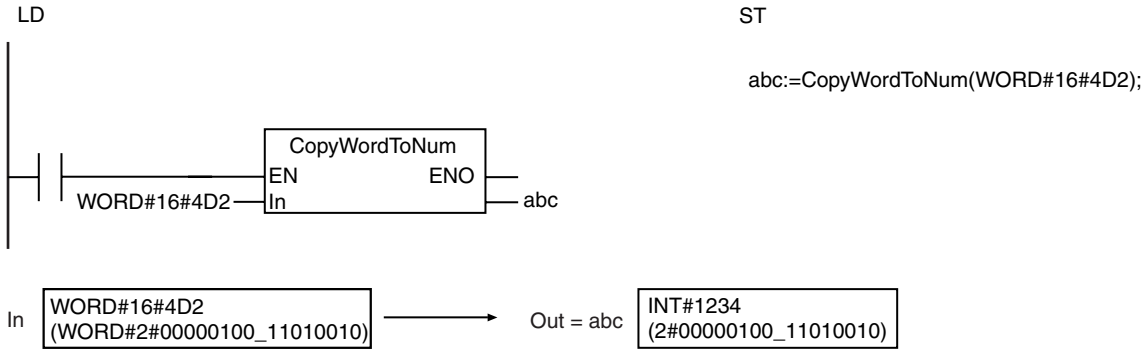
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out	Must be a signed integer data type that is the same size as the data type of *In*.																			

Function

The Copy**ToNum instruction copies the content of copy source *In* directly to copy destination *Out*.
There are four instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
BYTE	SINT	CopyByteToNum
WORD	INT	CopyWordToNum
DWORD	DINT	CopyDwordToNum
LWORD	LINT	CopyLwordToNum

The following example for the CopyWordToNum instruction is for when *In* is WORD#16#4D2.



Copy**To*** (Bit String to Real Number)

The Copy**To*** instruction copies the content of a bit string directly to a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**To***	Bit Pattern Copy (Bit String to Real Number) Group	FUN	(@)CopyDwordToReal EN ENO In Out (@)CopyLwordToLreal EN ENO In Out	Out:=CopyDwordToReal(In); or Out:=CopyLwordToLreal(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In				OK	OK															
Out	Must be REAL if the data type of *In* is DWORD and LREAL if the data type of *In* is LWORD.																			

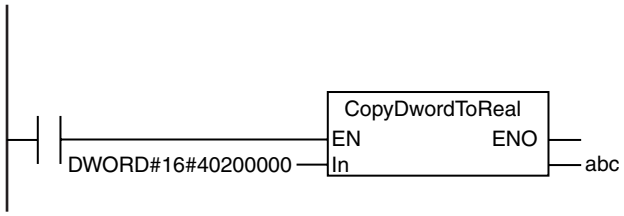
Function

The Copy**To*** instruction copies the content of copy source *In* directly to copy destination *Out*. There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
DWORD	REAL	CopyDwordToReal
LWORD	LREAL	CopyLwordToLreal

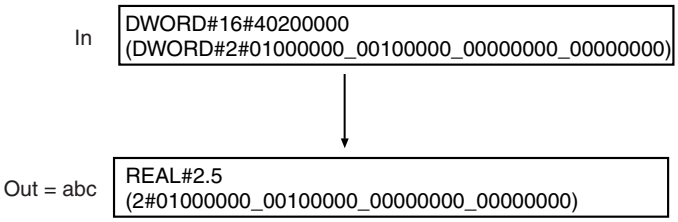
The following example for the CopyDwordToReal instruction is for when *In* is DWORD#16#40200000.

LD



ST

```
abc:=CopyDwordToReal(DWORD#16#40200000);
```



CopyNumTo** (Signed Integer to Bit String)

The CopyNumTo** instruction copies the content of a signed integer directly to a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CopyNumTo**	Bit Pattern Copy (Signed Integer to Bit String) Group	FUN	(@)CopyNumTo** EN ENO In Out "**" must be a bit string data type.	Out:=CopyNumTo**(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

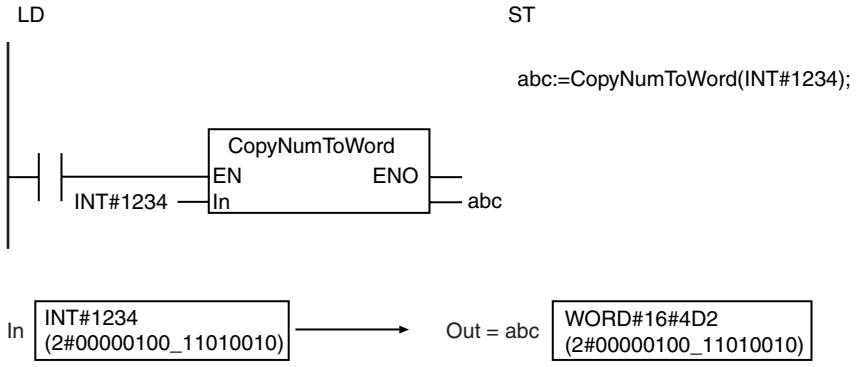
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In										OK	OK	OK	OK							
Out	Must be a bit string data type that is the same size as the data type of *In*.																			

Function

The CopyNumTo** instruction copies the content of copy source *In* directly to copy destination *Out*.
There are four instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
SINT	BYTE	CopyNumToByte
INT	WORD	CopyNumToWord
DINT	DWORD	CopyNumToDword
LINT	LWORD	CopyNumToLword

The following example for the CopyNumToWord instruction is for when *In* is INT#1234.



CopyNumTo** (Signed Integer to Real Number)

The CopyNumTo** instruction copies the content of a signed integer directly to a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CopyNumTo**	Bit Pattern Copy (Signed Integer to Real Number) Group	FUN	(@)CopyNumToReal EN ENO In Out (@)CopyNumToLreal EN ENO In Out	Out:=CopyNumToReal(In); or Out:=CopyNumToLreal(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In												OK	OK							
Out	Must be REAL if the data type of *In* is DINT and LREAL if the data type of *In* is LINT.																			

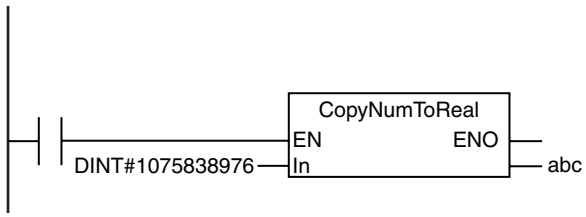
Function

The CopyNumTo** instruction copies the content of copy source *In* directly to copy destination *Out*.
There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
DINT	REAL	CopyNumToReal
LINT	LREAL	CopyNumToLreal

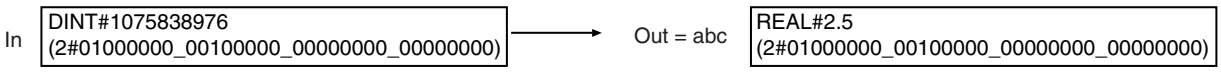
The following example for the CopyNumToReal instruction is for when *In* is DINT#1075838976.

LD



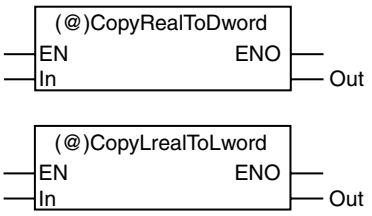
ST

```
abc:=CopyNumToReal(DINT#1075838976);
```



Copy**To*** (Real Number to Bit String)

The Copy**To*** instruction copies the content of a real number directly to a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**To***	Bit Pattern Copy (Real Number to Bit String) Group	FUN		Out:=CopyRealToDword(In); or Out:=CopyLrealToLword(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0.0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

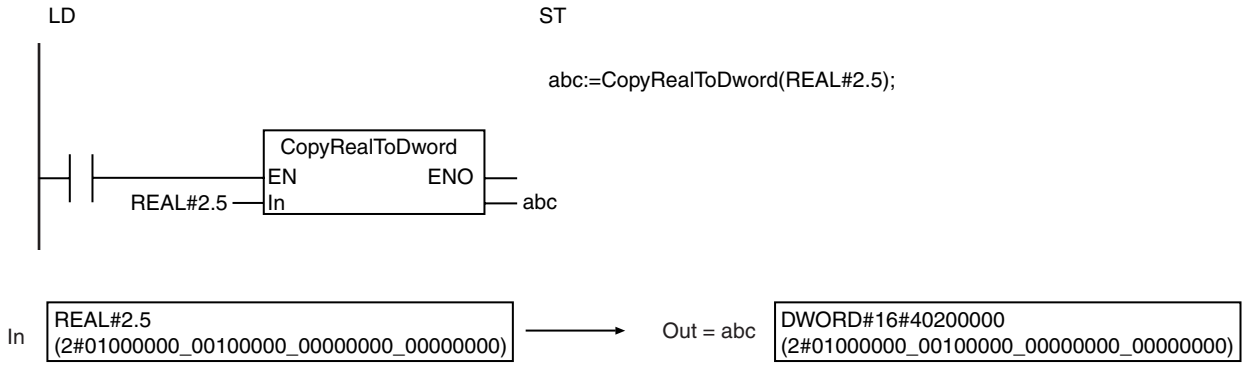
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	Must be DWORD if the data type of *In* is REAL and LWORD if the data type of *In* is LREAL.																			

Function

The Copy**To*** instruction copies the content of copy source *In* directly to copy destination *Out*. There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
REAL	DWORD	CopyRealToDword
LREAL	LWORD	CopyLrealToLword

The following example for the CopyRealToDword instruction is for when *In* is REAL#2.5.



Copy**ToNum (Real Number to Signed Integer)

The Copy**ToNum instruction copies the content of a real number directly to a signed integer.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**ToNum	Bit Pattern Copy (Real Number to Signed Integer) Group	FUN	(@)CopyRealToNum EN ENO In Out (@)CopyLrealToNum EN ENO In Out	Out:=CopyRealToNum(In); or Out:=CopyLrealToNum(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0.0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

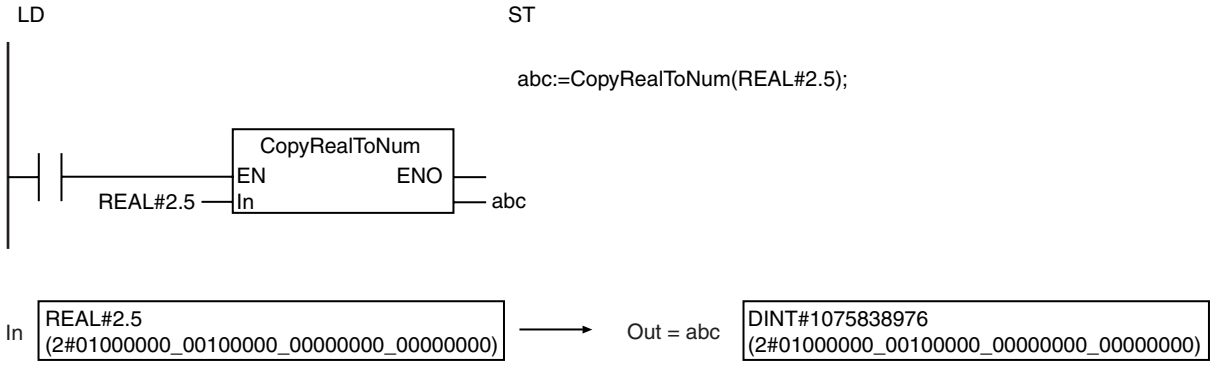
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	Must be DINT if the data type of *In* is REAL and LINT if the data type of *In* is LREAL.																			

Function

The Copy**ToNum instruction copies the content of copy source *In* directly to copy destination *Out*.
There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
REAL	DINT	CopyRealToNum
LREAL	LINT	CopyLrealToNum

The following example for the CopyRealToNum instruction is for when *In* is REAL#2.5.

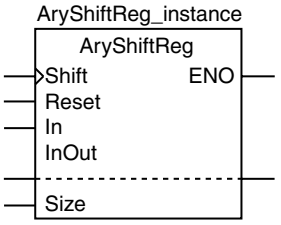


Shift Instructions

Instruction	Name	Page
AryShiftReg	Shift Register	2-352
AryShiftRegLR	Reversible Shift Register	2-354
ArySHL and ArySHR	Array N-element Left Shift/ Array N-element Right Shift	2-357
SHL and SHR	N-bit Left Shift/ N-bit Right Shift	2-360
NSHLC and NSHRC	Shift N-bits Left with Carry/ Shift N-bits Right with Carry	2-362
ROL and ROR	Rotate N-bits Left/ Rotate N-bits Right	2-364

AryShiftReg

The AryShiftReg instruction shifts a shift register one bit to the left and inserts the input value to the least-significant bit. The shift register consists of array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryShiftReg	Shift Register	FB		AryShiftReg_instance(Shift, Reset, In, InOut, Size);

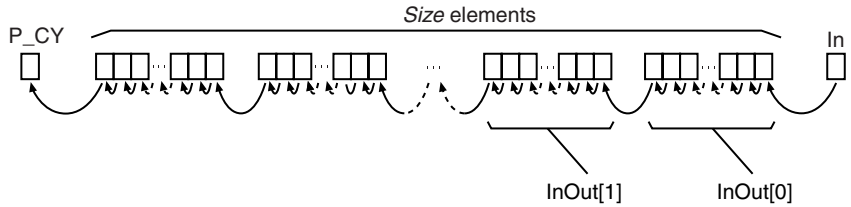
Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Shift	Shift	Input	Shifted when signal changes to TRUE.	Depends on data type.	---	FALSE
Reset	Reset		TRUE: Register is reset.			
In	Input value		Value to insert to least-significant bit of *InOut[]*.			
Size	Number of elements in array of bit strings		Number of elements to use as a shift register in *InOut[]*.			1
InOut[] (array)	Array of bit strings	In-out	Array of bit strings	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Shift	OK																			
Reset	OK																			
In	OK																			
Size							OK													
InOut[] (array)	OK	OK	OK	OK	OK															

Function

The AryShiftReg instruction shifts *Size* array elements in the array of bit strings *InOut[]* to the left (i.e., toward most-significant bit) when *Shift* changes to TRUE. The shift operation starts from *InOut[0].* Input value *In* is inserted to the least-significant bit. The most-significant bit of the array of bit strings is output to the Carry (CY) Flag (P_CY).



When Reset is TRUE, CY and all of bits in *Size* elements starting from *InOut[0]* are set to FALSE.

The following example is for when *InOut[]* is a BYTE array and *Size* is UINT#2.

Related System-defined Variables

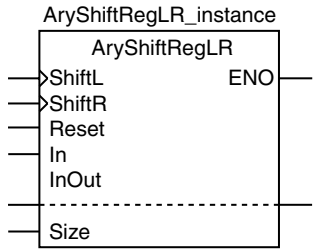
Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	Value stored in Carry Flag

Precautions for Correct Use

- While *Reset* is TRUE, the register is not shifted even if *Shift* changes to TRUE.
- *ENO* will change to TRUE when *Shift* changes to TRUE and the shift operation is performed normally, or when *Reset* is TRUE and the reset operation is performed normally.
- The *InOut[]* does not change if the value of *Size* is 0.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

AryShiftRegLR

The AryShiftRegLR instruction shifts a bit string one bit to the left or right and inserts the input value to the least-significant or most-significant bit. The bit string consists of array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryShiftRegLR	Reversible Shift Register	FB		AryShiftRegLR_instance (ShiftL, ShiftR,Reset, In, InOut, Size);

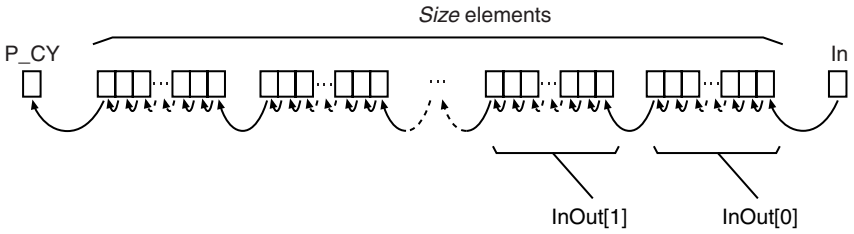
Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
ShiftL	Left shift	Input	Shifted left when signal changes to TRUE.	Depends on data type.	---	FALSE
ShiftR	Right shift		Shifted right when signal changes to TRUE.			
Reset	Reset		TRUE: Register is reset.			
In	Input value		Value to insert to least-significant or most-significant bit of *InOut[]*			
Size	Number of elements in array of bit strings		Number of elements to use as a shift register in *InOut[].*			1
InOut[] (array)	Array of bit strings	In-out	Array of bit strings	Depends on data type.	---	---

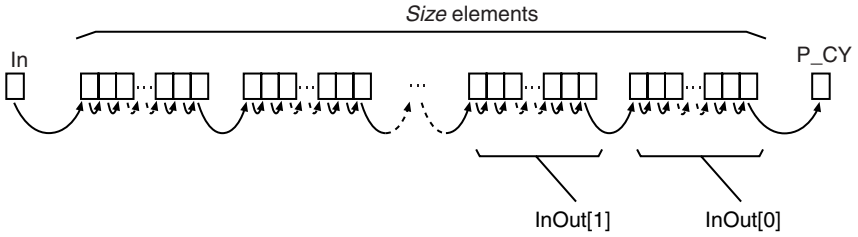
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ShiftL	OK																			
ShiftR	OK																			
Reset	OK																			
In	OK																			
Size							OK													
InOut[] (array)	OK	OK	OK	OK	OK															

Function

The AryShiftRegLR instruction shifts *Size* array elements of array of bit strings *InOut[]* to the left when *ShiftL* changes to TRUE. The shift operation starts from *InOut[0]*. Input value *In* is inserted to the least-significant bit. The most-significant bit of the array of bit strings is output to the Carry (CY) Flag (P_CY).

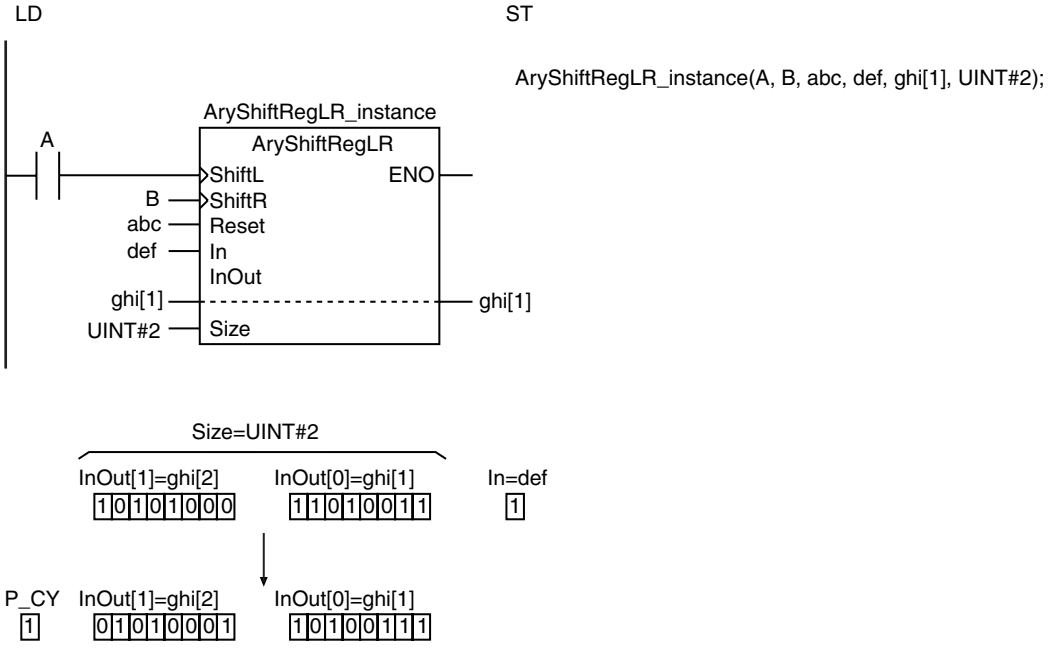


When *ShiftR* changes to TRUE, the bits are shifted one bit to the right and *In* is inserted to the most-significant bit. The least-significant bit of the array of bit strings is output to the Carry (CY) Flag (P_CY).



When Reset is TRUE, P_CY and all of the bits in *Size* elements starting from *InOut[0]* are set to FALSE.

The following example is for when *InOut* is BYTE data, *Size* is UINT#2 and *ShiftL* changes to TRUE.



Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	Value stored in Carry Flag

Precautions for Correct Use

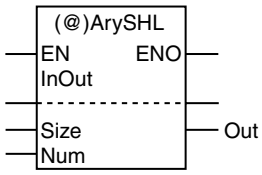
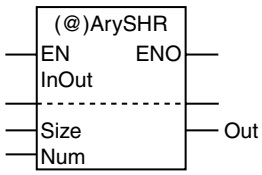
- While *Reset* is TRUE, the register is not shifted even if *ShiftL* or *ShiftR* changes to TRUE.
- The register is not shifted if both *ShiftL* and *ShiftR* change to TRUE at the same time.
- *ENO* will change to TRUE when *ShiftL* or *ShiftR* changes to TRUE and the shift operation is performed normally, or when *Reset* is TRUE and the reset operation is performed normally.
- The *InOut[]* does not change if the value of *Size* is 0.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

ArySHL and ArySHR

These instructions shift array elements by one or more elements.

ArySHL: Shifts the array to the left (toward the higher elements).

ArySHR: Shifts the array to the right (toward the lower elements).

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySHL	Array N-element Left Shift	FUN		ArySHL(InOut, Size, Num);
ArySHR	Array N-element Right Shift	FUN		ArySHR(InOut, Size, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of elements in shift register	Input	Number of elements in shift register	Depends on data type.	---	1
Num	Number of elements to shift		Number of elements to shift			
InOut[] (array)	Shift register array	In-out	Shift register array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
Num							OK													
InOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of structures can also be specified.																			
Out	OK																			

Function

These instructions shift the upper *Size* elements in shift register array *InOut[]* by *Num* elements.

The values that are shifted out of the array are discarded.

The default value for the data type of *InOut[]* is stored in the empty elements. If an initial value attribute is set for *InOut[]*, the specified initial value is used. If an initial value attribute is not set, the default value for the data type is used. If *InOut[]* is an array of structures, the members of the structures in all elements are initialized.

The default values for the data types are given below.

Data type of *InOut*	Default
BOOL	FALSE
BYTE, WORD, DWORD, or LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, or LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	''

● ArySHL

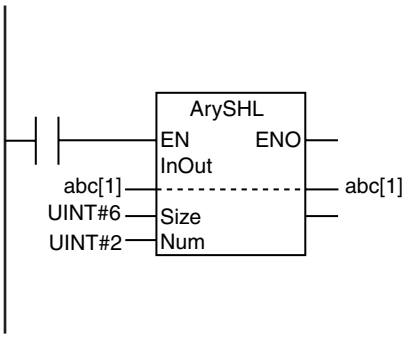
The ArySHL instruction shifts the array to the left (toward the higher elements of the array).

● ArySHR

The ArySHR instruction shifts the array to the right (toward the lower elements of the array).

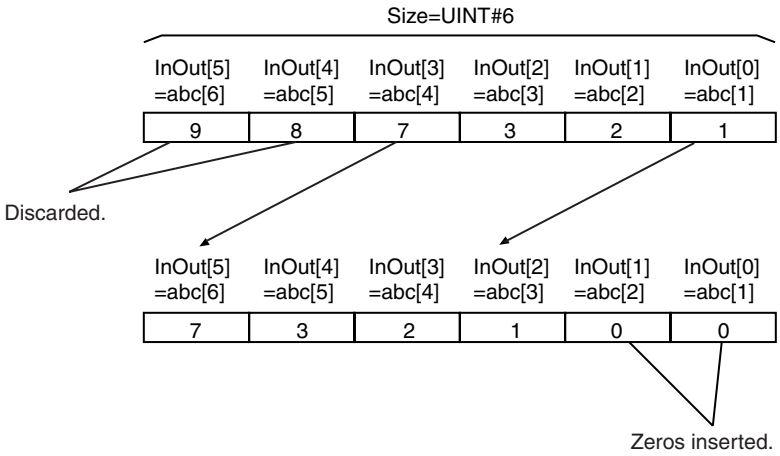
The following example shows the ArySHL instruction when *Size* is UINT#6 and *Num* is UINT#2.

LD



ST

```
ArySHL(abc[1], UINT#6, UINT#2);
```



Additional Information

If *InOut[]* is BOOL data, the results will be the same as shifting a bit string of *Size* bits by *Num* bits.

Precautions for Correct Use

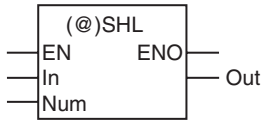
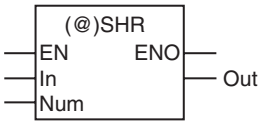
- The shift operation is not performed if the value of *Num* is 0.
- If the value of *Num* is larger than *Size*, all values from *InOut[0]* to *InOut[Size*–1] are initialized.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

SHL and SHR

These instructions shift a bit string by one or more bits.

SHL: Shifts the bit string to the left (toward the higher bits).

SHR: Shifts the bit string to the right (toward the lower bits).

Instruction	Name	FB/FUN	Graphic expression	ST expression
SHL	N-bit Left Shift	FUN		Out:=SHL(In, Num);
SHR	N-bit Right Shift	FUN		Out:=SHR(In, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to shift	Input	Data to shift	Depends on data type.	---	*
Num	Number to shift		Number of bits to shift	0 to No. of bits in *In*	Bits	1
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK														
Out	Must be same data type as *In*																			

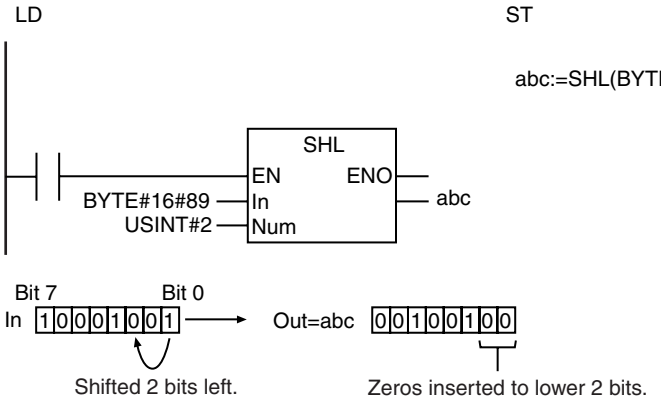
Function

These instructions shift data to shift *In* (bit string data) by the number of bits specified in number to shift *Num*. The bits that are shifted out of the register are discarded and zeros are inserted into the other end of the register.

● SHL

The SHL instruction shifts bits from right to left (from least-significant to most-significant bits).

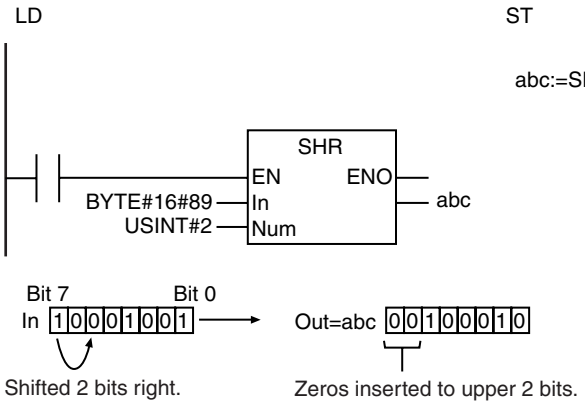
The following example if for when *In* is BYTE#16#89 and *Num* is USINT#2.



● SHR

The SHR instruction shifts bits from left to right (from most-significant to least-significant bits).

The following example shows the SHR instruction when *In* is BYTE#16#89 and *Num* is USINT#2.



Additional Information

The ROL and ROR instructions insert the bits that are shifted out of the register into the other end of the register.

Precautions for Correct Use

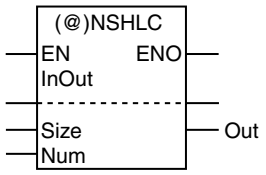
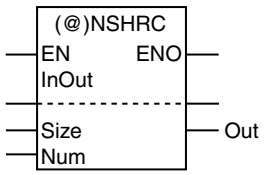
- The data types of *In* and *Out* must be the same.
- If *Num* is 0, an error will not occur and the value of *In* will be assigned directly to *Out*.
- If the value of *Num* exceeds the number of bits specified in *In*, an error will not occur and the value of *Out* will be 16#0.

NSHLC and NSHRC

These instructions shift an array of bit strings by one or more bits. The Carry (CY) Flag is included.

NSHLC: Shifts the array to the left (toward the higher elements).

NSHRC: Shifts the array to the right (toward the lower elements).

Instruction	Name	FB/FUN	Graphic expression	ST expression
NSHLC	Shift N-bits Left with Carry	FUN		NSHLC(InOut, Size, Num);
NSHRC	Shift N-bits Right with Carry	FUN		NSHRC(InOut, Size, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of bits in shift register	Input	Number of bits in shift register	Depends on data type.	Bits	1
Num	Number of bits to shift		Number of bits to shift			
InOut[] (array)	Shift register array	In-out	Bit string array to shift	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size						OK														
Num						OK														
InOut[] (array)	OK	OK	OK	OK	OK															
Out	OK																			

Function

These instructions shift *Size* array elements in shift register array *InOut[]* by the number of bits specified in *Num*. The shift register starts at *InOut[0]*. The last bit that is shifted out of the register is output to the Carry (CY) Flag. Zeros are inserted for the bits at the other end.

● NSHLC

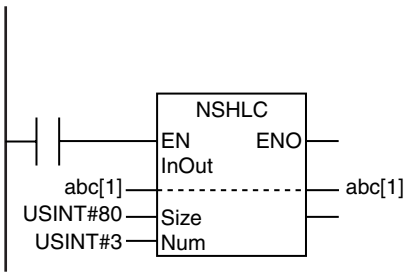
The NSHLC instruction shifts bits from the lower elements in the array to the higher elements and from the least-significant bits to the most-significant bits.

● NSHRC

The NSHRC instruction shifts bits from the higher elements in the array to the lower elements and from the most-significant bits to the least-significant bits.

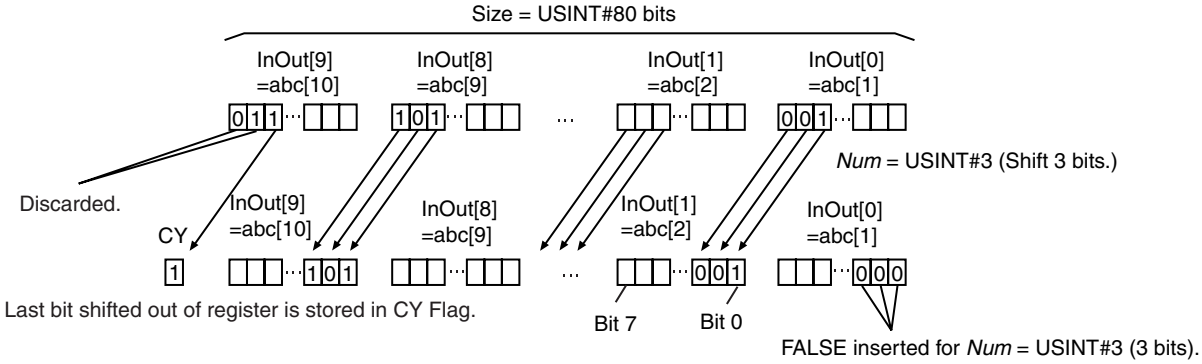
The following example shows the NSHLC instruction when *InOut[]* is a BYTE array, *Size* is USINT#80 and *Num* is USINT#3.

LD



ST

```
NSHLC(abc[1], USINT#80, USINT#3);
```



Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	Value stored in Carry Flag

Precautions for Correct Use

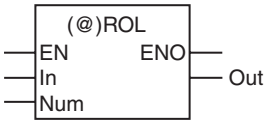
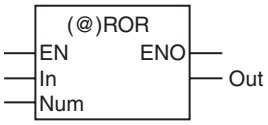
- The shift operation is not performed if the value of *Num* is 0.
- If the value of *Num* is larger than *Size*, *Size* bits from bit 0 of *InOut[0]* are changed to FALSE. The value of the Carry Flag (CY) changes to FALSE.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

ROL and ROR

These instructions rotate a bit string by one or more bits.

ROL: Rotates the bit string to the left (toward the higher bits).

ROR: Rotates the bit string to the right (toward the lower bits).

Instruction	Name	FB/FUN	Graphic expression	ST expression
ROL	Rotate N-bits Left	FUN		Out:=ROL(In, Num);
ROR	Rotate N-bits Right	FUN		Out:=ROR(In, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to rotate	Input	Data to rotate	Depends on data type.	---	*
Num	Number of bits		Number of bits to rotate	0 to No. of bits in *In*	Bits	1
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK														
Out	Must be same data type as *In*																			

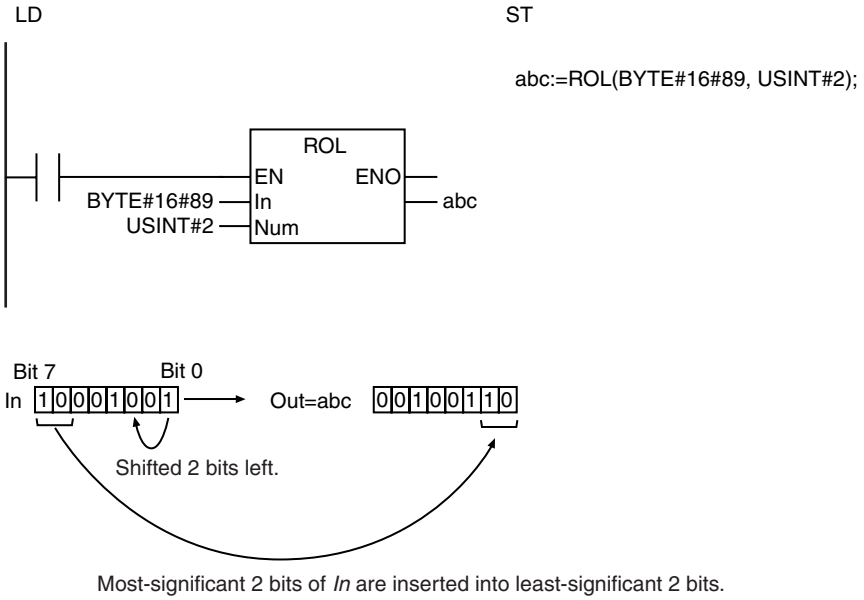
Function

These instructions rotate data to rotate *In* (bit string data) by the number of bits specified in number of bits *Num*. Bits that are shifted out of the register are inserted into the other end of the register.

● ROL

The ROL instruction rotates bits from right to left (from least-significant to most-significant bits).

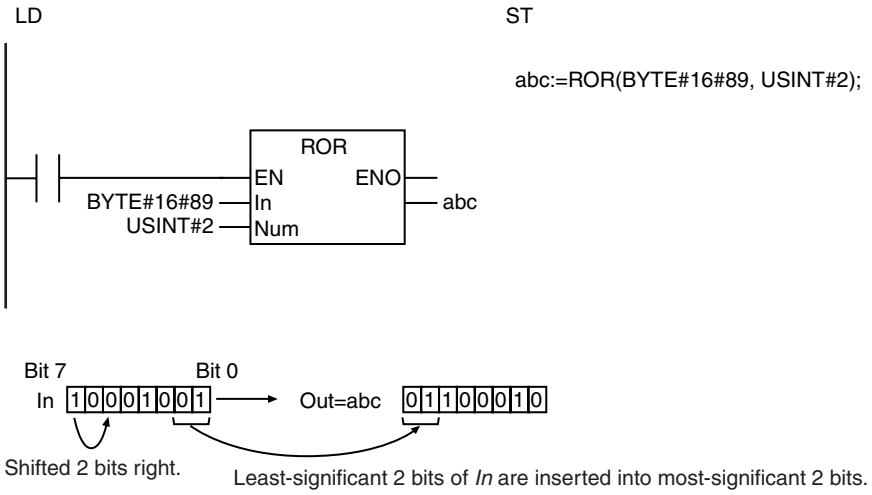
The following example is for when *In* is BYTE#16#89 and *Num* is USINT#2.



- **ROR**

The ROR instruction rotates bits from left to right (from most-significant to least-significant bits).

The following example shows the ROR instruction when *In* is BYTE#16#89 and *Num* is USINT#2.



Additional Information

The SHL and SHR instructions discard the bits that are shifted out of the register and insert zeros into the other end of the register.

Precautions for Correct Use

- The data types of *In* and *Out* must be the same.
- If *Num* is 0, an error will not occur and the value of *In* will be assigned directly to *Out*.
- If the value of *Num* exceeds the number of bits specified in *In*, an error will not occur and the bits will be rotated by the number of bits specified in *Num*. For example, if *In* is WORD data, the value of *Out* will be the same regardless of whether the value of *Num* is USINT#1 or USINT#17.

Conversion Instructions

Instruction	Name	Page	Instruction	Name	Page
Swap	Swap Bytes	2-368	FixNumToString	Fixed-decimal Number-to-Text String Conversion	2-428
Neg	Reverse Sign	2-369	StringToFixNum	Text String-to-Fixed-decimal Conversion	2-430
Decoder	Bit Decoder	2-371	DtToString	Date and Time-to-Text String Conversion	2-433
Encoder	Bit Encoder	2-374	DateToString	Date-to-Text String Conversion	2-435
BitCnt	Bit Counter	2-376	TodToString	Time of Day-to-Text String Conversion	2-436
ColmToLine_**	Column to Line Conversion Group	2-377	GrayToBin_** and BinToGray_**	Gray Code-to-Binary Code Conversion Group/ Binary Code-to-Gray Code Conversion	2-438
LineToColm	Line to Column Conversion	2-379	StringToAry	Text String-to-Array Conversion	2-441
Gray	Gray Code Conversion	2-381	AryToString	Array-to-Text String Conversion	2-443
PWLApprox	Broken Line Approximation	2-384	DispartDigit	Four-bit Separation	2-445
MovingAverage	Moving Average	2-387	UniteDigit_**	Four-bit Join Group	2-447
PIDAT	PID with Autotuning	2-393	Dispart8Bit	Byte Data Separation	2-449
DispartReal	Separate Mantissa and Exponent	2-418	Unite8Bit_**	Byte Data Join Group	2-451
UniteReal	Combine Real Number Mantissa and Exponent	2-421	ToAryByte	Conversion to Byte Array	2-453
NumToDecString and NumToHexString	Fixed-length Decimal Text String Conversion/ Fixed-length Hexadecimal Text String Conversion	2-423	AryByteTo	Conversion from Byte Array	2-458
HexStringToNum_**	Hexadecimal Text String-to-Number Conversion Group	2-426	SizeOfAry	Get Number of Array Elements	2-463

Swap

The Swap instruction swaps the upper byte and lower byte of a 16-bit value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Swap	Swap Bytes	FUN	(@)Swap EN ENO In Out	Out:=Swap(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

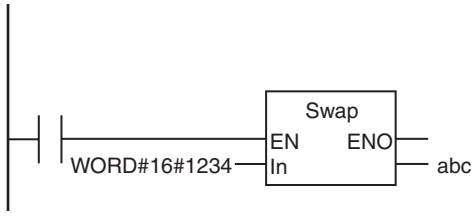
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			OK																	
Out			OK																	

Function

The Swap instruction swaps the upper byte and lower byte of data to convert *In* and assigns the result to conversion result *Out*.

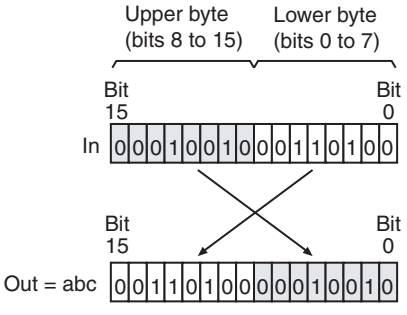
The following example is for when *In* is WORD#16#1234.

LD



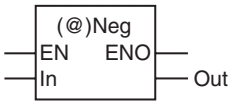
ST

```
abc:=Swap(WORD#16#1234);
```



Neg

The Neg instruction reverses the sign of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Neg	Reverse Sign	FUN		Out:=Neg(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

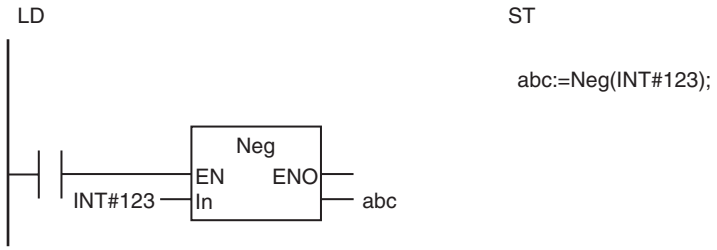
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

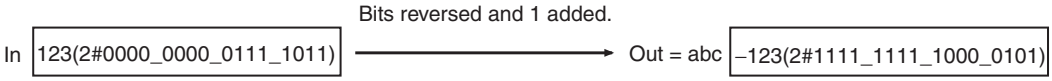
Function

The Neg instruction reverses the sign of data to convert *In*. The value of *Out* depends on the data type of *In*.

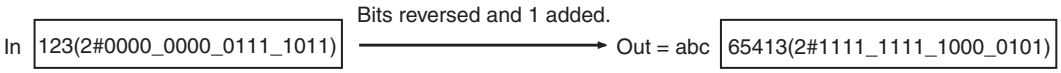
Data type of *In*	Value of *Out*
Signed integer: SINT, INT, DINT, or LINT	All bits in *In* are reversed and then 1 is added. (This is the same as multiplying *In* by –1.)
Unsigned integers: USINT, UNIT, UDINT, or ULINT	All bits in *In* are reversed and then 1 is added.
Real numbers: REAL or LREAL	*In* × (–1)

The following example is for when *In* is INT#123.



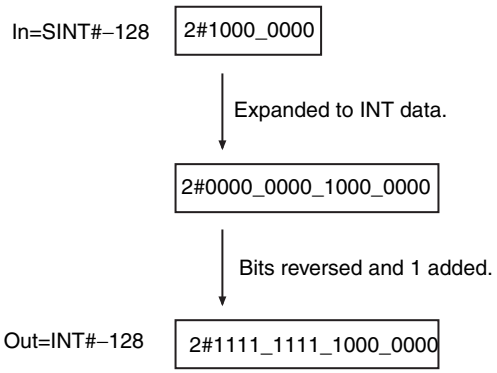


The following example is for when *In* is UINT#123.



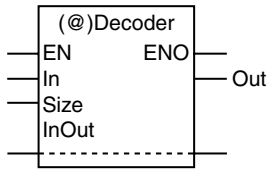
Precautions for Correct Use

If you use a different data type for *In* and *Out*, make sure the valid range of *Out* includes the valid range of *In.* Otherwise, an error will not occur and the value of *Out* will be an illegal value. For example, if the value of *In* is SINT#-128 and the data type of *Out* is INT, the value of *Out* will be INT#-128 instead of INT#128.



Decoder

The Decoder instruction sets the specified bit to TRUE and the other bits to FALSE in array elements that consist of a maximum of 256 bits.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Decoder	Bit Decoder	FUN		Decoder(In, Size, InOut);

OLL

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Conversion bit position	Input	Bit position to convert	Depends on data type.	---	0
Size	Bits to convert		Number of bits to convert	0 to 8	Bits	1
InOut[] (array)	Array to convert	In-out	Array to convert	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

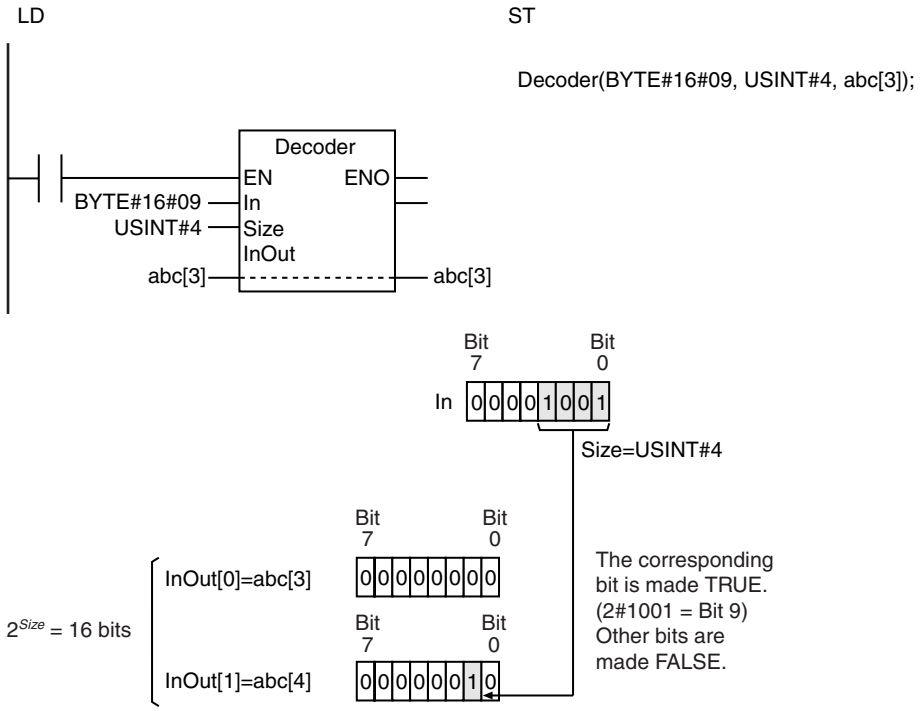
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK																		
Size						OK														
InOut[] (array)	OK	OK	OK	OK	OK															
Out	OK																			

Function

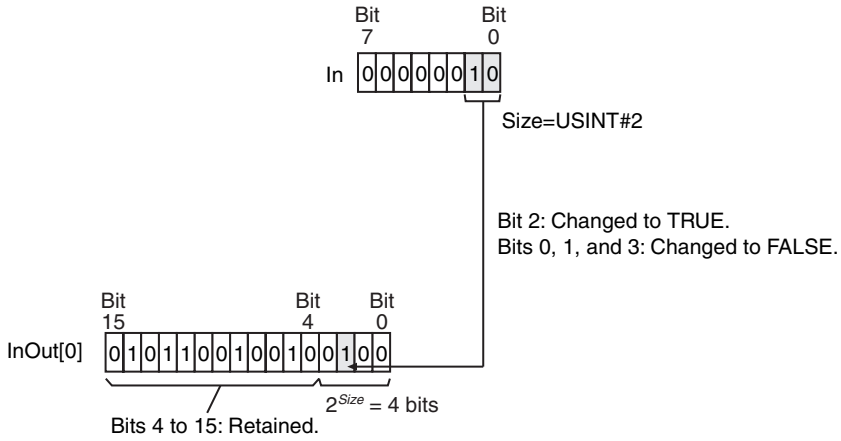
The Decoder instruction converts array elements with 2^{Size} bits that start from *InOut[0]* in array to convert *InOut[]*. It sets the specified bit to TRUE. It sets the other bits to FALSE. The bit to make TRUE is specified by the *Size* bits in the lower byte of conversion bit position *In*. Always attach the element number to the in-out parameter that is passed to *InOut[]*, e.g., *array[3]*.

Consider an example where *In* is BYTE#16#09, *Size* is USINT#4, and *InOut[]* is a BYTE array. The value of the *Size* bits in the lower bits of *In* is 16#9, which is 9 decimal. Therefore, the ninth bit from the least-significant bit of *InOut[]* is made TRUE and the other bits are made FALSE.

InOut[] is a BYTE array, so the ninth bit from the least-significant bit is bit 1 in *InOut[1]*. Therefore, bit 1 in *InOut[1]* is made TRUE, all other bits in *InOut[1]* are made FALSE, and all bits in *InOut[0]* are made FALSE.



If the number of bits in the elements of *InOut[]* is larger than the number of bits specified by *Size*, the values of the remaining bits are retained. Consider an example where *In* is BYTE#16#02, *Size* is USINT#2, and *InOut[]* is a WORD array.
Size is USINT#2, so the value is set in the lower 4 bits of *InOut[0]*. The values of the remaining bits in *InOut[0]* (bits 4 to 15) are retained.



Additional Information

Use the Encoder instruction (page 2-374) to find the position of the highest TRUE bit in array elements that consist of a maximum of 256 bits.

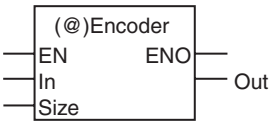
Precautions for Correct Use

- If the value of *Size* is 0, all bits in *InOut[]* change to FALSE.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.

- The value of *Size* is outside of the valid range.
- The value of *Size* exceeds the array area of *InOut[]*.
- *InOut[]* is not a BOOL array or an array of bit strings.
- An array without a subscript is passed to *InOut[]*.

Encoder

The Encoder instruction finds the position of the highest TRUE bit in array elements that consist of a maximum of 256 bits.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Encoder	Bit Encoder	FUN		Out:=Encoder(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to convert	Input	Array to convert	Depends on data type.	---	*
Size	Bits to convert		Number of bits to convert	0 to 8	Bits	1
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK															
Size						OK														
Out		OK																		

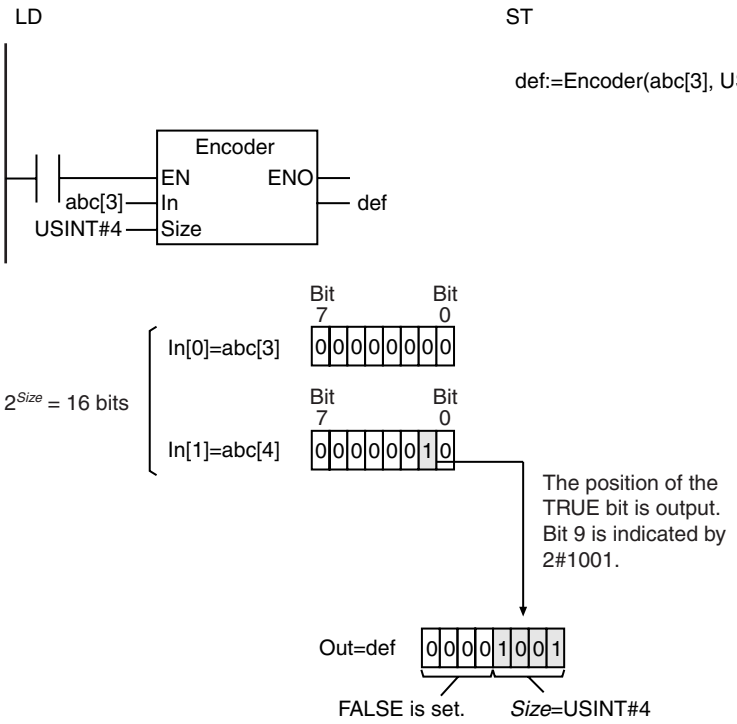
Function

The Encoder instruction finds the position of a TRUE bit in a specified range of bits in array to convert *In[]*. The instruction looks for a TRUE bit in 2^{Size} bits from *In[0]*. The position of the TRUE bit in this range is expressed in binary and stored in the *Size* bits in the lower bits of conversion result *Out*. FALSE is stored in the remaining bits of *Out*.

If there is more than one TRUE bit in the specified range, the position of the highest bit that is TRUE is found. Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

Consider an example for when *Size* is USINT#4 and *In[]* is a BYTE array. *Size* is USINT#4, so the range in which to find a TRUE bit is 2^4, or 16 bits, from *In[0]*. In the following diagram, the ninth bit in the range is TRUE.

Size is USINT#4, so 2#1001 (i.e., 9) is stored in the lower 4 bits of *Out*. FALSE is stored in the upper four bits of *Out*.



Additional Information

Use the Decoder instruction (page 2-371) to make one bit TRUE and the other bits FALSE in array elements that consist of a maximum of 256 bits.

Precautions for Correct Use

- If the value of *Size* is 0, all bits in *Out* change to FALSE.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - The value of all bits in *In[]* that are specified by *Size* change to FALSE.
  - *In[]* is not a BOOL array or an array of bit strings.
  - An array without a subscript is passed to *In[]*.

BitCnt

The BitCnt instruction counts the number of TRUE bits in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BitCnt	Bit Counter	FUN	(@)BitCnt EN ENO In Out	Out:=BitCnt(In);

Variables

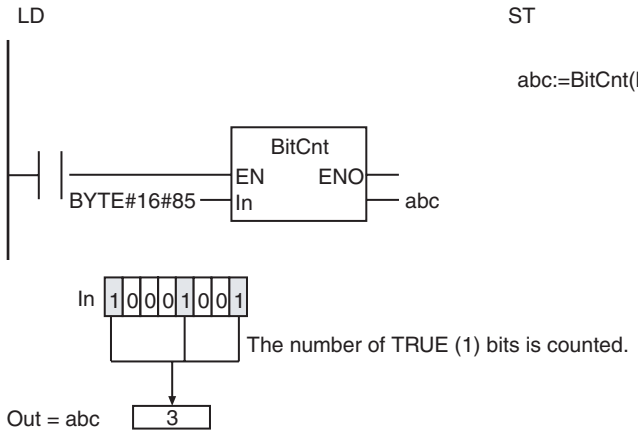
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Count string	Input	String in which to count TRUE bits	Depends on data type.	---	*
Out	Count result	Output	Number of TRUE bits	0 to No. of bits in *In*	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK														

Function

The BitCnt instruction counts the number of TRUE bits in count string *In*. The following example is for when *In* is BYTE data with a value of BYTE#16#85.



ColmToLine_**

The ColmToLine_** instruction extracts bit values from the specified position of array elements and outputs them as a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ColmToLine_**	Column to Line Conversion Group	FUN	(@)ColmToLine_** EN ENO In Out Size Pos "**" must be a bit string data type.	Out:=ColmToLine_**(In, Size, Pos); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to convert	Input	Array to convert	Depends on data type.	---	*
Size	Number of elements to convert		Number of elements in *In[]* to convert	0 to No. of bits in *Out*		1
Pos	Bit position to convert		Bit position to convert	0 to No. of bits in *In[]* – 1		0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK															
Size						OK														
Pos						OK														
Out		OK	OK	OK	OK															

Function

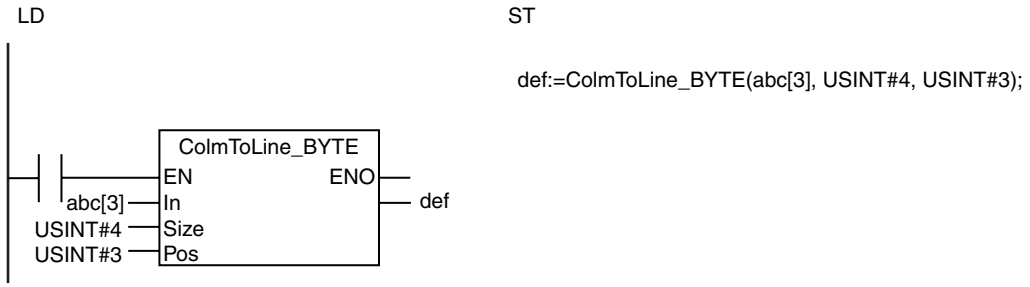
The ColmToLine_** instruction extracts bit values from the specified position of array elements and outputs them in order as a bit string.

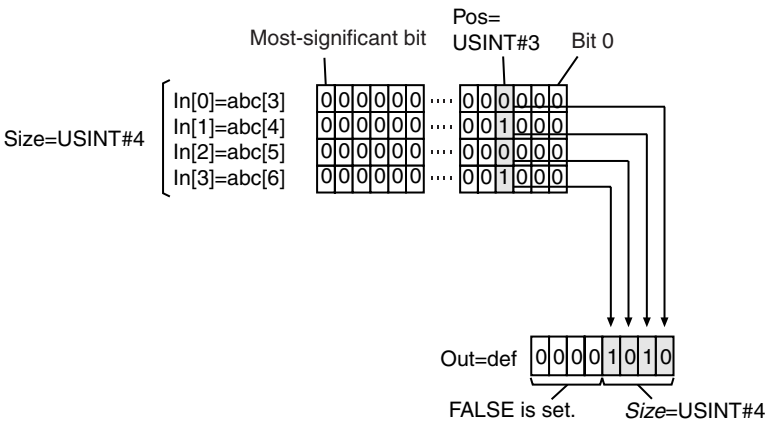
First, *Size* elements are extracted from array to convert *In[]* starting from *In[0]*. Then, only the values of bits in *Pos* are extracted. These are placed in order in a bit string of *Size* bits and stored in conversion result *Out* from the least-significant bit. FALSE is stored in the remaining bits of *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the BYTE data type, the instruction is ColmToLine_BYTE.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

The following example shows the ColmToLine_BYTE instruction when *Pos* is USINT#3 and *Size* is USINT#4.





Additional Information

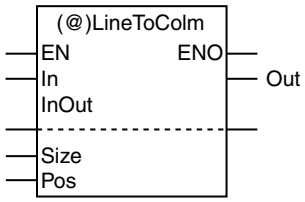
Use the LineToColm instruction (page 2-379) to output a bit string to the specified bit position in array elements.

Precautions for Correct Use

- If the value of *Size* is 0, all bits in *Out* change to FALSE.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - *In[]* is not an array of bit strings.
  - An array without a subscript is passed to *In[]*.

LineToColm

The LineToColm instruction takes the bits from a bit string and outputs them to the specified bit position in array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LineToColm	Line to Column Conversion	FUN		LineToColm(In, InOut, Size, Pos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Size	Number of elements in result		Number of elements in result	0 to No. of bits in *In*		1
Pos	Conversion bit position		Bit position to receive the conversion	0 to No. of bits in *InOut[]* – 1		0
InOut[] (array)	Conversion result array	In-out	Conversion result	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

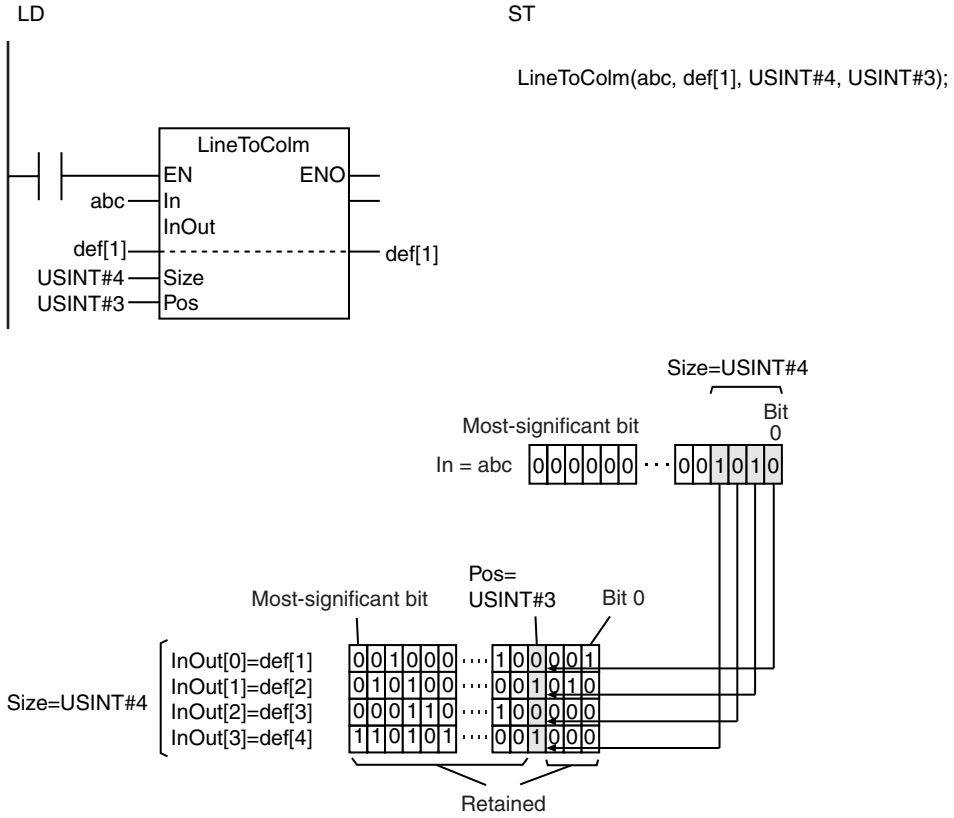
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Size						OK														
Pos						OK														
InOut[] (array)		OK	OK	OK	OK															
Out	OK																			

Function

The LineToColm instruction takes the bits from a bit string and outputs them to the specified bit position in array elements.

First, *Size* bits are extracted from the least-significant bit of data to convert *In*. These bits are treated individually. Then, the bits are stored in conversion result array *InOut[]* in the *Pos* bit of the elements starting from *InOut[0]*. *Size* specifies the number of array elements to receive bits. The values of all bits for which values are not stored are retained.

The following example is for when *Pos* is USINT#3 and *Size* is USINT#4.



Additional Information

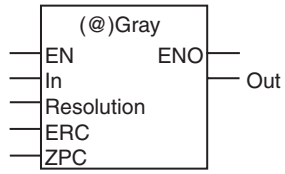
Use the ColmToLine_** instruction (page 2-377) to extract bit values from the specified position of array elements and output them as a bit string.

Precautions for Correct Use

- If the value of *Size* is 0, the values in *InOut[]* do not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *InOut[]* is not an array of bit strings.
  - An array without a subscript is passed to *InOut[]*.

Gray

The Gray instruction converts a gray code into an angle.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Gray	Gray Code Conversion	FUN		Out:=Gray(In, Resolution, ERC, ZPC);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Gray code to convert	Depends on data type.	---	0
Resolution	Resolution		Resolution	_R256, _R1B to _R15B, _R360, _R720, or _R1024		_R256
ERC	Encoder remainder correction		Encoder remainder correction	0 to *Resolution*		0
ZPC	Zero point correction		Zero point correction			
Out	Conversion result	Output	Conversion result	*	°	---

* 0 to 359.999999999999

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			OK																	
Resolution	Refer to *Function* for the enumerators of the enumerated type _eGRY_RESOLUTION.																			
ERC							OK													
ZPC							OK													
Out															OK					

Function

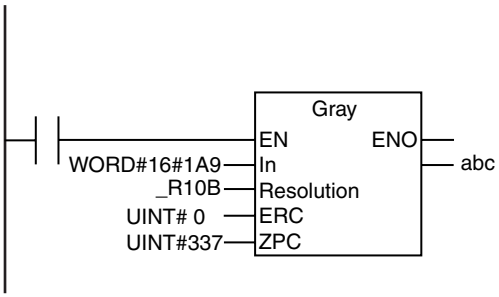
The Gray instruction converts the gray code in *In* to an integer, compensates (decreases) it by encoder remainder correction *ERC* and zero point correction *ZPC*. It then converts the result to an angle according to resolution *Resolution*. The conversion result *Out* will be an angle in degrees.

The data type of *Resolution* is enumerated type _eGRY_RESOLUTION. The meaning of the enumerators are as follows:

Enumerator	Meaning
_R256	256
_R1B	1-bit (2)
_R2B	2-bit (4)
_R3B	3-bit (8)
_R4B	4-bit (16)
_R5B	5-bit (32)
_R6B	6-bit (64)
_R7B	7-bit (128)
_R8B	8-bit (256)
_R9B	9-bit (512)
_R10B	10-bit (1024)
_R11B	11-bit (2048)
_R12B	12-bit (4096)
_R13B	13-bit (8192)
_R14B	14-bit (16384)
_R15B	15-bit (32768)
_R360	360
_R720	720
_R1024	1024

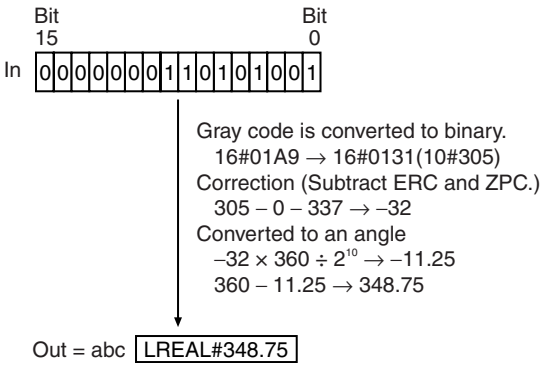
The following example is for when *In* is WORD#16#1A9, *Resolution* is _R10B, *ERC* is UINT#0, and *ZPC* is UINT#337. The value of *Out* will be LREAL#348.75.

LD



ST

```
abc:=Gray(WORD#16#1A9, _R10B, UINT#0, UINT#337);
```



Additional Information

Refer to the user documentation for your rotary encoder for the values to specify for *Resolution* and *ERC*.

Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *Resolution* is outside of the valid range.
- The value of *ERC* exceeds the resolution that is specified in *Resolution*.
- The value of Z*PC* exceeds the resolution that is specified in *Resolution*.
- *In*, when converted to a bit string, is smaller than the value of *ERC*.
- The value of the bit string after correction for *ERC* exceeds the resolution that is specified in *Resolution*.

PWLApprox

The PWLApprox instruction performs broken line approximations for integer or real number data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
PWLApprox	Broken Line Approximation	FUN	(@)PWLApprox EN ENO In Out Line Num	Out:=PWLApprox(In, Line, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Line[] (array)	Broken line data array		Broken line data array			
Num	Number of broken line data		Number of broken line data			1
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

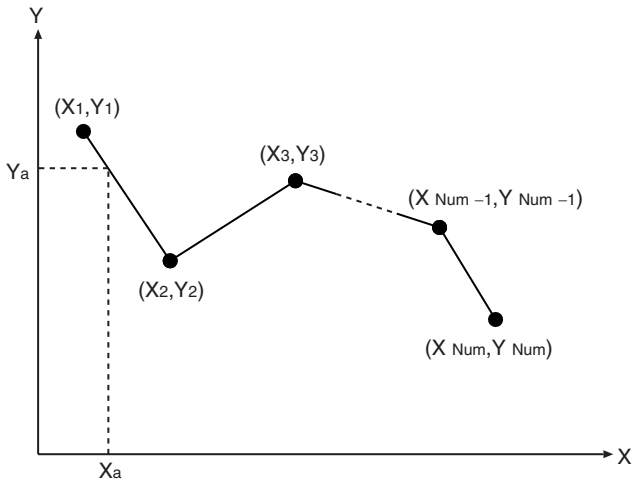
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Line[] (array)	Must be an array with elements that have the same data type as *In*.																			
Num							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The PWLApprox instruction performs approximation for data to convert *In.* The approximation is based on broken line data that consists of *Num* times 2 elements that start with *Line[0][0]* in broken line data array *Line[]*.

Line[] must be a two-dimensional or three-dimensional array. Set the number of elements for the first dimension to 2. Set the number of elements for the second dimension to 2. Set the array elements from *Line[0,0]* to X_1, Y_1, X_2, Y_2, etc., as shown in the following figure.

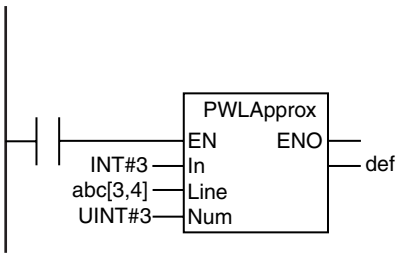
The instruction will find the Ya that corresponds to Xa on the broken line graph.



Always attach the element numbers to the input parameter that is passed to *Line[]*, e.g., *array[3,4]*.

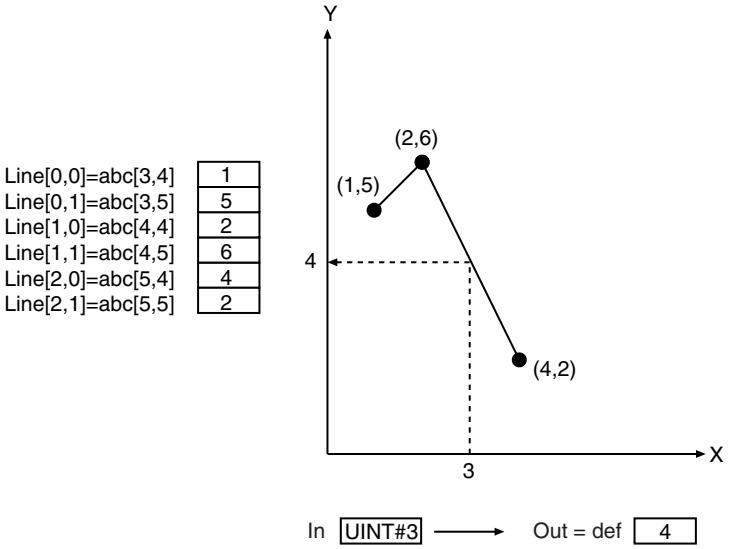
The following example is for when *Num* is UINT#3, *In* is INT#3, (X_1,Y_1) is (1,5), (X_2,Y_2) is (2,6), and (X_3,Y_3) is (4,2).

LD



ST

```
def:=PWLApprox(INT#3, abc[3,4], UINT#3);
```



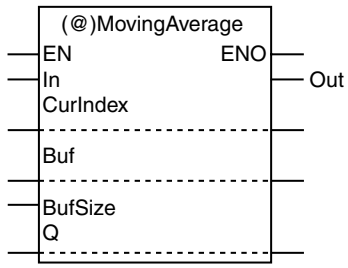
Precautions for Correct Use

- If the value of *In* is smaller than the value of *Line[0,0]* (i.e., the value of X_1), then the value of *Out* will be the value of *Line[0,1]* (i.e., the value of Y_1).

- If the value of *In* is larger than the value of *Line[Num−1,0]* (i.e., the value of X_{Num}), then the value of *Out* will be the value of *Line[Num−1,1]* (i.e., the value of Y_{Num}).
- *Line[]* must be a two-dimensional or three-dimensional array. Set the number of elements for the first dimension to 2.
- If the value of *Num* is 0, the value of *Out* is 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Num* exceeds the array area of *Line[]*.
  - The broken line data does not meet this requirement: $X_1 < X_2 < ... < X_{Num}$.
  - *In* and *Line[]* are REAL data and their values are nonnumeric data or infinity.

MovingAverage

The MovingAverage instruction calculates a moving average.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MovingAverage	Moving Average	FUN		Out:=MovingAverage(In, CurIndex, Buf, BufSize, Q);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Input value	Input	Number to include in average	Depends on data type.	---	*
BufSize	Maximum number stored		Maximum number of elements to include in average			1
CurIndex	Input value storage position	In-out	Position in *Buf[]* in which to store *In*	Depends on data type.	---	---
Buf[] (array)	Input value storage array		Array to store *In* values			
Q	Calculation completed flag		TRUE: *BufSize* elements or more have been stored in *Buf[]* FALSE: *BufSize* elements are not yet stored in *Buf[]*			
Out	Calculation result	Output	Calculation result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied.A building error will occur.

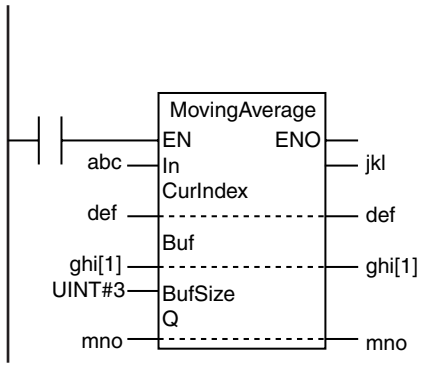
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
BufSize							OK													
CurIndex							OK													
Buf[] (array)	Must be an array with elements that have the same data type as *In*.																			
Q	OK																			
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The MovingAverage instruction stores the value of input value *In* in input value storage array *Buf[]* each time it is executed. It stores the average of the stored values in calculation result *Out*. Specify the maximum number of elements to include in the average with *BufSize*.

The processing procedure when *BufSize* is UINT#3 is described below as an example. The instruction and statement are written as follows:

LD

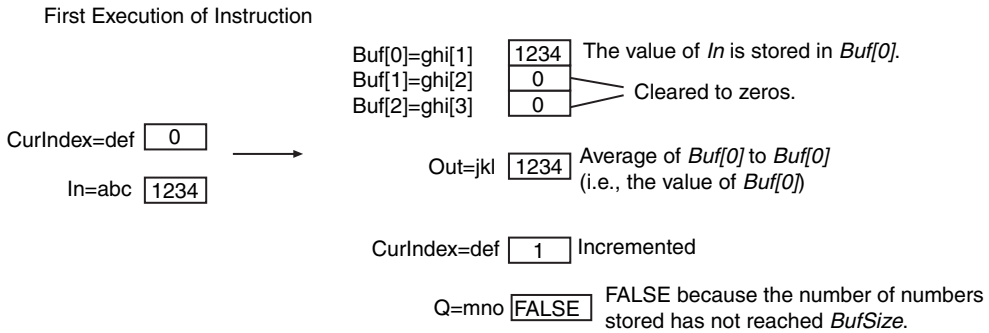


ST

```
jkl:=MovingAverage(abc, def, ghi[1], UINT#3, mno);
```

First Time a Number Is Input

The input value storage position *CurIndex* is set to 0 and the instruction is executed.
Buf[0] to *Buf[BufSize−1]* of input value storage array *Buf[]* are cleared to zeros and the first input value *In* is stored in *Buf[0]*.
The value of calculation completed flag *Q* changes to FALSE. This indicates that the number of values that are stored in *Buf[]* has not reached *BufSize* yet.
While the value of *Q* is FALSE, the average value is calculated for the *CurIndex* + 1 numbers that start from *Buf[0]*. The calculation result is stored in *Out*.
Finally, the value of *CurIndex* is incremented.

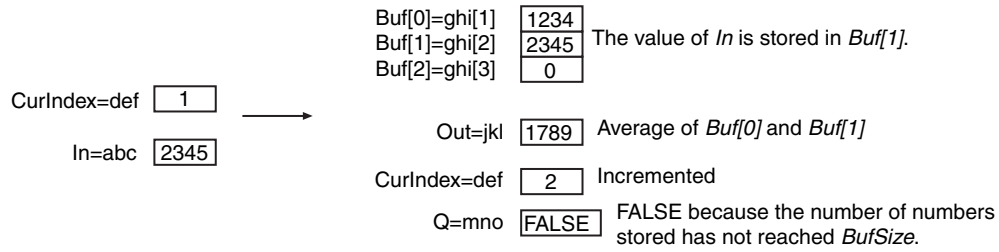


Inputting Numbers Up to *BufSize*

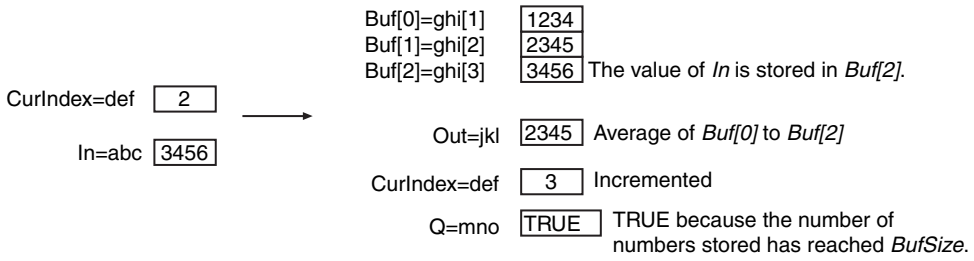
Each time the instruction is executed, the value of *In* is stored in *Buf[CurIndex]*. The average of *CurIndex* + 1 numbers that start from *Buf[0]* is calculated and stored in *Out*.

When the number of instruction executions reaches *BufSize*, the value of *Q* changes to TRUE.

Second Execution of Instruction



Third Execution of Instruction

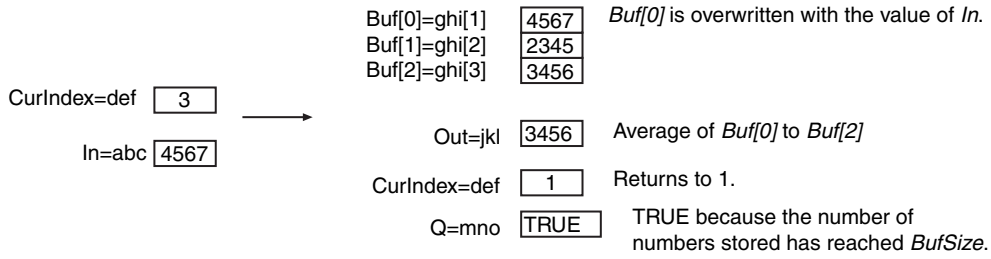


Inputting Numbers after Reaching *BufSize*

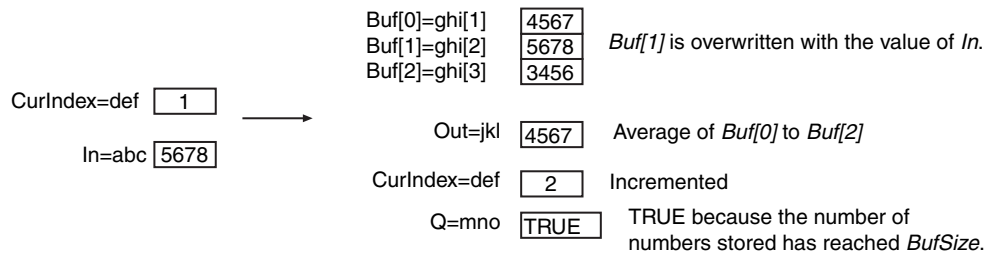
Each time the instruction is executed, *Buf[0]* to *Buf[BufSize−1]* are overwritten with the value of *In* in cyclic fashion. The average of *Buf[0]* to *Buf[BufSize−1]* is calculated and stored in *Out*.

The value of *CurIndex* returns to 1 after it reaches *BufSize* and it is then incremented again. The value of *Q* remains TRUE.

Fourth Execution of Instruction



Fifth Execution of Instruction



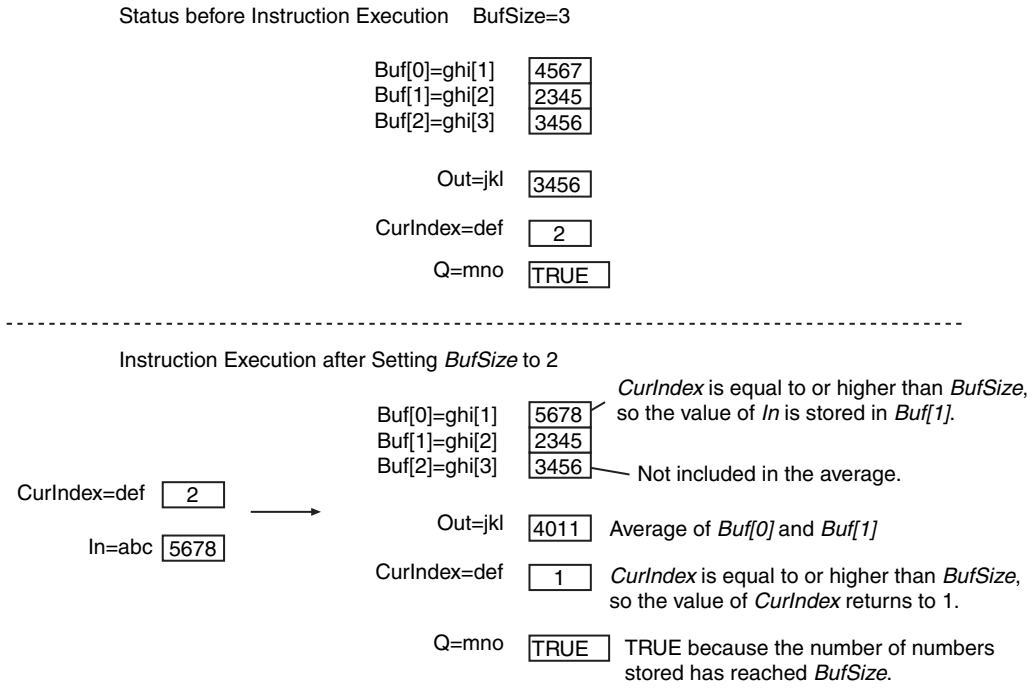
Initializing the Stored Values

If the value of *CurIndex* is set to 0 before the instruction is executed, the values in *Buf[0]* to *Buf[BufSize−1]* are set to 0 and the current value of *In* is stored again in *Buf[0]*.

The value of *CurIndex* changes to 1 and the value of *Q* changes to FALSE.

Changing the Value of *BufSize*

If you change the value of *BufSize* and execute the instruction, operation is performed with the new value of *BufSize* and the current value of *CurIndex*.



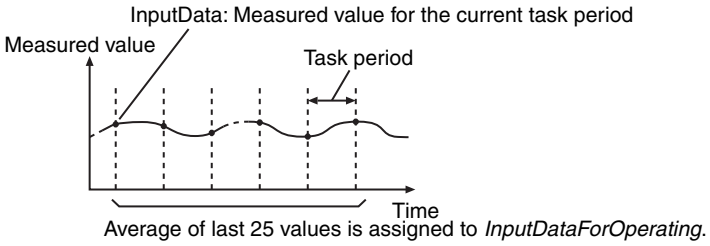
Precautions for Correct Use

- Use the same data type for *In* and the elements of *Buf[]*.
- Use a *Buf[]* array that is at least as large as the value of *BufSize.*
- Even if the calculation result exceeds the valid range of *Out*, an error will not occur. The value of *Out* will be an illegal value.
- If the value of *BufSize* is 0, the values of *Out* and *CurIndex* change to 0. The value of *Q* changes to TRUE.
- If you change the value of *BufSize*, always set the value of *CurIndex* to 0 and initialize the stored values.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* and *Buf[]* are different data types.
  - The value of *BufSize* is outside of the valid range.
  - The value of *BufSize* exceeds the size of the *Buf[]* array.
  - *Buf[]* is not an integer array.
  - An array without a subscript is passed to *Buf[]*.

Sample Programming

This sample shows how to eliminate the effect of noise and other disturbances in analog input data, e.g., from a sensor. It assigns the average (*DataAve*) of the last 25 values of the input data (*InputData*) to the input data (*InputDataForOperating*) for the next process.
InputData is input every task period as long as the value of the execution condition (*Trigger*) is TRUE. Until 25 values of *InputData* are input, there is not enough data to calculate the average, so the most recent value of *InputData* is assigned to *InputDataForOperating*.

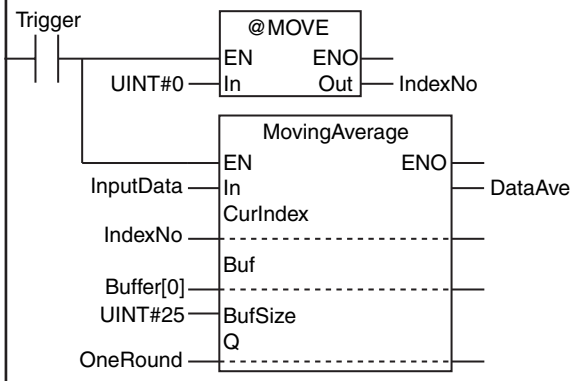
When the value of *Trigger* changes to TRUE, the average is cleared and input of *InputData* is started again from the beginning.



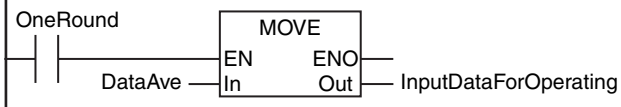
LD

Variable	Data type	Initial value	Comment
Trigger	BOOL	False	Execution condition
InputData	INT	10	Input value
Buffer	ARRAY[0..24] OF INT	[25(0)]	Input value storage array
DataAve	INT	0	Average value
OneRound	BOOL	False	Flag that indicates 25 inputs
IndexNo	UINT	0	Input value storage position
InputDataForOperating	INT	0	Input to next operation

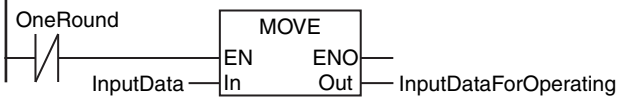
When *Trigger* changes to TRUE, 0 is assigned to *IndexNo*.
While *Trigger* is TRUE, the value of *InputData* is input every task period and the average is calculated.



When there are 25 or more input values for *InputData* , *DataAve* is assigned to *InputDataForOperating*.



Until there are 25 or more input values for *InputData* , *InputData* is assigned to *InputDataForOperating*.



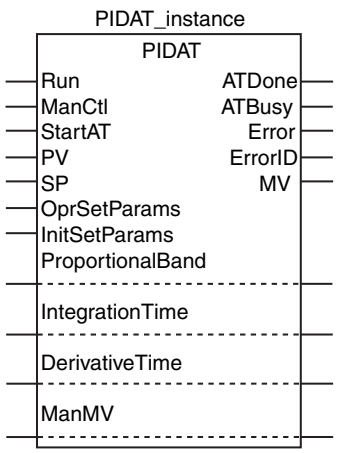
ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	False	Execution condition
LastTrigger	BOOL	False	Value of *Trigger* from previous task period
Operating	BOOL	False	Processing
OperatingStart	BOOL	False	Processing started
Buffer	ARRAY[0..24] OF INT	[25(0)]	Input value storage array
InputData	INT	10	Input value
DataAve	INT	0	Average value
OneRound	BOOL	False	Flag that indicates 25 inputs
IndexNo	UINT	0	Input value storage position
InputDataForOperating	INT	0	Input to next operation

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// Clear the average.
IF (OperatingStart=TRUE) THEN
    IndexNo:=UINT#0;
    OperatingStart:=FALSE;
END_IF;

// Calculate the moving average.
IF (Operating=TRUE) THEN
    DataAve:=MovingAverage(
        In          :=InputData,
        CurIndex:=IndexNo,
        Buf         :=Buffer[0],
        BufSize  :=UINT#25,
        Q           :=OneRound);

    IF (OneRound=TRUE) THEN
        // Assign the average of last 25 values to InputDataForOperating.
        InputDataForOperating:=DataAve;
    ELSE
        // Assign the most recent value to InputDataForOperating.
        InputDataForOperating:=InputData;
    END_IF;
END_IF;

// End average processing.
IF (Trigger=FALSE) THEN
    Operating:=FALSE;
END_IF;
```

PIDAT

The PIDAT instruction performs PID control with autotuning (2-PID control with set point filter).

Instruction	Name	FB/FUN	Graphic expression	ST expression
PIDAT	PID Control with Autotuning	FB		PIDAT_instance(Run, ManCtl, StartAT, PV, SP, OprSetParams, InitSetParams, ProportionalBand, IntegrationTime, DerivativeTime, ManMV, ATDone, ATBusy, Error, ErrorID, MV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Run	Execution condition	Input	TRUE: Execute FALSE: Stop	Depends on data type.	---	FALSE
ManCtl	Manual/auto control		TRUE: Manual operation FALSE: Automatic operation			
StartAT	Autotuning execution condition		TRUE: Execute FALSE: Cancel			
PV	Process value		Process value	*1		
SP	Set point		Set point			
OprSet Params	Operation setting parameters		Parameters set during operation	---		0
InitSet Params	Initial setting parameters		Initial setting parameters			---
Proportional Band	Proportional band	In-out	Proportional band	0.01 to 1000.00	% FS	---
IntegrationTime	Integration time		Integration time The higher the value is, the weaker the integral action is. No integral action is performed for 0.	T#0.0000s to T#10000.0000s*2	s	
DerivativeTime	Derivative time		Derivative time The higher the value is, the stronger the derivative action is. No derivative action is performed for 0.	T#0.0000s to T#10000.0000s*2		
ManMV	Manual manipulated variable		Manual manipulated variable	–320 to 320	%	

Name	Meaning	I/O	Description	Valid range	Unit	Default
ATDone	Autotuning normal completion	Output	TRUE: Normal completion FALSE: *3	Depends on data type.	---	---
ATBusy	Autotuning busy		TRUE: Autotuning FALSE: Not autotuning			
MV	Manipulated variable		Manipulated variable	−320 to 320	%	

*1 Value of input range lower limit *InitSetParams.RngLowLmt* to Value of input range upper limit *InitSetParams.RngUpLmt*

*2 Digits below 0.0001 s are truncated.

*3 FALSE indicates an error end, that PID control is in progress without autotuning, or that PID control is not in progress.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Run	OK																			
ManCtl	OK																			
StartAT	OK																			
PV														OK						
SP														OK						
OprSet Params	Refer to *Function* for details on the structure _sOPR_SET_PARAMS.																			
InitSet Params	Refer to *Function* for details on the structure _sINIT_SET_PARAMS.																			
Proportional Band														OK						
IntegrationTime																OK				
DerivativeTime																OK				
ManMV														OK						
ATDone	OK																			
ATBusy	OK																			
MV														OK						

Function

The PIDAT instruction performs PID control of a manipulated variable for a temperature controller or other device. PID control is started when the value of execution condition *Run* changes to TRUE. While the value of *Run* is TRUE, the following process is repeated periodically: process value *PV* is read, PID processing is performed, and manipulated variable *MV* is output. PID control is stopped when the value of *Run* changes to FALSE.

Autotuning is supported to automatically find the optimum PID constants. When the value of the autotuning execution condition *StartAT* changes to TRUE, the PID constants are autotuned.

Structure Specifications

The data type of operation setting parameter *OprSetParams* is structure _sOPR_SET_PARAMS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
OprSetParams	Operation Setting Parameters	Parameters that are set during operation.	_sOPR_SET_PARAMS	---	---	---
MVLowLmt	MV Lower Limit	The lower limit of the MV.	REAL	–320 to 320*	%	0
MVUpLmt	MV Upper Limit	The upper limit of the MV.	REAL			100
ManResetVal	Manual Reset Value	The value of *MV* when the deviation is 0 for the proportional action.	REAL	–320 to 320		0
MVTrackSw	MV Tracking Switch	TRUE: ON FALSE: OFF	BOOL	Depends on data type.	---	FALSE
MVTrackVal	MV Tracking Value	The value that is set in *MV* during MV tracking.	REAL	–320 to 320	%	0
StopMV	Stop MV	The value that is set in *MV* when instruction execution is stopped.	REAL			
ErrorMV	Error MV	The value that is set in *MV* when an error occurs.	REAL			
Alpha	2-PID Parameter α	The set point filter is disabled if the set point filter coefficient α is 0.	REAL	0.00 to 1.00	---	0.65
ATCalcGain	Autotuning Calculation Gain	Adjustment coefficient from autotuning results. Stability is given higher priority with higher values. The speed of response is given higher priority with lower values.	REAL	0.1 to 10.0		1.0
ATHystrs	Autotuning Hysteresis	The hysteresis of the limit cycle.	REAL		% FS	0.2

* *MVLowLmt* must be less than *MVUpLmt*.

The data type of initial setting parameter *InitSetParams* is structure _sINIT_SET_PARAMS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
InitSetParams	Initial Setting Parameters	Initial setting parameters.	_sINIT_SET_PARAMS	---	---	---
SampTime	Sampling Period	The period for PID processing.	TIME	T#0.0001s to #100.0000s	s	T#0.1s
RngLowLmt	Lower Limit of Input Range	The lower limit of *PV* and *SP*.	REAL	–32000 to 32000*	---	0
RngUpLmt	Upper Limit of Input Range	The upper limit of *PV* and *SP*.	REAL			100
DirOpr	Action Direction	TRUE: Forward action FALSE: Reverse action	BOOL	Depends on data type.		FALSE

* *RngLowLmt* must be less than *RngUpLmt*.

Meanings of Variables

The meanings of the variables that are used in this command are described below.

● *Run* (Execution Condition)

This is the execution condition for the instruction. PID control is performed while the value is TRUE. PID control is stopped when the value changes to FALSE.

● *ManCtl* (Manual/Auto Control)

This instruction can be executed in one of two modes: Manual operation or automatic operation. The value of *ManCtl* determines which mode is used.

Value of *ManCtl*	Operation mode	Value of *MV*
TRUE	Manual	Value of *ManMV* (PID control is not performed.)
FALSE	Automatic	Value that is calculated for PID control

● *StartAT* (Autotuning Execution Condition)

This is the execution condition for autotuning the PID constants. If the value of *StartAT* is TRUE when the value of *Run* changes to TRUE, autotuning is performed when PID control is started. If the value of *StartAT* changes to TRUE during PID control (i.e., when the value of *Run* is TRUE), autotuning is performed during PID control. In either case, autotuning is canceled if the value of *StartAT* changes to FALSE during autotuning. Autotuning is described in more detail later.

● *PV* (Process Value)

This is the process value of the controlled system.

● *SP* (Set Point)

This is the set point for the controlled system.

● *MVLowLmt* (MV Lower Limit) and *MVUpLmt* (MV Upper Limit)

You can limit the value of *MV*. *MVLowLmt* and *MVUpLmt* are the lower and upper limits to *MV*. *MVLowLmt* must always be less than *MVUpLmt*.

MV from PID processing	Value of *MV*
Less than *MVLowLmt*	MvLowLmt
Between *MVLowLmt* and *MVUpLmt*, inclusive	*MV* from PID processing
Greater than *MVUpLmt*	MvUpLmt

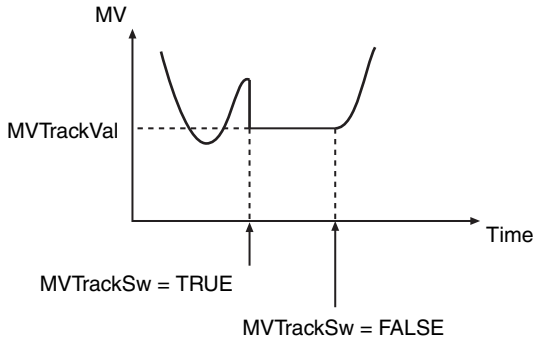
If stop MV *StopMV*, error MV *ErrorMV*, or manual MV *ManMV* is set in manipulated variable *MV*, limit control is not applied.

● *ManResetVal* (Manual Reset Value)

This is the value of *MV* when the deviation (i.e., the difference between *PV* and *SP*) is 0 for the proportional action. The value of *ManResetVal* determines the location of the proportional action band.

● *MVTrackSw* (MV Tracking Switch)

MV tracking is a function that sets the *MV* to an external input value (called the MV tracking value) during automatic operation. MV tracking is performed while the value of *MVTrackSw* is TRUE. When the value of *MVTrackSw* changes to FALSE, the value of MV returns to the result of PID processing. The value of MV is changed smoothly at this time (bumpless).



- ***MVTrackVal* (MV Tracking Value)**

  This is the value to which *MV* is set during MV tracking. The value of *MVTrackVal* does not have to be between *MVLowLmt* and *MVUpLmt*.

- ***StopMV* (Stop MV)**

  This is the value to which *MV* is set when the value of *Run* is FALSE (i.e., when execution of this instruction is stopped).

- ***ErrorMV* (Error MV)**

  This is the value to which *MV* is set when an error occurs (i.e., when the value of *Err* is TRUE). If the value of *ErrorMV* is not within the valid range (–320 to 320), the value of *MV* will be 0 when an error occurs.

- ***Alpha* (2-PID Parameter α)**

  This parameter determines the coefficient of the set point filter. Refer to the description in *2-PID Control with Set Point Filter* for details. Normally set the value of *Alpha* to 0.65.

- ***AtCalcGain* (Autotuning Calculation Gain)**

  This variable gives the coefficient of the PID constants that were calculated by autotuning when they are applied to the actual PID constants. If a value of 1.00 is specified, the results of autotuning are used directly. Increase the value of ATCalcGain to give priority to stability and decrease it to give priority to response.

- ***ATHystrs* (Autotuning Hysteresis)**

  This is the hysteresis that is used in the limit cycle for autotuning. More accurate tuning is achieved if the value of *ATHystrs* is small. However, if the process value is not stable and proper autotuning is difficult, increase the value. Refer to the description of autotuning for details.

- ***SampTime* (Sampling Period)**

  This is the minimum value of the period for PID processing. Refer to the description of the execution timing of PID processing for details. PID processing is not performed again until the time specified for *SampTime* has elapsed since the last time PID processing was performed.

- ***RngLowLmt* (Lower Limit of Input Range) and *RngUpLmt* (Upper Limit of Input Range)**

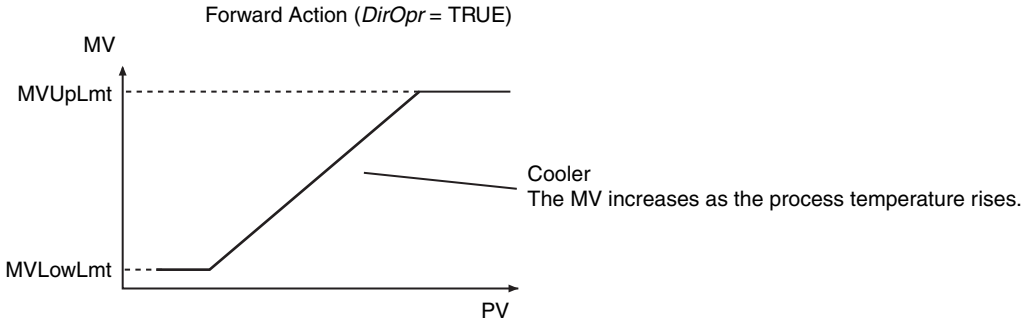
  These are the lower limit and upper limit of *PV* and *SP*. An error will occur if the value of the parameter connected to *PV* or *SP* exceeds either of these limits. *RngLowLmt* must always be less than *RngUpLmt*.

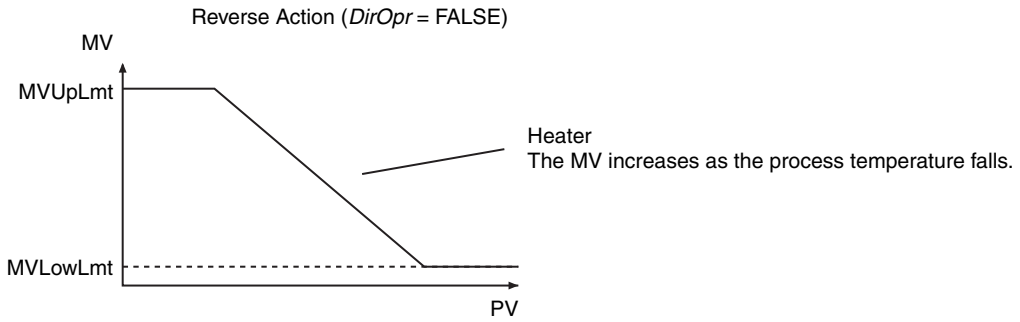
● *DirOpr* (Action Direction)

This variable specifies if *MV* is increased or decreased for changes in the value of *PV*. These are called a forward action and a reverse action.

Value of *DirOpr*	Meaning	Value of *MV*
TRUE	Forward action	Increases with the value of *PV*.
FALSE	Reverse action	Decrease with the value of *PV*.

The difference between a forward action and reverse action are described here for temperature control. A forward action is used to control the MV for a cooler. That is, the higher the process temperature, the larger the MV of the cooler must be. On the other hand, a reverse action is used to control the MV for a heater. That is, the lower the process temperature, the larger the MV of the heater must be.





● *ProportionalBand* (Proportional Band)

This is one of the three PID constants. Refer to the description of the proportional action for details. The larger the *ProportionalBand* is, the greater the offset is. Hunting occurs if the *ProportionalBand* is too small.

● *IntegrationTime* (Integration Time)

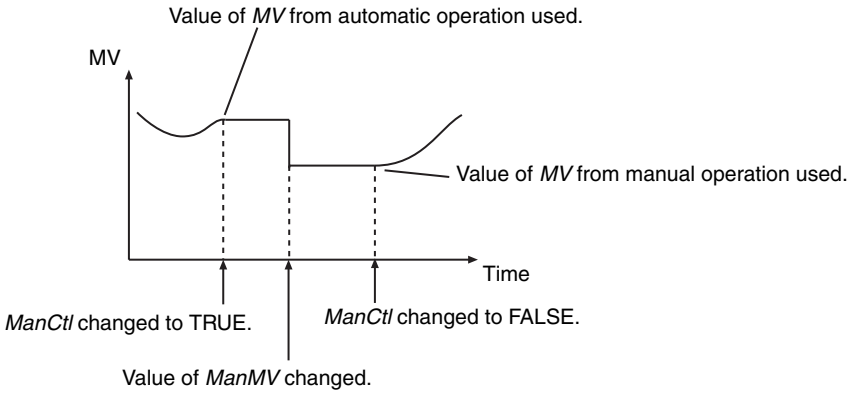
This is one of the three PID constants. Refer to the description of the integral action for details. The larger the value of *IntegrationTime* is, the weaker the integral action is.

● *DerivativeTime* (Derivative Time)

This is one of the three PID constants. Refer to the description of the derivative action for details. The larger the value of *DerivativeTime* is, the stronger the derivative action is.

● *ManMV* (Manual Manipulated Variable)

MV is set to this value during manual operation (while *ManCtl* is TRUE). However, immediately after changing from automatic to manual operation, the value of MV from automatic operation is used. *MV* is set to the value of *ManMV* only when it changes after operation switches to manual operation. When operation changes from manual to automatic operation, the value of MV from manual operation is used. The value of *ManMV* does not have to be between *MVLowLmt* and *MVUpLmt*.



● *ATDone* (Autotuning Normal Completion)

This flag indicates when autotuning was completed normally. It changes to TRUE when autotuning is completed normally and remains TRUE as long as the value of *StartAT* is TRUE. It is FALSE in the following cases.

- An autotuning error end occurred.
- Autotuning is in progress (i.e., while the value of *ATBusy* is TRUE).
- PID control is in progress without autotuning.
- PID control is not in progress (i.e., the value of *Run* is FALSE).
- The value of *StartAT* is FALSE.

● *ATBusy* (Autotuning Busy)

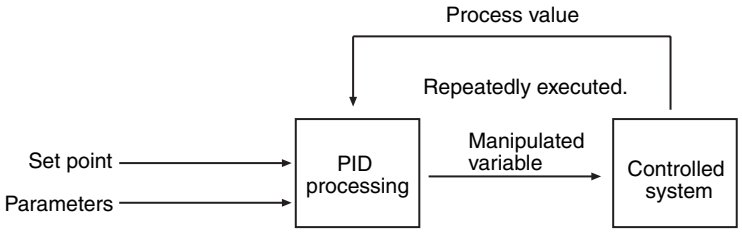
This flag indicates when autotuning is in progress. It is TRUE while autotuning is in progress. Otherwise it is FALSE.

● MV (Manipulated Variable)

This is the manipulated variable that is applied to the controlled system.

Introduction to PID Control

PID control is a feedback control method that repeatedly measures the process value of the controlled system and calculates a manipulated variable so that the process value approaches a set point. This instruction therefore outputs a manipulated variable for the following inputs: process value, set point, and calculation parameters. PID control periodically measures the process value, calculates the manipulated variable, and outputs the manipulated variable so that the process value approaches the set point.

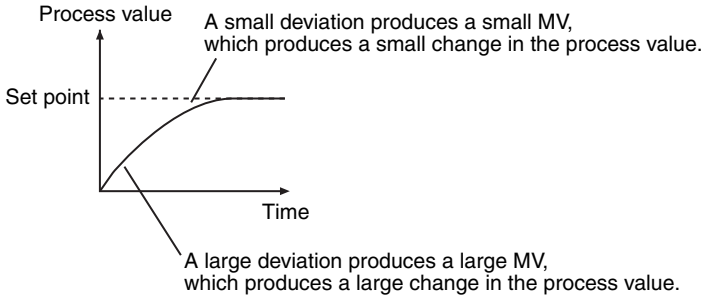


Proportional (P), Integral (I), and Derivative (D) Actions

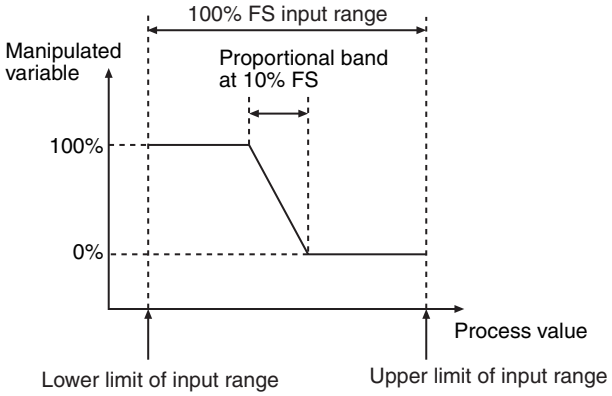
PID control is performed by combining the proportional action, integral action, and derivative action. These actions are described next.

● Proportional Action (P)

The proportional action increases the absolute value of the manipulated variable in proportion to the deviation between the process value and the set point. The process value of the controlled system changes as shown below.

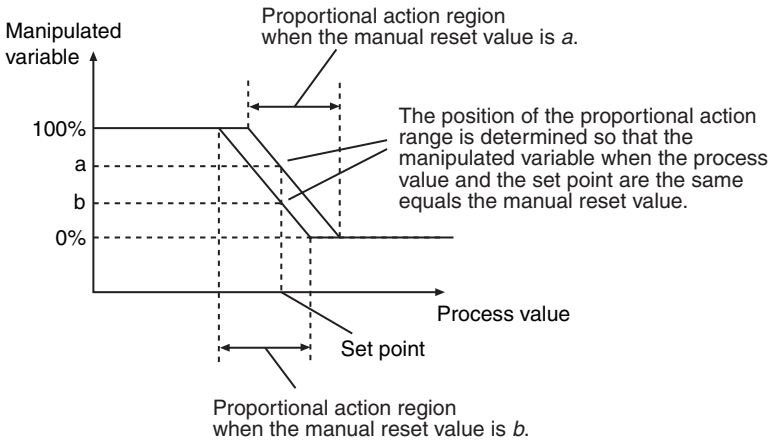


The proportional band is one of the settings that are used for the proportional action. The proportional band is the range of the process value to which the proportional action is applied. If the process value is not in the proportional band, the manipulated variable is set to 100% or 0%.
The proportional band is expressed as the percentage of the input range in which to perform the proportional action (% FS). The following diagram shows the proportional band set to 10% FS.

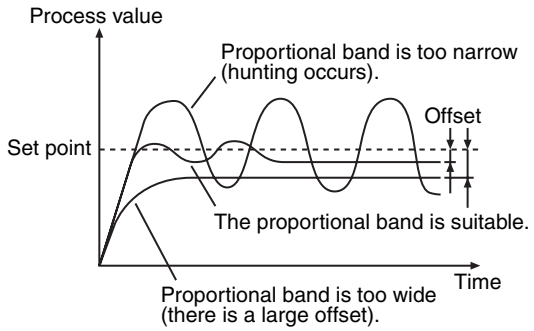


Another parameter for the proportional action is the manual reset value. The manual reset value is the manipulated variable that is used when the deviation is 0. The manual reset value determines the position of the proportional action range in the process value–manipulated variable graph. The relationship between the manual reset value and the proportional action region is shown below.

The position of the proportional action range is determined so that the manipulated variable when the process value and the set point are the same equals the manual reset value.

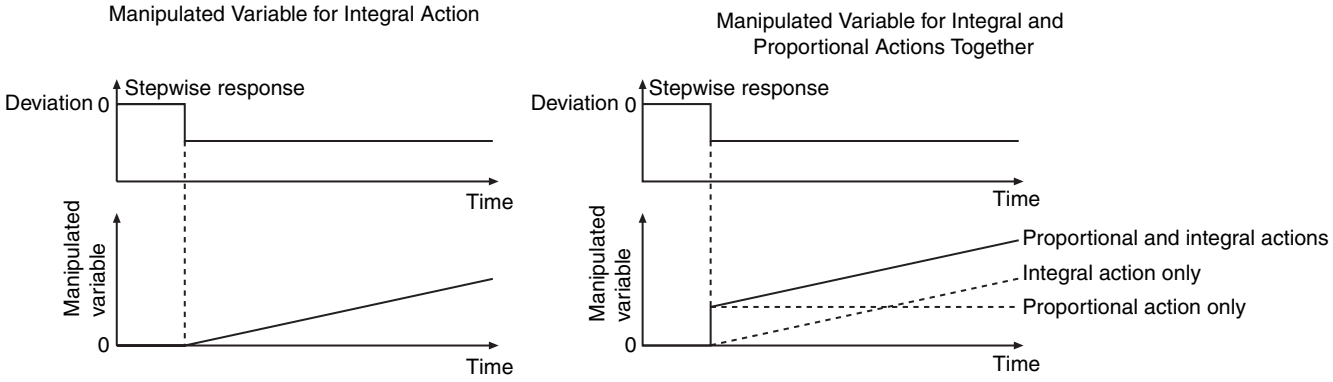


If the manual reset value is not suitable, the deviation will never reach 0. The remaining deviation is called the offset or the residual deviation. You can make the proportional band narrower to reduce the offset. If the proportional band is too narrow, the process value will not stop at the set point. This is called overshooting. If the process value does not stabilize and oscillates around the set point, it is called hunting.

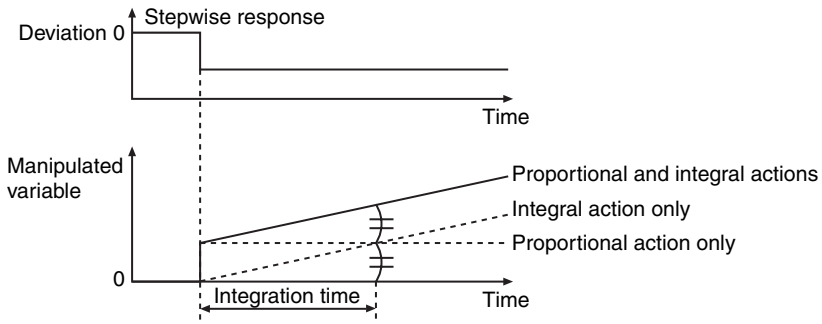


● Integral Action (I)

Very accurate adjustment of the proportional band and manual reset value is required to bring the offset to 0 with only the proportional action. Also, the size of the offset varies with the disturbance, so it is necessary to repeat the adjustment frequently. To simplify the operation, an integral action is used in combination with the proportional action. The integral action integrates the deviation on the time axis and then increases the absolute value of the manipulated variable in proportion to the result. When normal distribution operation is performed, the manual reset value is ignored. The following graph on the left shows changes in the manipulated variable for the integral action when a deviation occurs in stepwise fashion. The following graph on the right shows changes in the manipulated variable when the integral and proportional actions are combined.

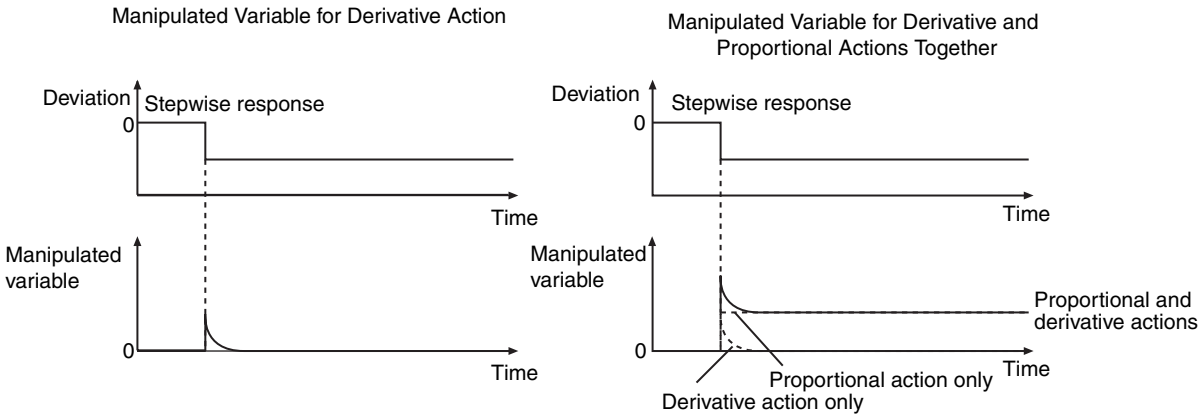


One of the parameters for the integral action is the integration time. This is the time for the manipulated variable from the integral action to equal the manipulated variable from the proportional action when a stepwise deviation occurs. The shorter the integration time is, the stronger the integral action is. A short integration time reduces the time for the offset to reach 0, but it can also cause hunting.

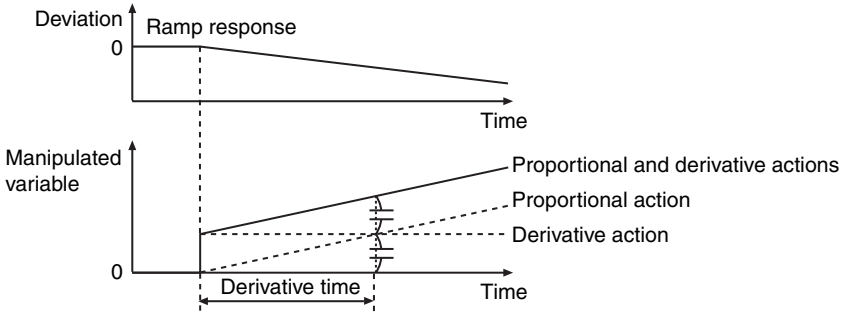


● Derivative Action (D)

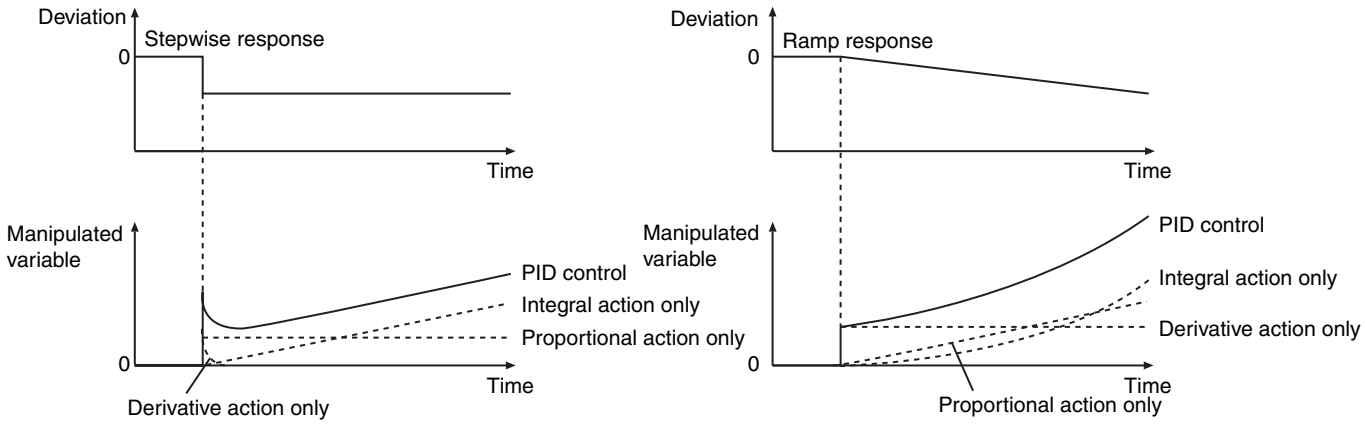
If the proportional and integral actions are used together, the offset will reach 0 and the process value will reach the set point. However, if disturbance causes the process value to change quickly, time is required to restore the original state. The derivative action functions to quickly return the process value to the set point when there is a disturbance. The derivative action differentiates the deviation on the time axis and then increases the absolute value of the manipulated variable in proportion to the result. In other words, the larger the change in the process value is, the larger the absolute value of the manipulated variable for the derivative action is. The changes in the manipulated variable for the derivative action when a deviation occurs in stepwise fashion are shown below. The changes in the manipulated variable when the derivative and proportional actions are combined are also shown.



One of the parameters for the derivative action is the derivative time. This is the time for the manipulated variable from the derivative action to equal the manipulated variable from the proportional action when a ramp deviation occurs. The longer the derivative time is, the stronger the derivative action is. A long derivative time provides a rapid response to disturbances, but it can also cause hunting.



The total of the manipulated variables for the proportional, integral, and derivative actions is the manipulated variable for PID control. The changes in the manipulated variable for PID control for a stepwise and ramp deviations are shown below.

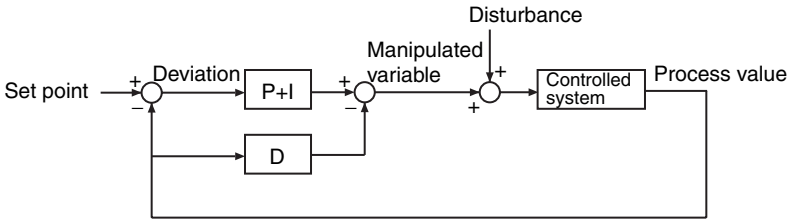


2-PID Control with Set Point Filter

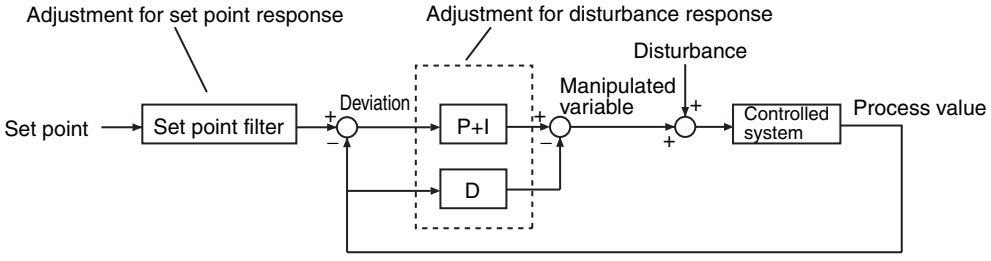
There are three main parameters that you must adjust to perform PID control: the proportional band, integration time, and derivative time. These are called the PID constants. The values of the PID constants affect the following two performances of PID control.

- Set point response: The ability to follow changes in the set point.
- Disturbance response: The ability of correcting the process value for large changes that are caused by disturbances

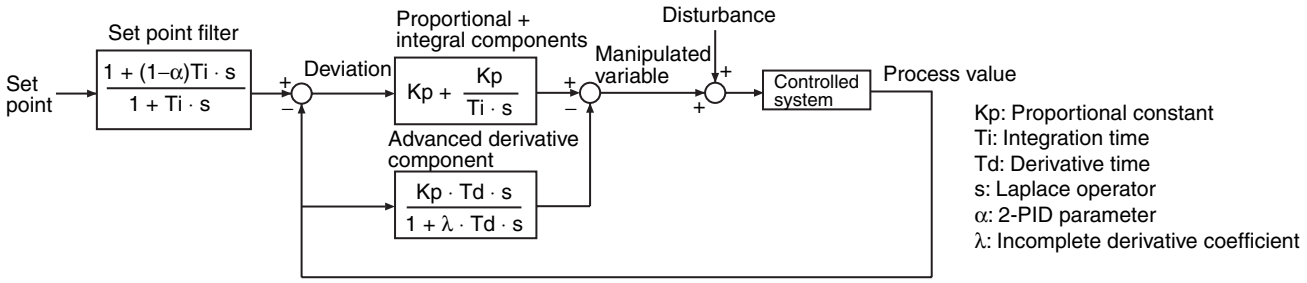
A block diagram for basic PID control is shown below. The set point and disturbance enter the block diagram at different positions. Therefore, finding the optimum PID constants for both set point response performance and disturbance response performance is difficult. In other words, if the PID constants are set for set point response, response to disturbances is slow. If the PID constants are set for disturbance response, overshooting occurs.



To enable both set point response and disturbance response, 2-PID control is used. The 2 in "2-PID" indicates that there are separate parameters to adjust the set point response and the disturbance response. A block diagram for this is shown below. A set point filter that includes an adjustment parameter is added. The PID constants are adjusted to maximize disturbance response. A set point filter adjusts the set point to optimize the set value response for those values. You can adjust the values of the PID constants and the set value of the set point filter independently to increase both the set point response and the disturbance response.



The formulas of the blocks of this instruction are shown below. The set point filter value (i.e., a coefficient for the set point) is adjusted by using the integration time and the 2-PID parameter α. The optimum value of α is 0.65. It normally does not need to be changed. The lower the value of α is, the smaller the influence of the set point filter is.



Starting PID Control

You must use suitable PID constants to execute this instruction. There are the following two ways to achieve this.

● When Suitable PID Constants Are Not Known

Perform autotuning at the start of operation to find suitable PID constants. Change the value of *Run* to TRUE while the value of *StartAT* is TRUE. First, autotuning is executed, and then PID control is started with the PID constants that are found.

● When Suitable PID Constants Are Known

Set suitable PID constants in *ProportinalBand*, *IntegrationTime*, and *DerivativeTime*, and then change *Run* to TRUE. *ProportinalBand*, *IntegrationTime*, and *DerivativeTime* are in-out variables. You cannot set constants for the input parameters. Always define suitable variables, and then assign the values to input parameters.

You can change the PID constants during operation. You can also perform autotuning during operation. To start autotuning during operation, change the value of *StartAT* to TRUE.

Control Status and Manipulated Variable

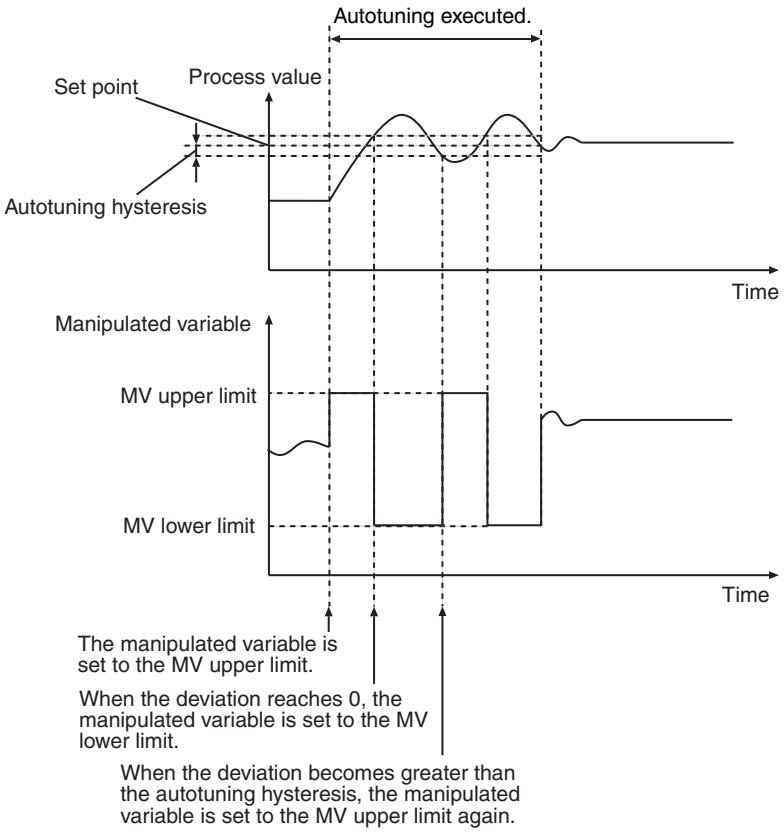
Manipulated variable *MV* is determined according to the control status as shown in the following table.

Control status	Value of variable					Manipulated variable *MV*
	ManCtl **(manual/auto control)**	***Run*** **(execution condition)**	***Error*** **(error end)**	***ManTrackSw*** **(manual tracking switch)**	***ATBusy*** **(autotuning busy)**	
Error end	FALSE	TRUE	TRUE	---	FALSE	*ErrorMV* (error MV)
MV tracking during automatic operation			FALSE	TRUE		*MVTrackVal* (MV tracking value)
Autotuning during automatic operation				FALSE	TRUE	Value repeatedly changes between upper limit of MV and lower limit of MV.
Not autotuning during automatic operation					FALSE	Value calculated with current PID constants.
Instruction execution stopped		FALSE	---	---		*StopMV* (Stop MV)
Manual operation	TRUE	---				*ManMV* (manual manipulated variable)

Autotuning

The 2-PID parameter α is not adjusted very often, so the main parameters that are adjusted for this instruction are the PID constants. The PIDAT instruction supports autotuning of the PID constants. The limit cycle method is used for autotuning. With the limit cycle method, the manipulated variable is temporarily changed to the upper and lower limits of the manipulated variable to find the optimum PID constants based on the resulting changes in the process value. If autotuning is executed when the set point is greater than the process value, the manipulated variable is first set to the upper limit. When the deviation reaches 0, the manipulated variable is set to the lower limit. When the deviation becomes greater than the autotuning hysteresis, the manipulated variable is set to the upper limit again. This process is repeated twice to calculate the optimum PID constants.

If autotuning is executed when the set point is less than the process value, the manipulated variable is first set to the lower limit. Then, the optimum values for the PID constants are calculated with the procedure that is given above.



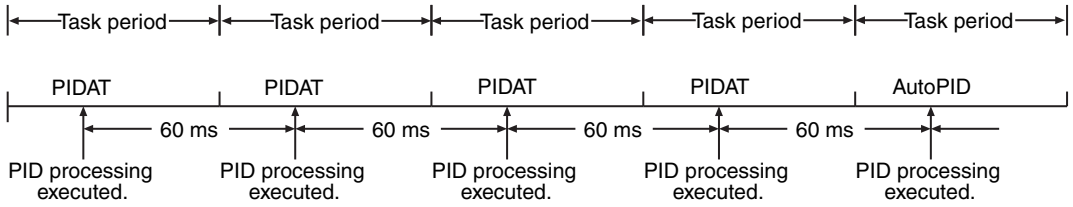
Autotuning is executed during PID control (i.e., when the value of *Run* is TRUE) if the value of *StartAT* changes to TRUE. If *StartAT* is TRUE when *Run* changes to TRUE, autotuning is executed at the start of PID control. When autotuning is completed normally, the calculated PID constants are used immediately. Autotuning is canceled if the value of *StartAT* changes to FALSE during autotuning (i.e., when *ATBusy* is TRUE). If autotuning is canceled, PID control is started again with the previous PID constants.

Execution Timing of PID Control

PID control is repeated periodically. PID processing is performed when the PIDAT instruction is executed in the user program. However, if sampling period *SampTime* has not elapsed since the last time PID processing was performed, PID processing is nor performed. If the elapsed time since the last time PID processing was executed exceeds *SampTime*, the excess time (elapsed time − *SampTime*) is carried forward to the next period. This is shown in the following diagram.

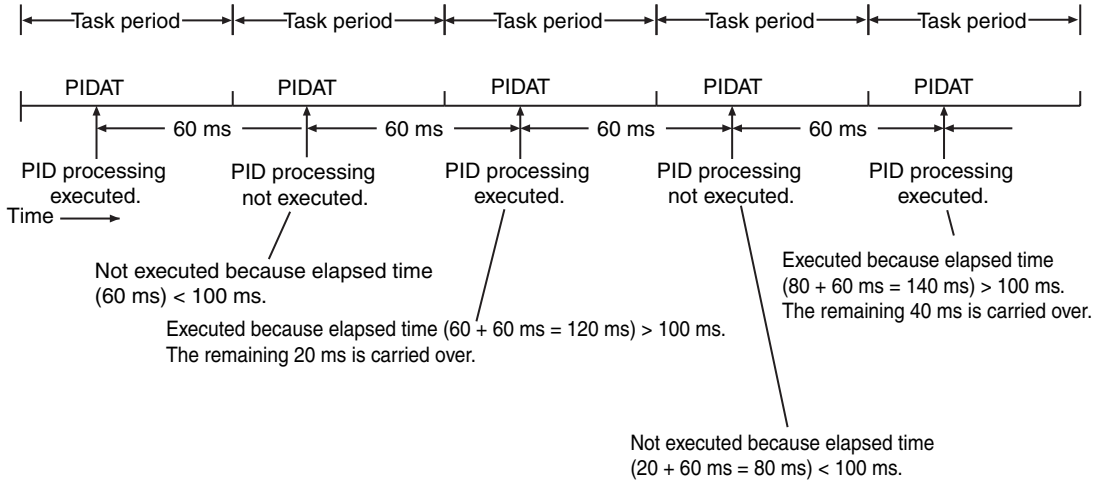
Task period = 60 ms and *SampTime* < 60 ms

The task period is greater than or equal to *SampTime*, so PID processing is executed once every task period.



Task period = 60 ms and *SampTime* = 100 ms

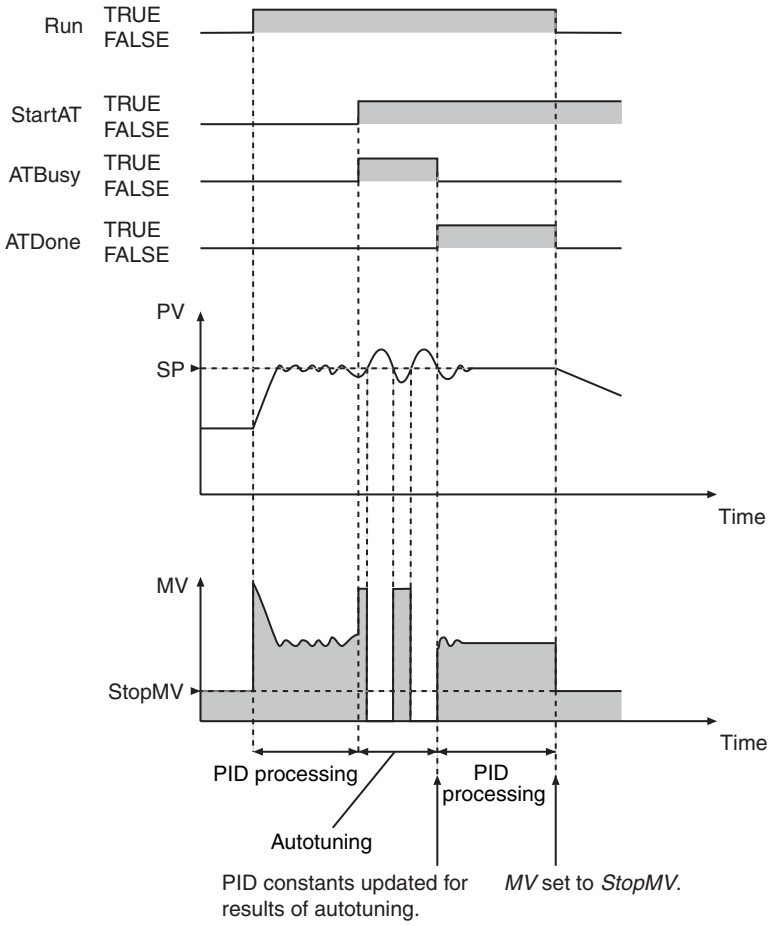
The task period is less than *SampTime*, so DIP processing is not executed every period.



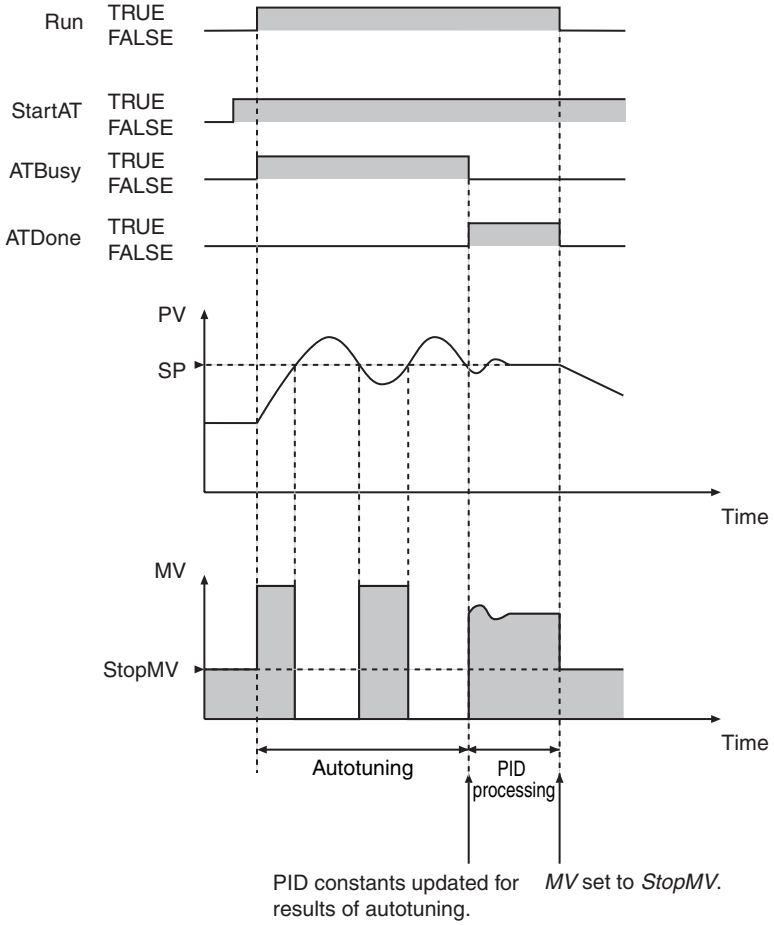
Timing Charts

Timing charts for the instruction variables are provided below for different situations.

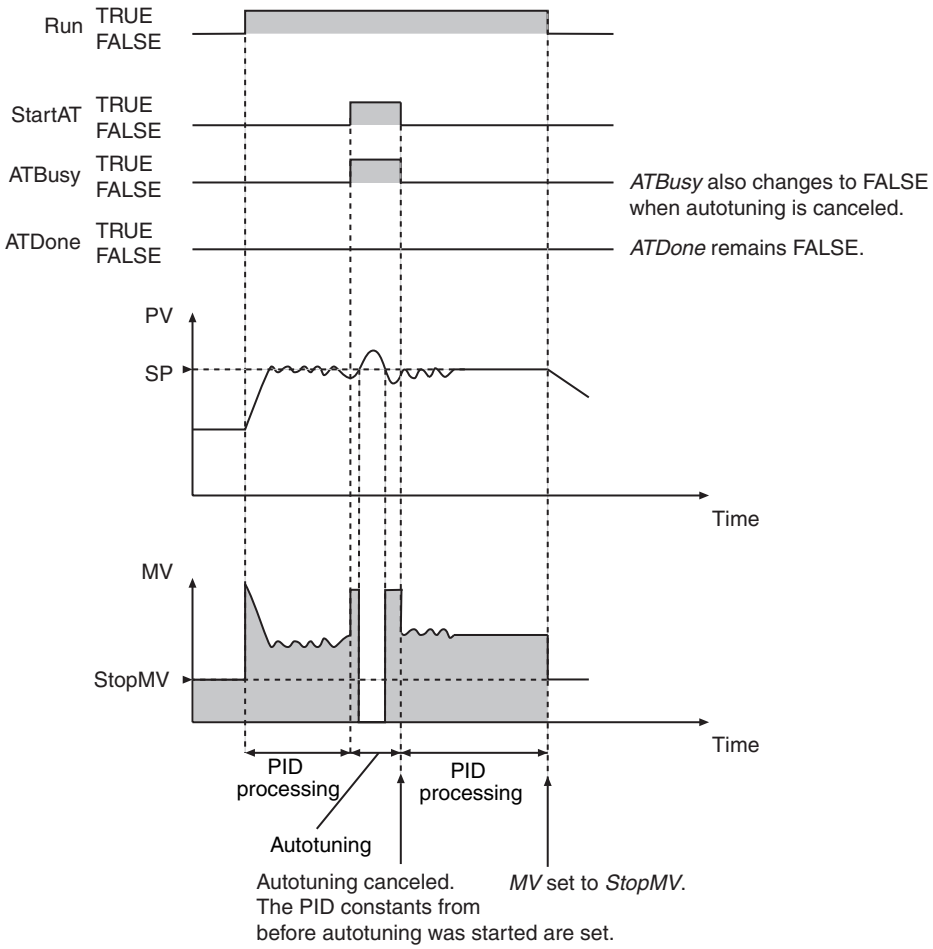
● Autotuning Executed during Automatic Operation



● Autotuning Executed at the Start of PIDAT Execution



● Autotuning Canceled

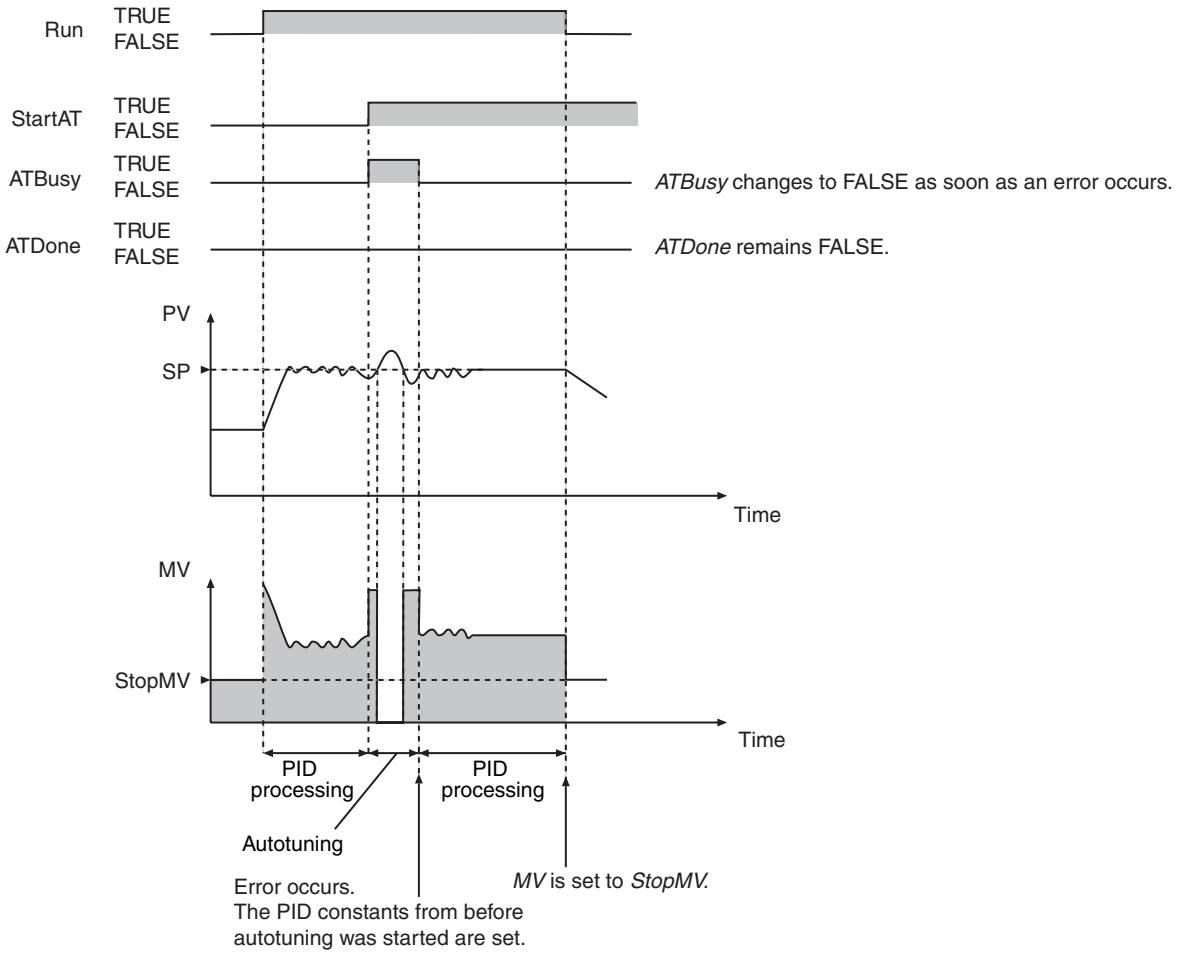


● An Autotuning Error Occurs during Autotuning

An autotuning error occurs and autotuning is stopped in the following cases.

- If the MV equals the MV upper limit and the time for the deviation to reach 0 exceeds 19,999 s.
- If the MV equals the MV lower limit and the time for the deviation to reach *AtHystrs* or higher exceeds 19,999 s.

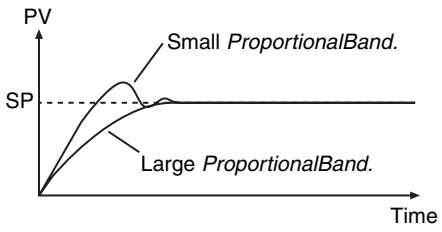
If autotuning is canceled, PID control is started again with the previous PID constants.



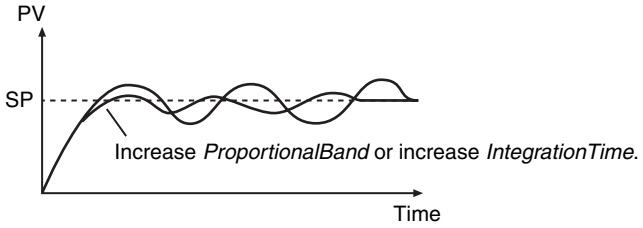
Additional Information

Adjusting PID Constants

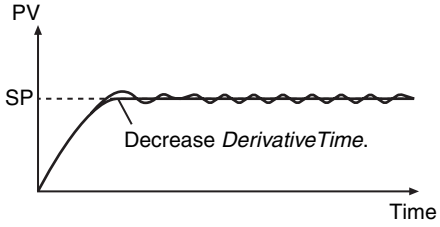
- If you need to eliminate hunting even if it takes time to stabilize the control system, increase the value of *ProportionalBand*. If a certain amount of hunting is not a problem, but it is necessary for the controlled system to stabilize quickly, decrease the value of *ProportionalBand*.



- If hunting continues too long, increase *ProportionalBand* or increase *IntegrationTime*.



- If rapid hunting occurs, decrease *DerivativeTime*.



Initial PID Constants for Temperature Control

If you use the PIDAT instruction for temperature control, use the following initial values of the PID constants as reference. Use the default values for the other variables.

Variables	Initial values (reference values)*
ProportinalBand	10%FS
IntegrationTime	233 s
DerivativeTime	40 s

* If you perform autotuning, use the results from autotuning.

Precautions for Correct Use

- The values of PV and SP must be between the values of *RngLowLmt* and *RngUpLmt*, inclusive. Align the units of these variables as shown below.

Unit	Values of *PV* and *SP*	Values of *RngLowLmt and RngUpLmt*
% FS	PV = (Process value in physical units − MIN)/(MAX − MIN) × 100	RngLowLmt = 0
	SP = (Set point in physical units − MIN)/(MAX − MIN) × 100*	RngUpLmt = 100
Physical unit	PV = Process value in physical units	RngLowLmt = MIN
	SV = Set point in physical units	RngUpLmt = MAX*

* MAX: Upper limit of input range in physical units, MIN: Lower limit of input range in physical units,

- The following table shows which variables can be changed depending on the operating status.

Variables	Control status		
	Instruction execution stopped*1	Automatic operation when autotuning is not being executed*2	Automatic operation when autotuning is being executed*3
Run	Possible	Possible	Possible
ManCtl	Possible	Possible	Possible
StartAT	Possible	Possible	Possible
PV	Possible	Possible	Possible
SP	Possible	Possible	Not possible
MVLowLmt	Possible	Possible	Not possible
MVUpLmt	Possible	Possible	Not possible
ManResetVal	Possible	Possible	Not possible
MVTrackSw	Possible	Possible	Not possible

Variables	Control status		
	Instruction execution stopped*1	Automatic operation when autotuning is not being executed*2	Automatic operation when autotuning is being executed*3
MVTrackVal	Possible	Possible	Not possible
StopMV	Possible	Possible	Possible
ErrorMV	Possible	Possible	Possible
Alpha	Possible	Possible	Not possible
ATCalcGain	Possible	Possible	Not possible
ATHystrs	Possible	Possible	Not possible
SampTime	Possible	Not possible	Not possible
RngLowLmt	Possible	Not possible	Not possible
RngUpLmt	Possible	Not possible	Not possible
DirOpr	Possible	Not possible	Not possible
ProportinalBand	Possible	Possible	Not possible
IntegrationTime	Possible	Possible	Not possible
DerivativeTime	Possible	Possible	Not possible
ManMV	Possible	Possible	Possible

*1 *ManCtl* is TRUE, *Run* is FALSE, *Error* is TRUE, or *MVTrackSw* is TRUE.

*2 *MacCtl* is FALSE, *Run* is TRUE, *Error* is FALSE, *MVTrackSw* is FALSE, and *ATBusy* is FALSE.

*3 *MacCtl* is FALSE, *Run* is TRUE, *Error* is FALSE, *MVTrackSw* is FALSE, and *ATBusy* is TRUE.

- *SampTime* is truncated below 100 nanoseconds.
- If the value of *StartAT* changes to TRUE while the value of *ManCtl* is TRUE, autotuning starts the next time the value of *ManCtl* changes to FALSE.
- If the value of *ErrorMV* is not within the valid range (−320 to 320), the value of *MV* will be 0 when an error occurs.
- Autotuning is canceled if the value of *ManCtl* changes to TRUE during autotuning.
- The value of *Error* does not change to TRUE even if an error occurs during autotuning.
- An error occurs in the following case. *Error* will change to TRUE, and an error code is assigned to *ErrorID*. *ATDone* and *ATBusy* change to FALSE. *MV* is set to the value of *ErrorMV* if the values of *ManCtl* and *Run* are FALSE. If the value of *ErrorMV* is outside of the valid range, the value of *MV* is 0.

Error	Value of *ErrorID*
The value of an input variable is outside of the valid range.	16#0400
RngLowLmt is greater than or equal to *RngUpLmt*.	16#0401
MVLowLmt is greater than or equal to *MVUpLmt*.	

- If an error stop is required for conditions other than the above, program the system so that the value of *Run* changes to FALSE when the error occurs.
- If an error occurs because the value of *PV* or *SP* exceeds the valid range, the error status is maintained for five seconds even if the value returns to within the valid range sooner. That is, the value of *Error* will remain FALSE for five seconds.
- PID control is restarted automatically if the value of *Run* is TRUE after the error is reset. Autotuning is restarted automatically if the values of *Run* and *StartAT* are TRUE.
- A check is made for errors each sampling period.

Sample Programming

In this sample, the PIDAT instruction is used to perform temperature control.

Specifications

Temperature control is performed according to the following specifications.

Item	Specification
Input type	K thermocouple
Input Unit	CJ1W-PH41U Analog Input Unit with Universal Inputs
Output Unit	CJ1W-OD212 Transistor Output Unit
Set point	90°C
Sampling period for PID control	100 ms
Output control period	1 s

Configuration and Settings

The following setting is used for the CJ1W-PH41U Analog Input Unit.

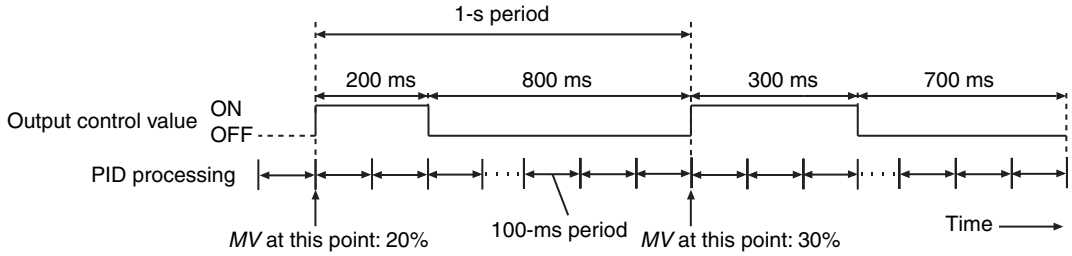
Setting	Set value
Input1:Input signal type	K(1)

The following I/O map settings are used.

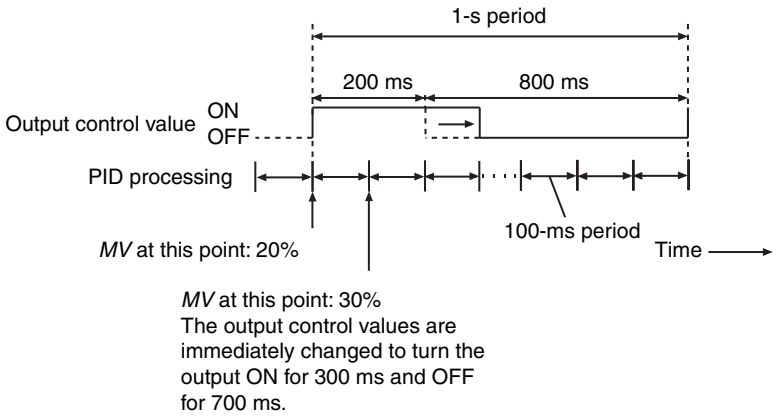
Unit	I/O port	Description	Variable
CJ1W-PH41U	Ch1_AIInPV	Measurement value for input 1 (INT data)	AI1
CJ1W-OD212	Ch1_Out00	Bit 00 of output word 1	DO1

Processing

- The manipulation value MV of the PIDAT instruction is obtained to control the output to the temperature controller. The output to the temperature controller is turned ON and OFF.
- The sampling period (*InitSetParams.SampTime*) of the PIDAT instruction is set to 100 ms. The task period must be sufficiently shorter than 100 ms. Therefore, the value of MV is refreshed every 100 ms.
- The output control period is 1 s. During that period, the ON time and OFF time of the output control value are controlled with a time-proportional output. For example, if the obtained value of *MV* is 20%, the output to the temperature control is ON for 200 ms and OFF for 800 ms. This is repeated at a 1-s period.



- If the most recent value of *MV* is smaller than the value of *MV* when the output control values were determined, the output control values do not change. If the most recent value of *MV* is larger than the value of *MV* when the output control values were determined, the most recent value is immediately reflected in the output control values. For example, assume that the output control values were determined when the value of *MV* was 20% (ON 200 ms, OFF 800 ms). If after 100 ms, the new value of *MV* is 30%, the output control values are immediately changed to turn the output ON for 300 ms and OFF for 700 ms.



- If autotuning is performed and the value of MV is 100%, the output is immediately turned ON regardless of the control period.

Application Programming

LD

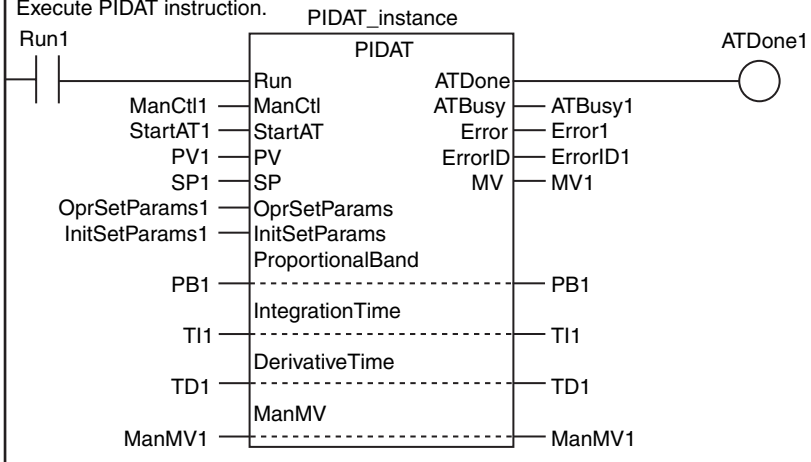
Variable	Data type	Initial value	Retain	Comment
Run1	BOOL	False	☐	Execution condition
ManCtl1	BOOL	False	☐	Manual/auto control
StartAT1	BOOL	False	☐	Autotuning execution condition
PV1	REAL	0.0	☐	Process value
SP1	REAL	90	☐	Set point
OprSetParams1	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=False, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)	☐	Operation setting parameters
InitSetParams1	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=False)	☐	Initial setting parameters
PB1	REAL	10	☑	Proportional band
TI1	TIME	T#0S	☑	Integration time
TD1	TIME	T#0S	☑	Derivative time
ManMV1	REAL	0.0	☐	Manual manipulated variable
ATDone1	BOOL	False	☐	Autotuning normal completion
ATBusy1	BOOL	False	☐	Executing autotuning
Error1	BOOL	False	☐	Error
ErrorID1	WORD	16#0	☐	Error ID
MV1	REAL	0.0	☐	Manipulated variable
PulseOnTime	TIME	T#0s	☐	Control output ON time
PulseCycTime	TIME	T#1s	☐	Control period
ResetPulse	BOOL	False	☐	Timer reset
PIDAT_instance	PIDAT		☐	
TOF_instance	TOF		☐	
TON_instance	TON		☐	

Obtain the process value.

Inline ST

```
1  PV1:=INT_TO_REAL(AI1)/REAL#10.0;   // Convert PV AI1 to real number.
                                      // CJ1W-PH41U output is ten times the process value, so divide by 10.0.
```

Execute PIDAT instruction.



Time-proportional output

Inline ST

```
 1  PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);          // Calculate ON time output control value.
 2  TOF_instance(In:=BOOL#FALSE, PT:=PulseOnTime, Q=>DO1);       // Switch between ON and OFF with TOF instruction.
 3  TON_instance(In:=BOOL#TRUE, PT:=PulseCycTime, Q=>ResetPulse); // Measure timer reset time with TON instruction.
 4  IF (ResetPulse=BOOL#TRUE) THEN                               // Reset timer.
 5      TOF_instance(In:=BOOL#TRUE);
 6      TON_instance(In:=BOOL#FALSE);
 7  END_IF;
 8  IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN           // If MV1 = 100% for autotuning...
 9      DO1:=BOOL#TRUE;                                          // Turn ON the output immediately.
10  END_IF;
```

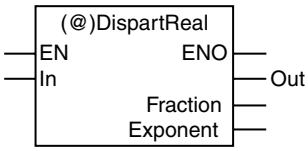
ST

Variable	Data type	Initial value	Retain	Comment
Run1	BOOL	False	☐	Execution condition
ManCtl1	BOOL	False	☐	Manual/auto control
StartAT1	BOOL	False	☐	Autotuning execution condition
PV1	REAL	0.0	☐	Process value
SP1	REAL	90	☐	Set point
OprSetParams1	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=False, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)	☐	Operation setting parameters
InitSetParams1	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=False)	☐	Initial setting parameters
PB1	REAL	10	☑	Proportional band
TI1	TIME	T#0S	☑	Integration time
TD1	TIME	T#0S	☑	Derivative time
ManMV1	REAL	0.0	☐	Manual manipulated variable
ATDone1	BOOL	False	☐	Autotuning normal completion
ATBusy1	BOOL	False	☐	Executing autotuning
Error1	BOOL	False	☐	Error
ErrorID1	WORD	16#0	☐	Error ID
MV1	REAL	0.0	☐	Manipulated variable
PulseOnTime	TIME	T#0s	☐	Control output ON time
PulseCycTime	TIME	T#1s	☐	Control period
ResetPulse	BOOL	False	☐	Timer reset
PIDAT_instance	PIDAT		☐	
TOF_instance	TOF		☐	
TON_instance	TON		☐	

```
// Convert PV AI1 to real number.
PV1:=INT_TO_REAL(AI1)/REAL#10.0;    // CJ1W-PH41U output is ten times the process value, so divide by 10.0.

// Execute PIDAT instruction.
PIDAT_instance(
    Run                 :=Run1,
    ManCtl              :=Manctl1,
    StartAT             :=StartAT1,
    PV                  :=PV1,
    SP                  :=SP1,
    OprSetParams        :=OprSetParams1,
    InitSetParams       :=InitSetParams1,
    ProportionalBand:=PB1,
    IntegrationTime     :=TI1,
    DerivativeTime      :=TD1,
    ManMV               :=ManMV1,
    ATDone              =>ATDone1,
    ATBusy              =>ATBusy1,
    Error               =>Error1,
    ErrorID             =>ErrorID1,
    MV                  =>MV1);

// Time-proportional output
PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);                  // Calculate ON time output control value.
TOF_instance(In:=BOOL#FALSE, PT:=PulseOnTime, Q=>DO1);               // Switch between ON and OFF with TOF instruction.
TON_instance(In:=BOOL#TRUE, PT:=PulseCycTime, Q=>ResetPulse);        // Measure timer reset time with TON instruction.
IF (ResetPulse=BOOL#TRUE) THEN                                       // Reset timer.
    TOF_instance(In:=BOOL#TRUE);
    TON_instance(In:=BOOL#FALSE);
END_IF;
IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN                   // If MV1 = 100% for autotuning...
    DO1:=BOOL#TRUE;                                                  // Turn ON the output immediately.
END_IF;
```

DispartReal

The DispartReal instruction separates a real number into the signed mantissa and the exponent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DispartReal	Separate Mantissa and Exponent	FUN		Out:=DispartReal(In, Fraction, Exponent);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Real number	Input	Real number to separate	Depends on data type.	---	*1
Out	Return value	Output	Always TRUE	TRUE only	---	---
Fraction	Signed mantissa		Signed mantissa	*2		
Exponent	Exponent		Exponent	*3		

*1 If you omit the input parameter, the default value is not applied. A building error will occur.

*2 The valid ranges depend on the data types of *In* and *Fraction*. Refer to *Function* for details.

*3 If *In* is REAL data, –44 to 32. If *In* is LREAL data, –322 to 294

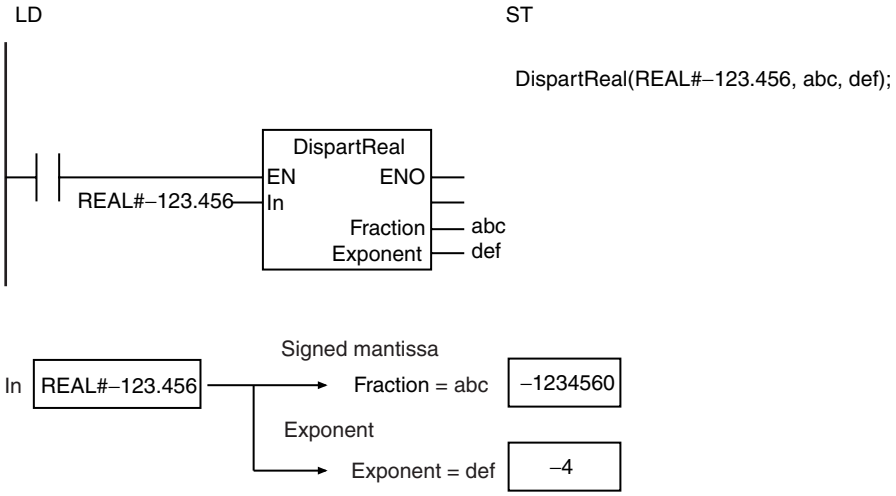
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	OK																			
Fraction	Must be DINT if the data type of *In* is REAL and LINT if the data type of *In* is LREAL.																			
Exponent											OK									

Function

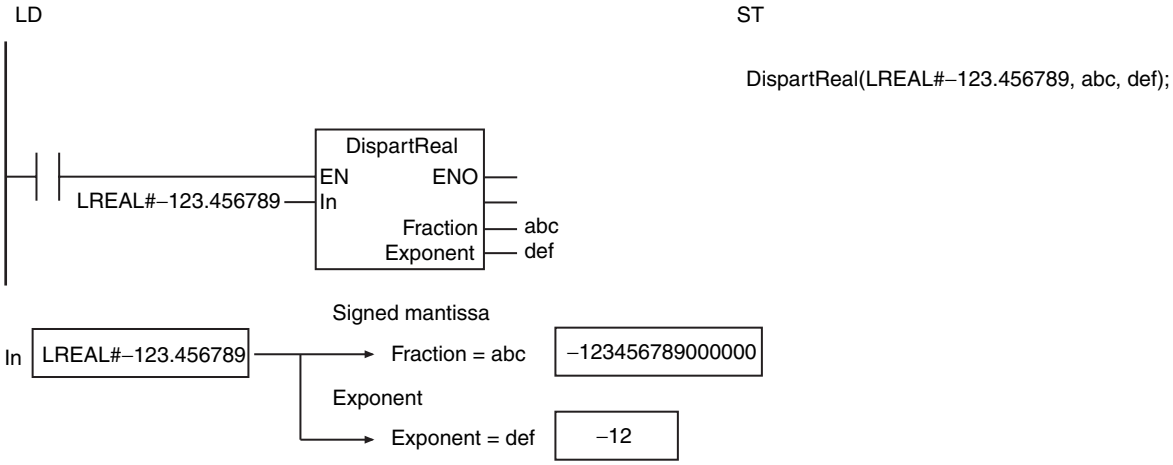
The DispartReal instruction separates real number *In* into signed mantissa *Fraction* and exponent *Exponent*.

If *In* is REAL data, *Fraction* is a 7-digit integer. If *In* is LREAL data, *Fraction* is a 15-digit integer.

The following example is for when *In* is REAL data with a value of REAL#−123.456.



The following example is for when *In* is LREAL data with a value of LREAL#−123.456789.



The following table shows the valid ranges for *Fraction* according to the data types *In* and *Fraction*.

Data type of *In*	Data type of *Fraction*	Valid range of *Fraction*
REAL	DINT	−9999999 to 9999999
LREAL	LINT	−999999999999999 to 999999999999999

Additional Information

Use the UniteReal instruction (page 2-421) to combine a signed mantissa and exponent to form a real number.

Precautions for Correct Use

- Depending on the value of *In*, error may occur in the conversion to an integer.

- If the number of valid digits in *In* exceeds the number of valid digits of *Fraction*, the value is rounded to fit in the valid range of *Fraction*. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 –1.49 → –1
0.5	If the ones digit is an even number, the value is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 –1.50 → –2 –2.50 → –2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 –1.51 → –2

- An error occurs in the following case. *ENO* will be FALSE, and *Fraction* and *Exponent* will not change.
  - The value of *In* is nonnumeric or infinity.

UniteReal

The UniteReal instruction combines a signed mantissa and exponent to make a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
UniteReal	Combine Real Number Mantissa and Exponent	FUN	(@)UniteReal EN ENO Fraction Out Exponent	Out:=UniteReal(Fraction, Exponent);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Fraction	Signed mantissa	Input	Signed mantissa	Depends on data type.	---	*
Exponent	Exponent		Exponent			0
Out	Real number	Output	Real number	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

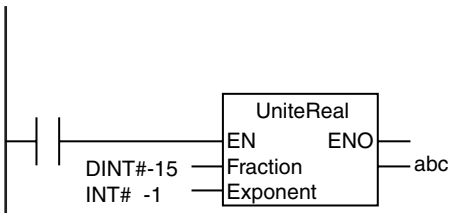
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Fraction												OK	OK							
Exponent											OK									
Out	Must be REAL if the data type of *Fraction* is DINT and LREAL if the data type of *Fraction* is LINT.																			

Function

The UniteReal instruction combines signed mantissa *Fraction* and exponent *Exponent* to make real number *Out*.

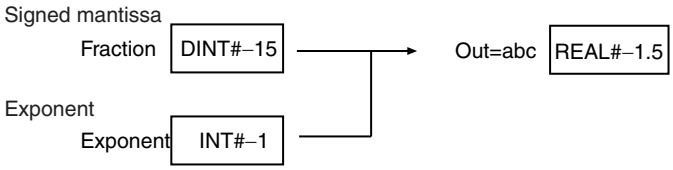
The following example is for when *Fraction* is DINT#−15 and *Exponent* is INT#−1.

LD



ST

```
abc:=UniteReal(DINT#15, INT#-1);
```



Additional Information

Use the DispartReal instruction (page 2-418) to separate a real number into the signed mantissa and exponent.

Precautions for Correct Use

- Depending on the values of *Fraction* and *Exponent*, error may occur in the conversion from an integer to a real number.
- If the combined result exceeds the valid range of *Out* and *Exponent* is positive, the value of *Out* will be infinity with the same sign as *Fraction*. If *Exponent* is negative, the value of *Out* will be 0.

NumToDecString and NumToHexString

NumToDecString: Converts an integer to a fixed-length decimal text string.
NumToHexString: Converts an integer to a fixed-length hexadecimal text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NumToDecString	Fixed-length Decimal Text String Conversion	FUN	(@)NumToDecString EN In L Fill ENO Out	Out:=NumToDecString(In, L, Fill);
NumToHexString	Fixed-length Hexadecimal Text String Conversion	FUN	(@)NumToHexString EN In L Fill ENO Out	Out:=NumToHexString(In, L, Fill);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Integer	Input	Integer	Depends on data type.	---	*
L	Number of characters		Number of characters in *Out*	0 to 1985		1
Fill	Fill character		Fill character	_BLANK or _ZERO		_BLANK
Out	Text string	Output	Text string	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
L							OK													
Fill	Refer to *Function* for the enumerators for the enumerated type _eFILL_CHR.																			
Out																				OK

Function

● NumToDecString

The NumToDecString instruction converts integer *In* to a decimal text string of UTF-8 alphanumeric characters. If *In* contains a negative value, a minus sign (–) is added to the front of the text string.

NumToHexString

The NumToHexString instruction converts integer *In* to a hexadecimal text string of UTF-8 alphanumeric characters. If *In* is negative, it is expressed in its two's complement (bits inverted and then 1 added).

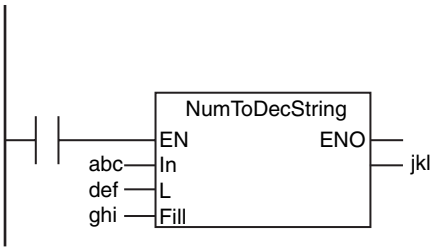
For either instruction, the number of characters in text string *Out* is adjusted to number of characters *L*. If there are not enough characters, the upper digits are filled with fill character *Fill*. If the number of characters in the conversion result exceeds *L*, *L* characters from the lower digits of the conversion result are assigned to *Out*. The NULL character is not included in the number of characters.

The data type of *Fill* is enumerated type _eFILL_CHR. The meaning of the enumerators are as follows:

Enumerator	Meaning
_BLANK	'' (blank character)
_ZERO	'0'

The following examples are for the NumToDecString instruction.

LD



ST

```
jkl:=NumToDecString(abc, def, ghi);
```

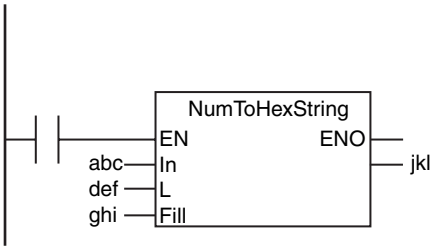
In = abc = INT#128, L = def = UINT#8, Fill = ghi = _BLANK
Out = jkl [ ][ ][ ][ ][ ]1[2][8]

In = abc = INT#–128, L = def = UINT#8, Fill = ghi = _BLANK
Out = jkl [ ][ ][ ][ ]–[1][2][8]

In = abc = INT#–128, L = def = UINT#8, Fill = ghi = _ZERO
Out = jkl [–][0][0][0][0][1][2][8]

The following examples are for the NumToHexString instruction.

LD



ST

```
jkl:=NumToHexString(abc, def, ghi);
```

In = abc = INT#128, L = def = UINT#8, Fill = ghi = _BLANK
Out = jkl [ ][ ][ ][ ][ ][8][0]

In = abc = INT#128, L = def = UINT#8, Fill = ghi = _ZERO
Out = jkl [0][0][0][0][0][0][8][0]

In = abc = INT#–128, L = def = UINT#8, Fill = ghi = _BLANK
Out = jkl [F][F][F][F][F][F][8][0]

Precautions for Correct Use

- The value of *Out* does not change if the value of *L* is 0.
- If the number of characters in the conversion result exceeds the value of *L*, *L* characters from the lower characters of the conversion result are stored in *Out*. The following is an example.

Instruction	Value of *In*	Value of *L*	Value of *Out*
NumToDecString	128	2	28
NumToHexString			80

- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *L* is outside of the valid range.
  - The value of *Fill* is outside of the valid range.
  - The conversion result exceeds the range of *Out*.

HexStringToNum_**

The HexStringToNum_** instruction converts a hexadecimal text string to an integer.

Instruction	Name	FB/FUN	Graphic expression	ST expression
HexStringToNum_**	Hexadecimal Text String-to-Number Conversion Group	FUN	(@)HexStringToNum_** EN ENO In Out "**" must be an integer data type.	Out:=HexStringToNum_**(In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Hexadecimal text string	Input	Hexadecimal text string	Depends on data type.	---	''
Out	Integer	Output	Integer	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out						OK	OK	OK	OK	OK	OK	OK	OK							

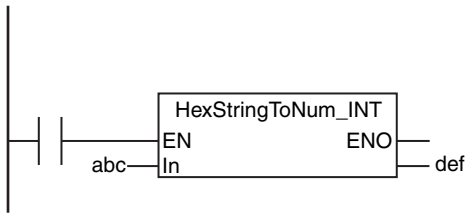
Function

The HexStringToNum_** instruction converts hexadecimal text string *In* to an integer. Any spaces (16#20) or '0' (16#30) in the upper digits are ignored. Underbars (16#5F) in the text string are ignored.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the INT data type, the instruction is HexStringToNum_INT.

A few examples are given below.

LD



ST

```
def:=HexStringToNum_INT(abc);
```

In = abc 8 0 → Out = def = 128

In = abc – 8 0 → Out = def = –128

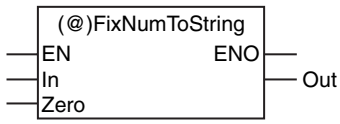
In = abc – 0 0 0 0 0 F → Out = def = –15

Precautions for Correct Use

- Even if the conversion result exceeds the valid range of *Out*, an error will not occur. The value of *Out* will be an illegal value.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* does not end in a NULL character.
  - The content of *In* includes characters that cannot be converted to numbers.

FixNumToString

The FixNumToString instruction converts a signed fixed-decimal number to a decimal text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FixNumToString	Fixed-decimal Number-to-Text String Conversion	FUN		Out:=FixNumToString(In, Zero);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Fixed-decimal number	Input	Signed fixed-decimal number	Depends on data type.	---	0
Zero	Zero augmentation		Augmentation of zeros if there are less than 3 decimal digits TRUE: Add '0' FALSE: Do not add '0'			TRUE
Out	Decimal text string	Output	Hexadecimal text string	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In				OK																
Zero	OK																			
Out																				OK

Function

The FixNumToString instruction converts signed fixed-decimal number *In* to a decimal text string. The following conversion is used.

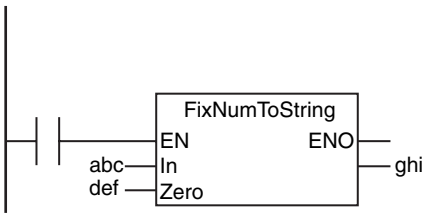
1 The hexadecimal number *In* is converted to a decimal number.

2 The result is divided by 1,000.

Zero augmentation *Zero* specifies whether to add '0' to the third decimal place of *Out* when there are less than three decimal digits in *In*. If the value of *Zero* is TRUE, '0' is added. A NULL character is placed at the end of *Out.*

A few examples are given below.

LD



ST

```
ghi:=FixNumToString(abc, def);
```

***In* = abc**	***Out* = ghi**	
	***Zero* = def = TRUE**	***Zero* = def = FALSE**
16#0001462C (10#83500)	'83.500'	'83.5'
16#00051AA4 (10#334500)	'334.500'	'334.5'
16#0003BEFC (10#245500)	'245.500'	'245.5'

Additional Information

The format for fixed-point decimal numbers is the same as the fixed-decimal output format of the OMRON FZ-series Vision Sensors.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The conversion result exceeds the valid range of *Out*.

StringToFixNum

The StringToFixNum instruction converts a decimal text string to a signed fixed-decimal number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringToFixNum	Text String-to-Fixed-decimal Conversion	FUN	(@)StringToFixNum EN ENO In Out	Out:=StringToFixNum(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Decimal text string	Input	Decimal text string	Depends on data type.	---	''
Out	Fixed-decimal number	Output	Fixed-decimal number	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out				OK																

Function

The StringToFixNum instruction converts decimal text string *In* to a fixed-decimal number. The following conversion is used.

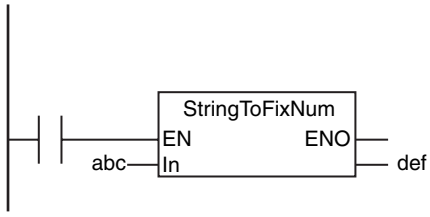
1 The number in *In* is multiplied by 1,000.

2 The fractional part is truncated.

3 The result is given as a 32-bit hexadecimal number (DWORD).

A few examples are given below.

LD

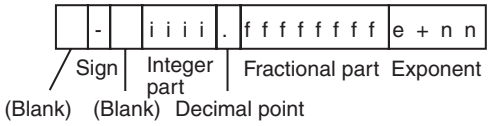


ST

```
def:=StringToFixNum(abc);
```

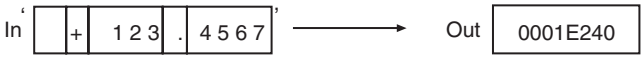
In = abc	Out = def
‘83.5’	16#0001462C (10#83500)
‘334.5’	16#00051AA4 (10#334500)
‘245.5’	16#0003BEFC (10#245500)

The format of the text sting in *In* is given below.

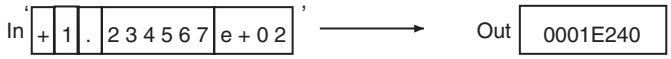


Name	Format
Sign	• Any consecutive blank characters (16#20) at the beginning of the text string are ignored. Any single plus or minus sign that follows is treated as the sign. • The sign can be omitted. • Any consecutive blank characters after the sign are ignored.
Integer part	• Consecutive numbers (‘0’ to ‘9’) after the sign and up to the decimal point are taken as the integer part. The sign may sometimes be omitted. There may be blank characters between the sign and the integer part. • If the decimal point and fractional part are omitted, the characters up to the exponent are taken as the integer part. • If the decimal point, fractional part, and exponent are omitted, the characters up to the end of the text string are taken as the integer part. • The integer part cannot be omitted. • The maximum number of digits in the integer part is the maximum text string length of 1986 minus the total number of bytes in the following: the sign, decimal point, fractional part, exponent, and blank characters before and after the sign.
Decimal point	• A single dot (‘.’) following the integer part is taken as the decimal point. • Omit the decimal point if there is no fractional part.
Fractional part	• Consecutive numbers (‘0’ to ‘9’) after the decimal point and up to the exponent are taken as the fractional part. • If the exponent is omitted, the characters up to the end of the text string are taken as the fractional part. • The fractional part can be omitted. If there is no decimal point, then there is no fractional part. • The fractional part can consist of a maximum of 15 digits.
Exponent	• The exponent consists of a single ‘e’ or ‘E’ after the fractional part, a following single plus or minus sign, and the remaining continuous numbers (‘0’ to ‘9’) to the end of the text string. • If there is no fractional part, then the above text string after the decimal point is taken as the exponent. • If there is no decimal point or fractional part, then the above text string after the integer part is taken as the exponent. • The exponent can be omitted. • The numeric part of the exponent can consist of a maximum of three digits.

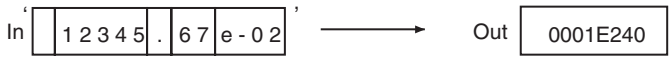
Example 1: The following example uses the sign, decimal point, and fractional part, but does not use an exponent.



Example 2: The following example uses the sign, decimal point, fractional part, and exponent.



Example 3: The following example does not use the sign, but uses the decimal point, fractional part, and exponent.



Example 4: The following example does not use the sign, fractional part, decimal point, and exponent.

Additional Information

The format for fixed-point decimal numbers is the same as the fixed-decimal output format of the OMRON FZ-series Vision Sensors.

Precautions for Correct Use

- The digits after the third decimal digit are truncated in *In*.
- Underbars (16#5F) in the text string in *In* are ignored.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* does not end in a NULL character.
  - The content of *In* includes characters that cannot be converted to numbers.
  - The content of *In* has a decimal point but not a fractional part.
  - The conversion result exceeds the valid range of *Out*.

DtToString

The DtToString instruction converts a date and time to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DtToString	Date and Time-to-Text String Conversion	FUN	(@)DtToString EN ENO In Out	Out:=DtToString(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Text string	Output	Text string	30 bytes (29 single-byte alphanumeric characters plus the final NULL character)	---	---

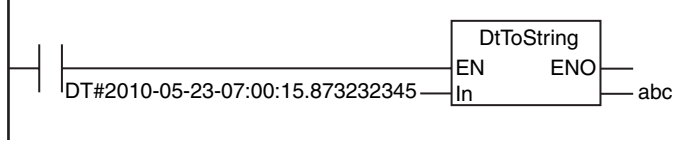
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out																				OK

Function

The DtToString instruction converts date and time *In* to a text string. A NULL character is placed at the end of text string *Out.*

An example when *In* is 2010-5-23-07:00:15.873232345 (7:00 am and 15.873232345 seconds on May 23, 2010) is given below. The value of variable *abc* will be '2010-05-23-07:00:15.873232345'.

LD



ST

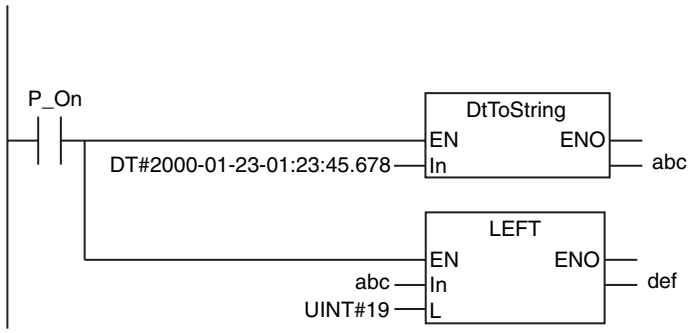
```
abc:=DtToString(DT#2010-05-23-07:00:15.873232345);
```

Additional Information

Out is in nanoseconds. To get a text string in seconds or milliseconds, combine this instruction with the LEFT or RIGHT instruction (page 2-522).

An example to get a text string in seconds is given below.

● LD



● ST

```
def:=LEFT(DtToString(DT#2000-01-23-01:23:45.678), UINT#19);
```

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The conversion result exceeds the valid range of *Out*.

DateToString

The DateToString instruction converts a date to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DateToString	Date-to-Text String Conversion	FUN	(@)DateToString EN ENO In Out	Out:=DateToString(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date	Input	Date	Depends on data type.	Year, month, day	D#1970-1-1
Out	Text string	Output	Text string	11 bytes (10 single-byte alphanumeric characters plus the final NULL character)	---	---

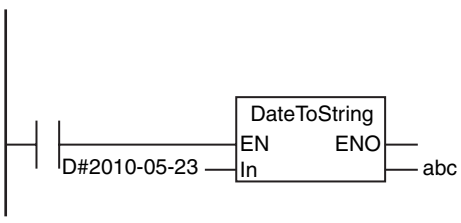
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK			
Out																				OK

Function

The DateToString instruction converts date *In* to a text string. A NULL character is placed at the end of *Out*.

An example when *In* is 2010-5-23 (May 23, 2010) is given below. The value of variable *abc* will be '2010-05-23'.

LD



ST

```
abc:=DateToString(D#2010-05-23);
```

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The conversion result exceeds the valid range of *Out*.

TodToString

The TodToString instruction converts a time of day to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TodToString	Time of Day-to-Text String Conversion	FUN	(@)TodToString EN ENO In Out	Out:=TodToString(In);

Variables

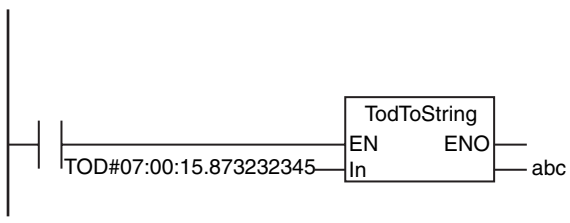
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time of day	Input	Time of day	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
Out	Text string	Output	Text string	19 bytes (18 single-byte alpha-numeric characters plus the final NULL character)	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		OK		
Out																				OK

Function

The TodToString instruction converts time of day *In* to a text string. A NULL character is placed at the end of *Out.* An example when *In* is 07:00:15.873232345 (7:00 am and 15.873232345 seconds) is given below. The value of variable *abc* will be '07:00:15.873232345'.

LD



ST

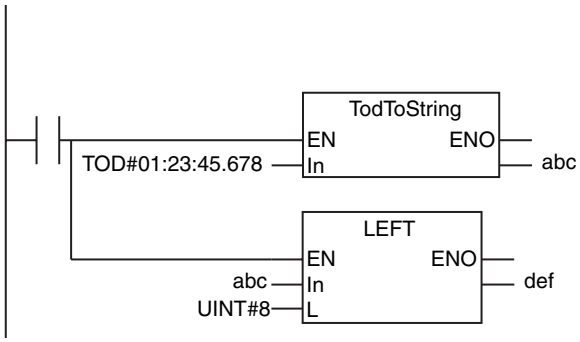
```
abc:=TodToString(TOD#07:00:15.873232345);
```

Additional Information

Out is in nanoseconds. To get a text string in seconds or milliseconds, combine this instruction with the LEFT or RIGHT instruction (page 2-522).

An example to get a text string in seconds is given below.

● LD



● ST

```
def:=LEFT(TodToString(TOD#01:23:45.678), UINT#8);
```

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The conversion result exceeds the valid range of *Out*.

GrayToBin_** and BinToGray_**

GrayToBin_**: Converts a gray code to a bit string.
BinToGray_**: Converts a bit string to a gray code.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GrayToBin_**	Gray Code-to-Binary Code Conversion Group	FUN	(@)GrayToBin_** EN ENO In Out "**" must be a bit string data type.	Out:=GrayToBin_**(In); "**" must be a bit string data type.
BinToGray_**	Binary Code-to-Gray Code Conversion	FUN	(@)BinToGray_** EN ENO In Out "**" must be a bit string data type.	Out:=BinToGray_**(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out	Must be same data type as *In*																			

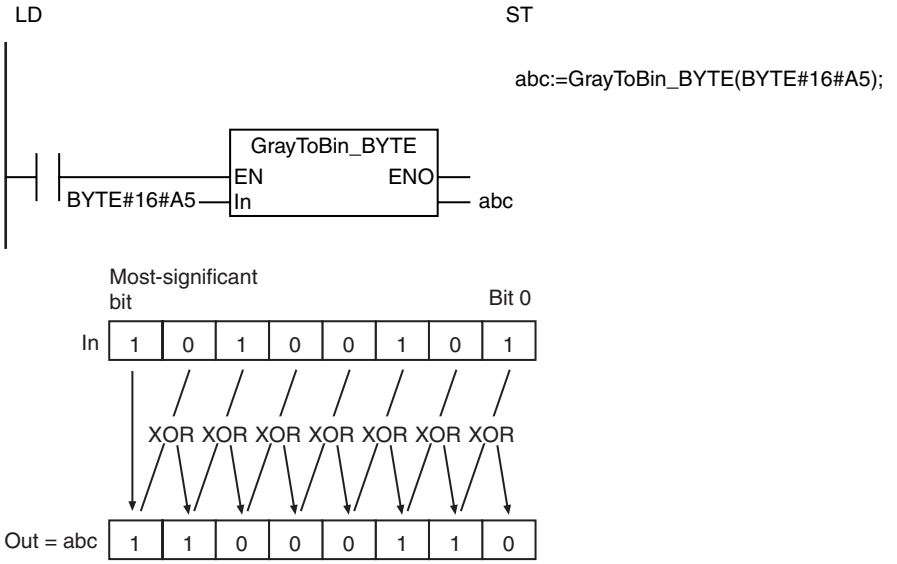
Function

- **GrayToBin_****

The GrayToBin_** instructions convert the gray code in date to convert *In* to a bit string. The conversion procedure is as follows for when *In* and *Out* are BYTE data.

1. The most-significant bit (bit 7) of *In* is used as is as the most-significant bit (bit 7) of *Out.*
2. An exclusive logical OR is taken of the value of bit 6 in *In* and the value of bit 7 in *Out*. The result is used as bit 6 of *Out*.
3. This process is repeated through the least-significant bit (bit 0) of *Out*.

The following example for the GrayToBin_BYTE instruction is for when *In* is BYTE#16#A5.



● BinToGray_**

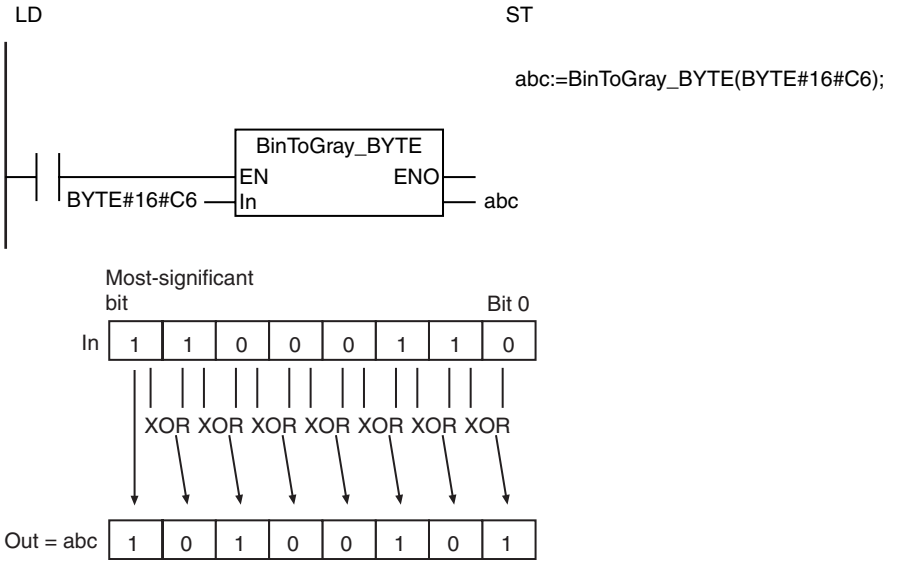
The BinToGray_** instructions convert the bit string in data to convert *In* to a gray code. The conversion procedure is as follows for when *In* and *Out* are BYTE data.

1 The most-significant bit (bit 7) of *In* is used as is as the most-significant bit (bit 7) of *Out.*

2 An exclusive logical OR is taken of the value of bit 7 in *In* and the value of bit 6 in *In*. The result is used as bit 6 of *Out*.

3 This process is repeated through the least-significant bit (bit 0) of *Out*.

The following example for the BinToGray_BYTE instruction is for when *In* is BYTE#16#C6.



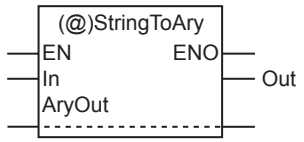
The name of the instruction is determined by the data types of *In* and *Out*. For example, if *In* and *Out* are the WORD data type, the instruction is GrayToBin_WORD or BinToGray_WORD.

Precautions for Correct Use

The data types of *In* and *Out* must be the same.

StringToAry

The StringToAry instruction converts a text string to a BYTE array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringToAry	Text String-to-Array Conversion	FUN		Out:=StringToAry(In, AryOut);

Variables

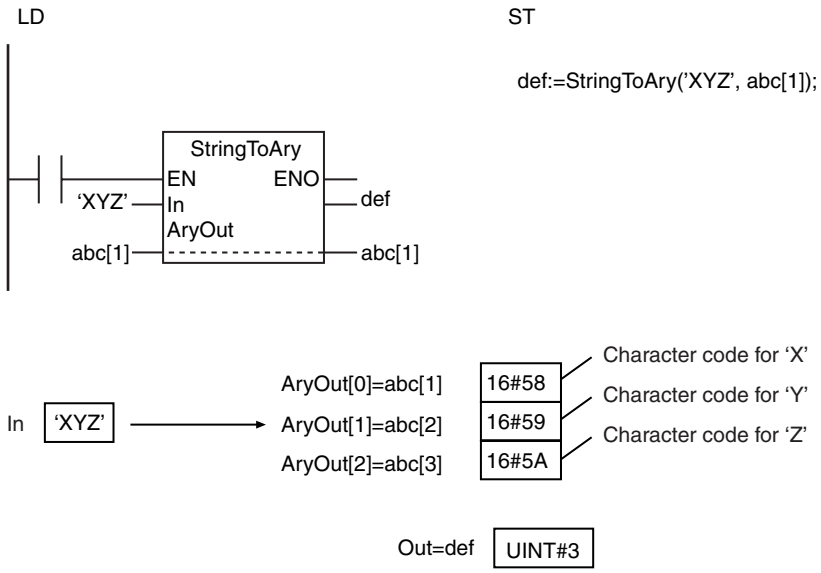
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string	Input	Text string	Depends on data type.	---	''
AryOut[] (array)	BYTE array	In-out	BYTE array	Depends on data type.	---	---
Out	Number of bytes to convert	Output	Number of bytes to convert	0 to 1985	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
AryOut[] (array)		OK																		
Out							OK													

Function

The StringToAry instruction takes the character codes in text string *In* as numbers and stores them individually in a BYTE array, *AryOut[]*. The number of bytes that was converted is stored in *Out*.

The following example is for when *In* is ‘XYZ’.



Precautions for Correct Use

- The NULL character at the end of *In* is not stored in *AryOut[]*.
- If the *In* text string contains only the NULL character, the value of *Out* will be 0 and *AryOut[]* will not change.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* and *AryOut[]* will not change.
  - *In* does not end in a NULL character.
  - The number of bytes in *In* is larger than the number of elements in *AryOut[]*.

AryToString

The AryToString instruction converts a BYTE array to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryToString	Array-to-Text String Conversion	FUN	(@)AryToString EN ENO In Out Size	Out:=AryToString(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	BYTE array	Input	BYTE array Maximum number of elements: 1985	Depends on data type.	---	*
Size	Number of elements to convert		Number of elements of *In[]* for conversion	0 to 1985		1
Out	Text string	Output	Text string	Depends on data type.	---	---

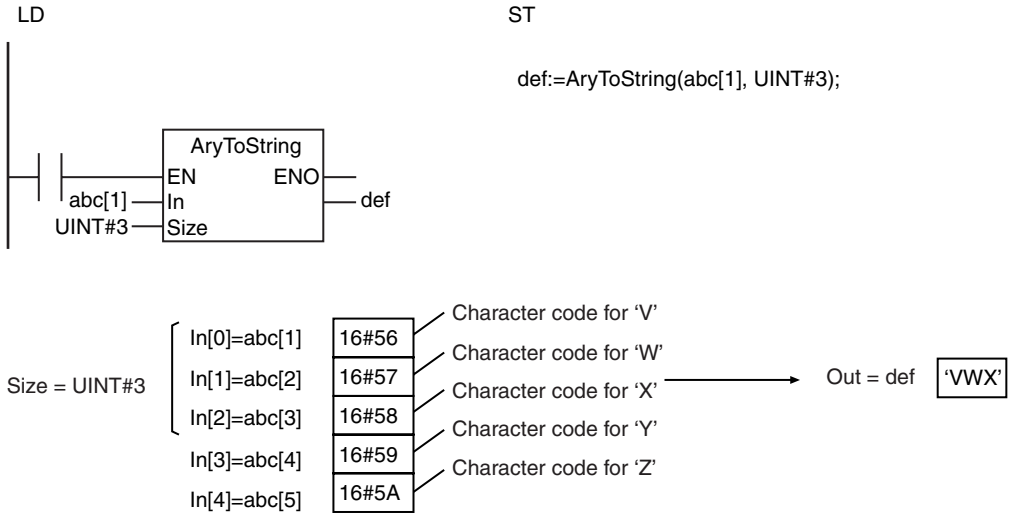
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Size							OK													
Out																				OK

Function

The AryToString instruction takes the elements of a BYTE array, *In[]*, from *In[0]* as character codes and stores them in text string *Out*. A NULL character is placed at the end of *Out*. *Size* specifies the number of elements of *In[]* to convert. If there is a NULL character between *In[0]* and *In[Size–1]*, no character codes past it are stored in *Out*.

The following example is for when *Size* is UINT#3.



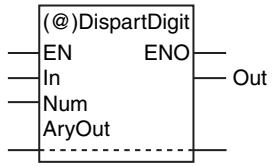
Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *Size* exceeds the array area of *In[]*.
- The conversion result exceeds the valid range of *Out*.

DispartDigit

The DispartDigit instruction separates a bit string into 4-bit units.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DispartDigit	Four-bit Separation	FUN		DispartDigit(In, Num, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to separate	Input	Bit string to separate	Depends on data type.	---	*
Num	Number of digits to separate		Number of digits to separate	0 to No. of bits in *In*		1
AryOut[] (array)	Separation results array	In-out	Separation results array	16#00 to 16#0F	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

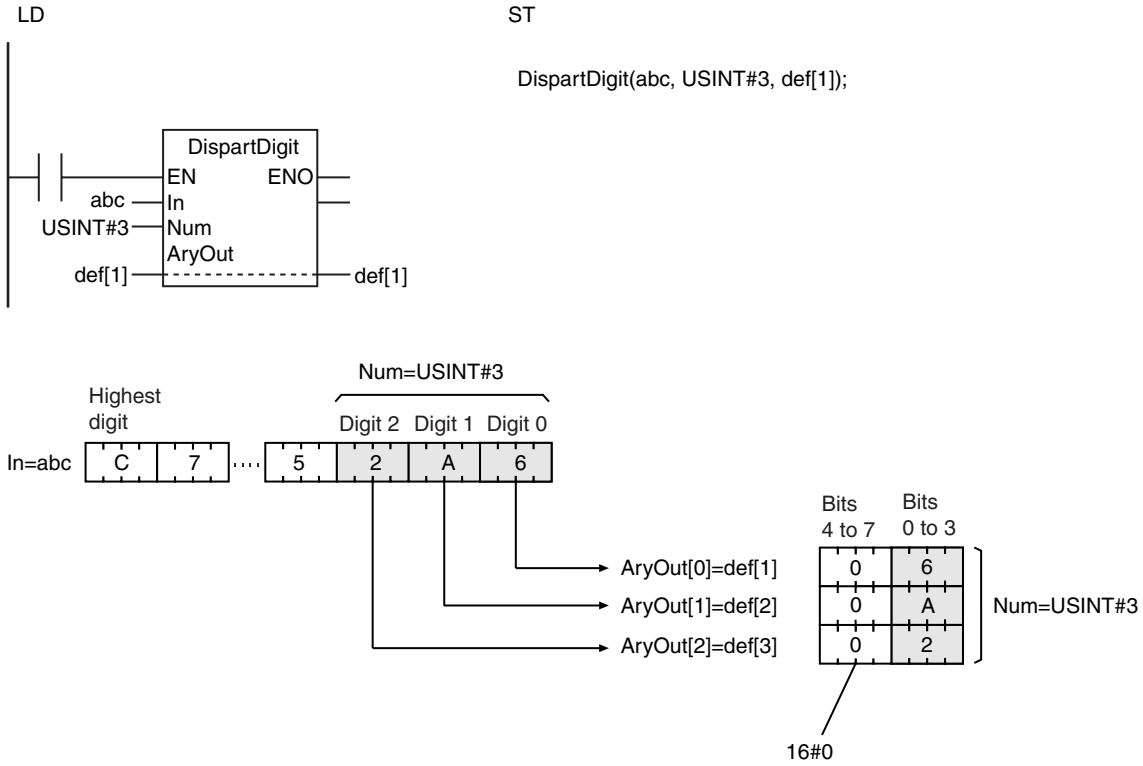
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK														
AryOut[] (array)		OK																		
Out	OK																			

Function

The DispartDigit instruction separates data to separate *In* into 4-bit units (digits) and stores them in separation results array *AryOut[]*.

First, *In* is separated into 4-bit units. Then, the lowest 4 bits are stored in *AryOut[0]*. *AryOut[0]* is BYTE data, so 16#0 is stored in bits 4 to 7. This process is repeated for the number of digits that is specified in number of digits to separate *Num*.

The following example is for when *Num* is USINT#3.



Additional Information

Use the UniteDigit_** instruction (page 2-447) to join 4-bit units from array elements.

Precautions for Correct Use

- The values in *AryOut[]* do not change if the value of *Num* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the array area of *AryOut[]*.
  - *AryOut[]* is not a BYTE array.
  - An array without a subscript is passed to *AryOut[]*.

UniteDigit_**

The UniteDigit_** instructions join 4-bit units of data into a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
UniteDigit_**	Four-bit Join Group	FUN	(@)UniteDigit_** EN ENO In Out Num "**" must be a bit string data type.	Out:=UniteDigit_**(In, Num); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to join	Input	Array to join	Depends on data type.	---	*
Num	Number of digits to join		Number of digits to join	0 to No. of bits in *Out*		1
Out	Joined result	Output	Bit string with joined result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Num						OK														
Out		OK	OK	OK	OK															

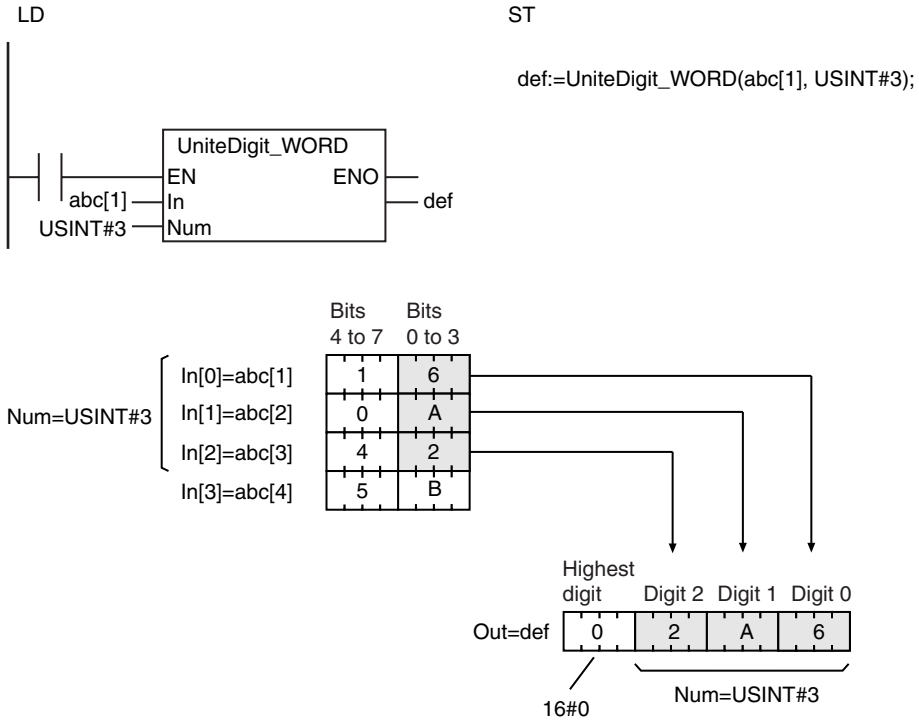
Function

The UniteDigit_** instructions join 4-bit units from the elements of array to join *In[]*. It creates a bit string in joined result *Out*. (Four bits is one digit.)

Number of digits to join *Num* specifies the number of array elements to join. First, the lower four bits from each element from *In[0]* to *In[Num–1]* are joined to create a bit string with *Num* digits. To this, 16#0 is added to the upper digits for the number of digits of *Out* minus the value of *Num*. The result is stored in *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the WORD data type, the instruction is UniteDigit_WORD.

The following example shows the UniteDigit_WORD instruction when *Num* is USINT#3.



Additional Information

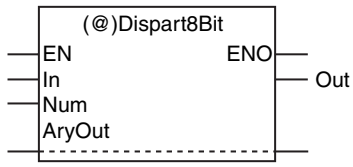
Use the DispartDigit instruction (page 2-445) to separate a bit string into 4-bit units.

Precautions for Correct Use

- If the value of Num is 0, the value of *Out* is 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the array area of *In[]*.
  - *In[]* is not a BYTE array.
  - An array without a subscript is passed to *In[]*.

Dispart8Bit

The Dispart8Bit instruction separates a bit string into individual bytes.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Dispart8Bit	Byte Data Separation	FUN		Dispart8Bit(In, Num, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to separate	Input	Bit string to separate	Depends on data type.	---	*
Num	Number of bytes to separate		Number of bytes to separate	0 to No. of bytes in *In*		1
AryOut[] (array)	Separation results array	In-out	Separation results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

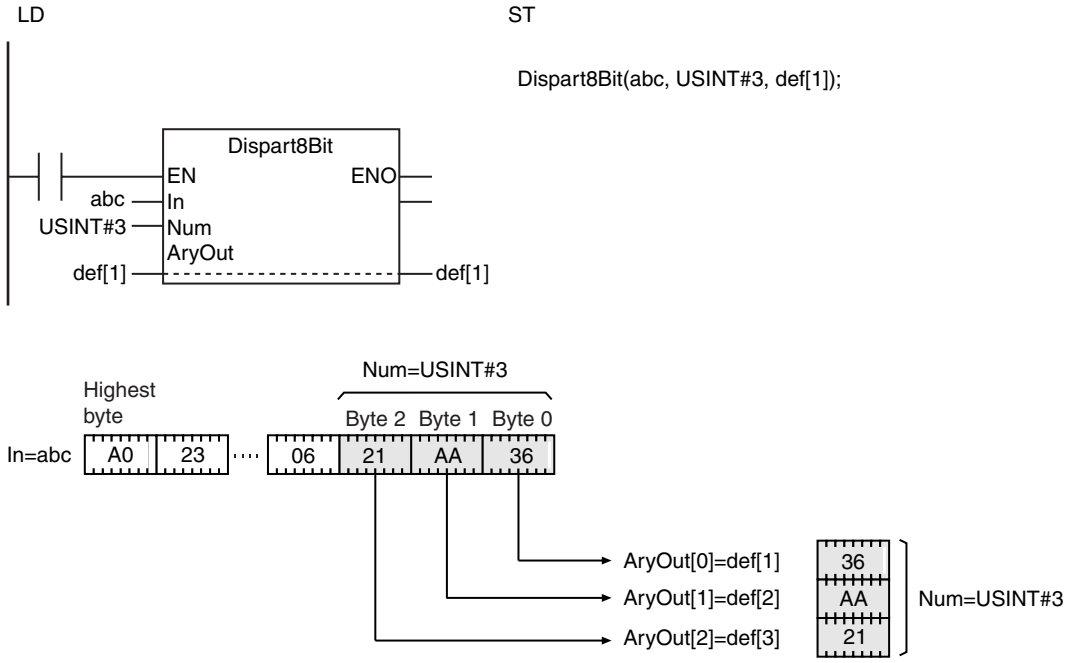
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK														
AryOut[] (array)		OK																		
Out	OK																			

Function

The Dispart8Bit instruction separates data to separate *In* into individual bytes and stores them in separation results array *AryOut[]*.

First, *In* is separated into bytes. Then, the lowest byte is stored in *AryOut[0].* Then, the next byte is stored in *AryOut[1].* This process is repeated for the number of bytes that is specified in number of bytes to separate *Num*.

The following example is for when *Num* is USINT#3.



Additional Information

Use the Unite8Bit_** instruction (page 2-451) to join 1-byte units from array elements.

Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the number of bytes in *In*.
  - *AryOut[]* is not a BYTE array.
  - An array without a subscript is passed to *AryOut[]*.

Unite8Bit_**

The Unite8Bit_** instructions join bytes of data into a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Unite8Bit_**	Byte Data Join Group	FUN	(@)Unite8Bit_** EN In Num ENO Out "**" must be a bit string data type.	Out:=Unite8Bit_**(In, Num); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to join	Input	Array to join	Depends on data type.	---	*
Num	Number of bytes to join		Number of bytes to join	0 to No. of bytes in *Out*		1
Out	Joined result	Output	Bit string with joined result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Num						OK														
Out		OK	OK	OK	OK															

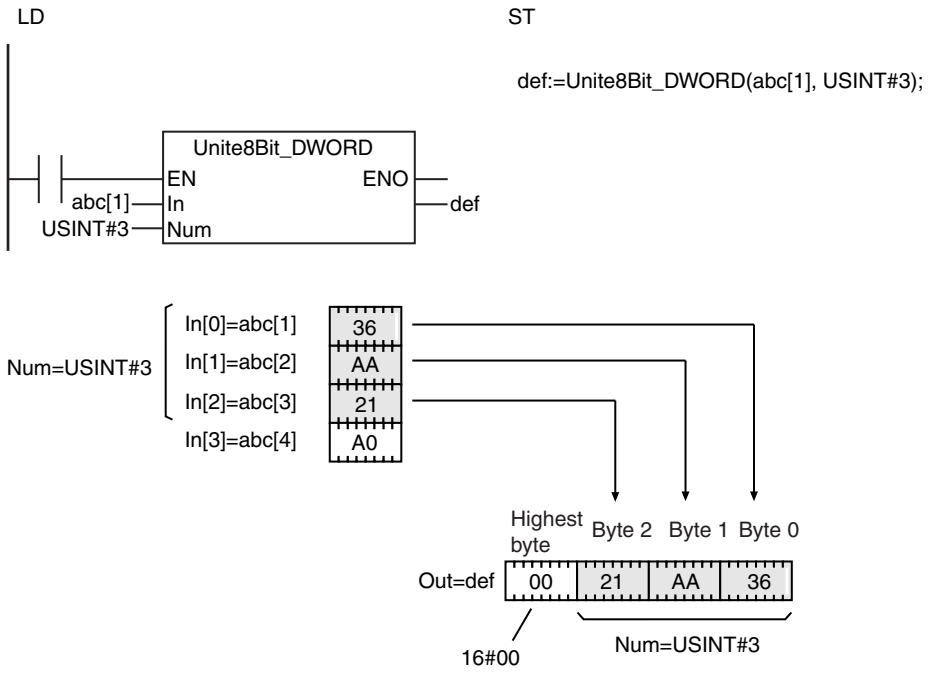
Function

The Unite8Bit_** instructions join elements of array to join *In[]* to create a bit string in joined result *Out*.

Number of bytes to join *Num* specifies the number of array elements to join. First, *In[0]* to *In[Num–1]* are joined to create a bit string with *Num* bytes. To this, 16#00 is added to the upper bytes for the number of bytes of *Out* minus the value of *Num*. The result is stored in *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if Out is the DWORD data type, the instruction is Unite8Bit_DWORD.

The following example shows the Unite8Bit_DWORD instruction when *Num* is USINT#3.



Additional Information

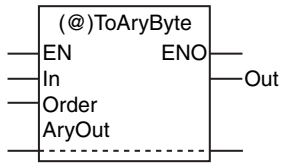
Use the Dispart8Bit instruction (page 2-449) to separate a bit string into 1-byte units.

Precautions for Correct Use

- If the value of Num is 0, the value of *Out* is 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the array area of *In[]*.
  - *In[]* is not a BYTE array.
  - An array without a subscript is passed to *In[]*.

ToAryByte

The ToAryByte instruction separates a variable into bytes and stores the bytes in a BYTE array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ToAryByte	Conversion to Byte Array	FUN		Out:=ToAryByte(In, Order, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Order	Conversion order		Conversion order	_LOW_HIGH or _HIGH_LOW		_LOW _HIGH
AryOut[] (array)	Conversion results array	In-out	Conversion results array	Depends on data type.	---	---
Out	Number of elements in result	Output	Number of elements in result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Order	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
AryOut[] (array)		OK																		
Out							OK													

Function

The ToAryByte instruction separates the value of data to convert *In* into individual bytes and stores them in order in conversion results array *AryOut[]* starting from *AryOut[0].* Number of elements in result *Out* contains the number of elements stored in *AryOut[].*

Conversion order *Order* specifies the order in which to convert the value of *In* to bytes. The data type of *Order* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerator	Meaning
_LOW_HIGH	Lower byte first, higher byte last
_HIGH_LOW	Higher byte first, lower byte last

When the Data Type of *In* Is Two Bytes or Larger

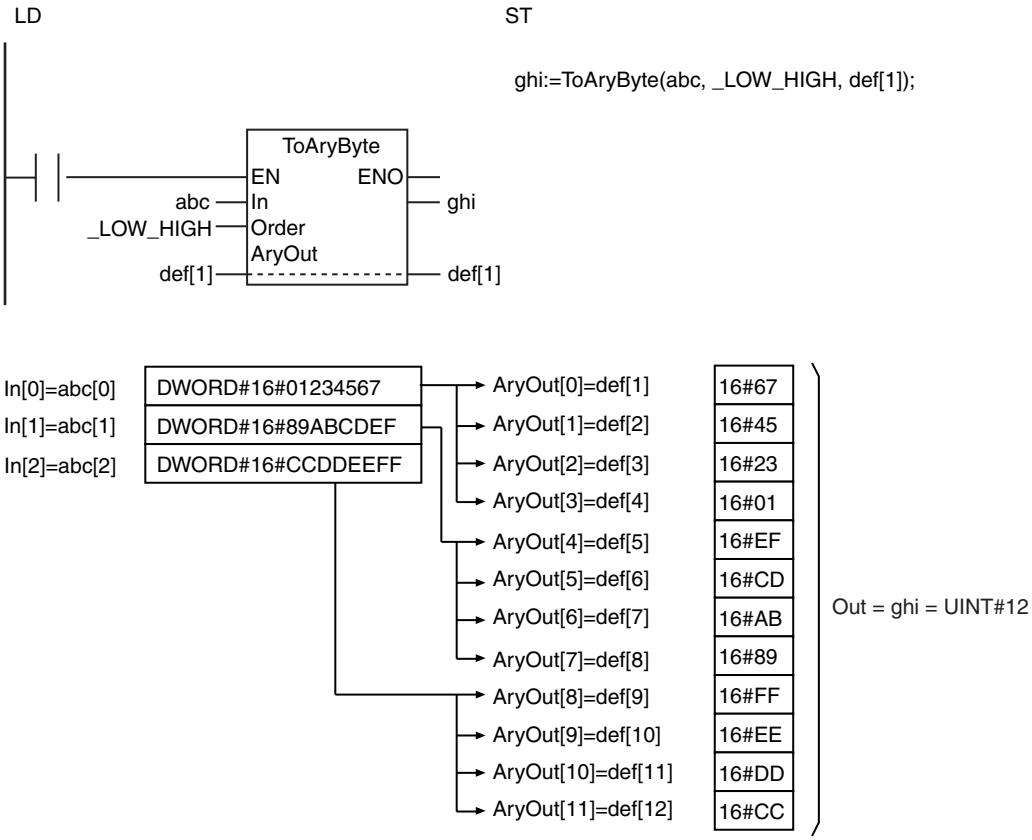
If the data type of *In* is two bytes or larger, *In* is separated into bytes and stored in *AryOut[]*. The following data types have two bytes or more.

Classification	Data type
Bit strings	WORD, DWORD, and LWORD
Integers	UINT, UDINT, ULINT, INT, DINT, and LINT
Real numbers	REAL and LREAL
Times, durations, dates, and text strings	TIME, DATE, TOD, DT, and STRING types of two bytes or more
Others	An enumeration, an array for which the total for all elements is 2 bytes or more, an array element that is 2 bytes or more, a structure for which the total for all members is 2 bytes or more, or a structure member that is 2 bytes or more

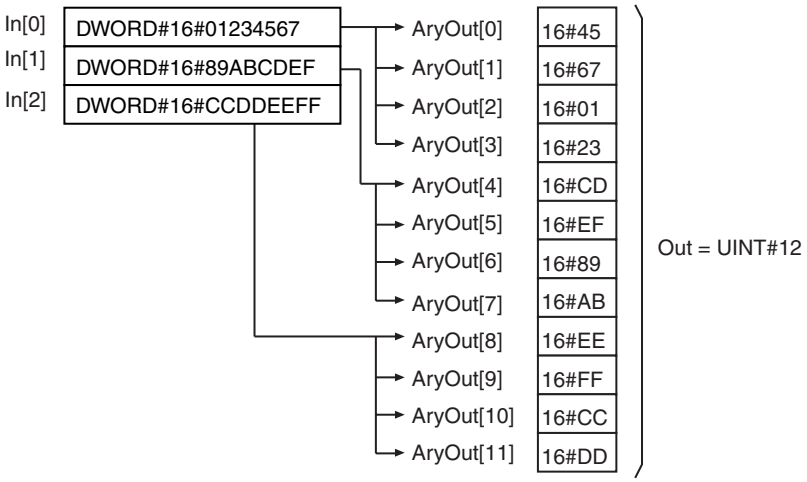
The processing procedure is as follows:

1. First, the value in *In* is separated into words (two bytes).
2. The lowest word is separated into bytes.
3. If *Order* is *_LOW_HIGH*, the lower byte is stored in *AryOut[0]* and the higher byte is stored in *AryOut[1].* If *Order* is *_HIGH_LOW*, the higher byte is stored in *AryOut[0]* and the lower byte is stored in *AryOut[1].*
4. The next word is separated into bytes and stored in *AryOut[2]* and *AryOut[3]* in the same way.
5. This process is repeated to the end of the value of *In*. If *In* is an array, the same process is repeated to the last element in *In*.

The following example is for when *In* is a DWORD array with three elements and *Order* is *_LOW_HIGH*.



The following example is for when *In* is the same as above and *Order* is *_HIGH_LOW*.



When the Data Type of *In* Is One Byte

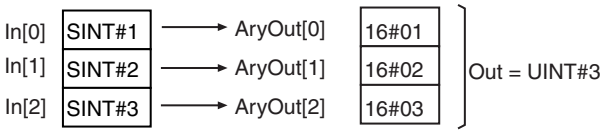
If the data type of *In* is one byte, *In* is stored in *AryOut[]* as one byte. The following data types have one byte.

Classification	Data type
Bit strings	BYTE
Integers	USINT and SINT
Real numbers	None
Times, durations, dates, and text strings	STRING types with one byte
Others	An array for which the total for all elements is 1 byte, an array element that is 1 byte, a structure for which the total for all members is 1 byte, or a structure member that is 1 byte.

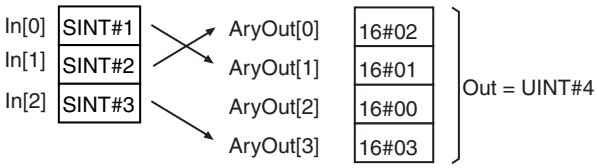
The following storage method is used.

Value of *Order*	*In* (array or not)	Storage method in *AryOut[]*
_LOW_HIGH	Not an array	Value of *In* is stored in *AryOut[0].*
	Array	Value of *In[i]* is stored in *AryOut[i].*
_HIGH_LOW	Not an array	Value of *In* is stored in *AryOut[1].* 16#00 is stored in *AryOut[0].*
	Array	*In[i]* (where *i* is even) is stored in *AryOut[i+1].* *In[i]* (where *i* is odd) is stored in *AryOut[i−1].* If the number of elements in *In[]* is odd, 16#00 is stored last in *AryOut[n−1].*

The following example is for when *In* is a SINT array with three elements and *Order* is *_LOW_HIGH*.



The following example is for when *In* is the same as above and *Order* is *_HIGH_LOW*.

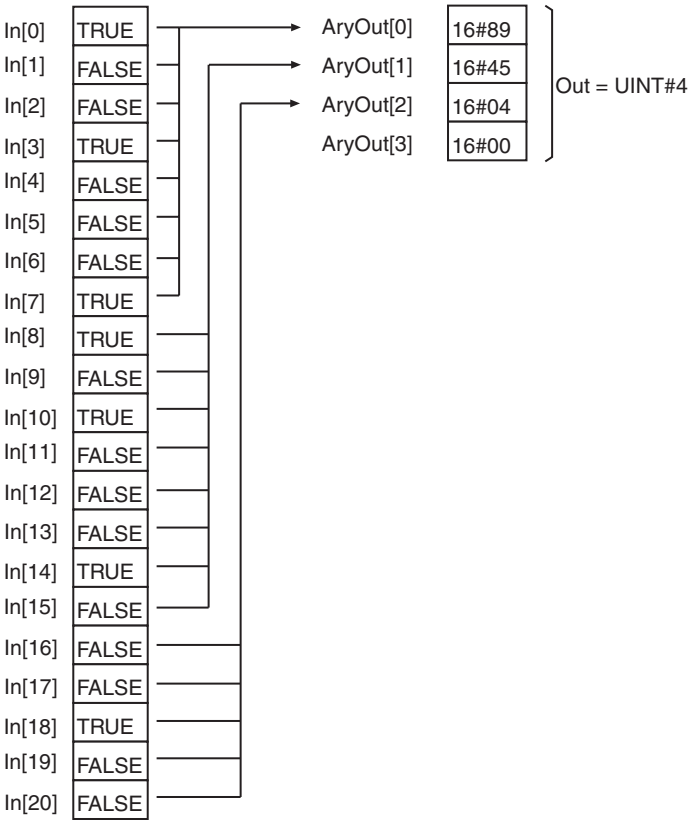


When *In* Is BOOL Data

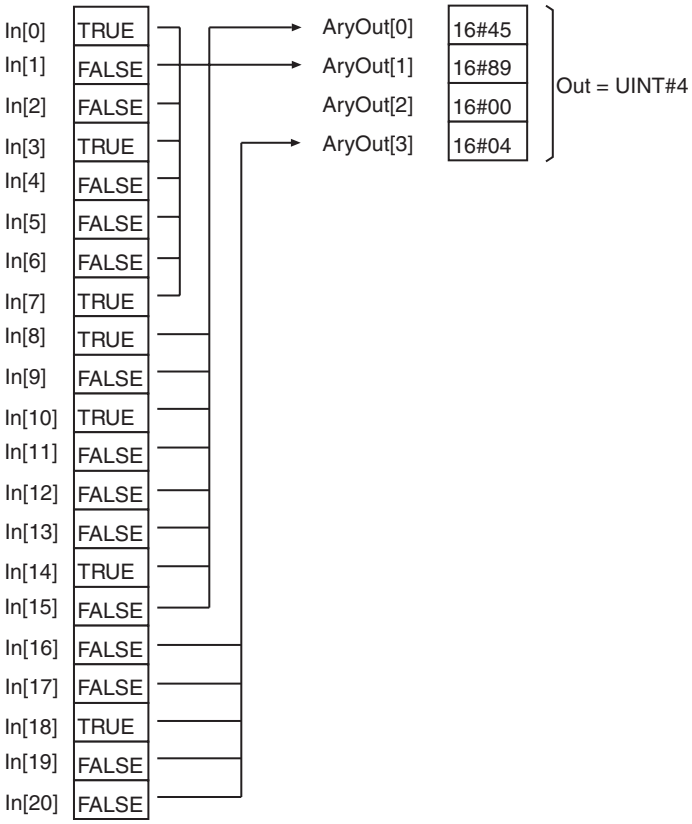
If the data type of *In* is BOOL (one bit), data is stored in *AryOut[]* as described below.

Value of *Order*	*In* (array or not)	Storage method in *AryOut[]*
_LOW_HIGH	Not an array	The logical OR of the value of *In* and 16#00 is stored in *AryOut[0].*
	Array	Values of *In[0]* to *In[7]* are joined and stored in *AryOut[0].* Values of *In[8]* to *In[15]* are joined and stored in *AryOut[1].* The same process is repeated to store the rest of the data. If there is not sufficient data in *In[]* for 8 values, FALSE is added to the most-significant bit. The value of *Out* is always even. If there are not sufficient bit values, the remaining values will all be FALSE.
_HIGH_LOW	Not an array	The logical OR of the value of *In* and 16#00 is stored in *AryOut[1].* 16#00 is stored in *AryOut[0]*
	Array	Values of *In[0]* to *In[7]* are joined and stored in *AryOut[1].* Values of *In[8]* to *In[15]* are joined and stored in *AryOut[0].* The same process is repeated to store the rest of the data. The value of *Out* is always even. If there are not sufficient bit values, the remaining values will all be FALSE.

The following example is for when *In* is a BOOL array with 21 elements and *Order* is *_LOW_HIGH*.



The following example is for when *In* is the same as above and *Order* is *_HIGH_LOW*.

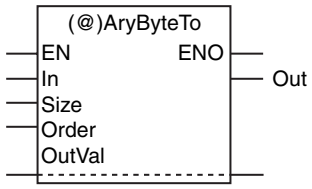


Precautions for Correct Use

- If *In* is STRING data, the text string is not converted to numbers. The contents of the variable is taken as a bit string and converted to a byte array.
- If *In* is a structure, adjustment areas between members may be inserted into AryOut[].
- If the value of *Order* is *_HIGH_LOW* and the total number of bytes in *In* is an odd number, 16#00 is added to the end of *In* to make an even number of bytes before the conversion is started.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* and *AryOut[]* will not change.
  - The value of *Order* is outside of the valid range.
  - The conversion result exceeds the array area of *AryOut[]*.

AryByteTo

The AryByteTo instruction joins BYTE array elements and stores the result in a variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryByteTo	Conversion from Byte Array	FUN		AryByteTo(In, Size, Order, OutVal);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to convert	Input	Array to convert	Depends on data type.	---	*
Size	Number of elements to convert		Number of elements in *In[]* to convert			1
Order	Conversion order		Conversion order	_LOW_HIGH or _HIGH_LOW		_LOW _HIGH
OutVal	Conversion result	In-out	Conversion result	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Size							OK													
Order	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
OutVal	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Out	OK																			

Function

The AryByteTo instruction takes the first *Size* elements in array to convert *In[]* and joins them to match the size of the data type of conversion result *OutVal*. It then stores the result in *OutVal*.

Order specifies the order to join the elements of *In[].* The data type of *Order* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, higher byte last
_HIGH_LOW	Higher byte first, lower byte last

When the Data Type of *OutVal* Is Two Bytes or Larger

If the data type of *OutVal* is two bytes or larger, elements from *In[]* are joined so that the result is just large enough for the size of the data type of *OutVal*. The result is stored in *OutVal*. The following data types have two bytes or more.

Classification	Data type
Bit strings	WORD, DWORD, and LWORD
Integers	UINT, UDINT, ULINT, INT, DINT, and LINT
Real numbers	REAL and LREAL
Times, durations, dates, and text strings	TIME, DATE, TOD, DT, and STRING types of two bytes or more
Others	An enumeration, an array for which the total for all elements is 2 bytes or more, an array element that is 2 bytes or more, a structure for which the total for all members is 2 bytes or more, or a structure member that is 2 bytes or more

The processing procedure is as follows:

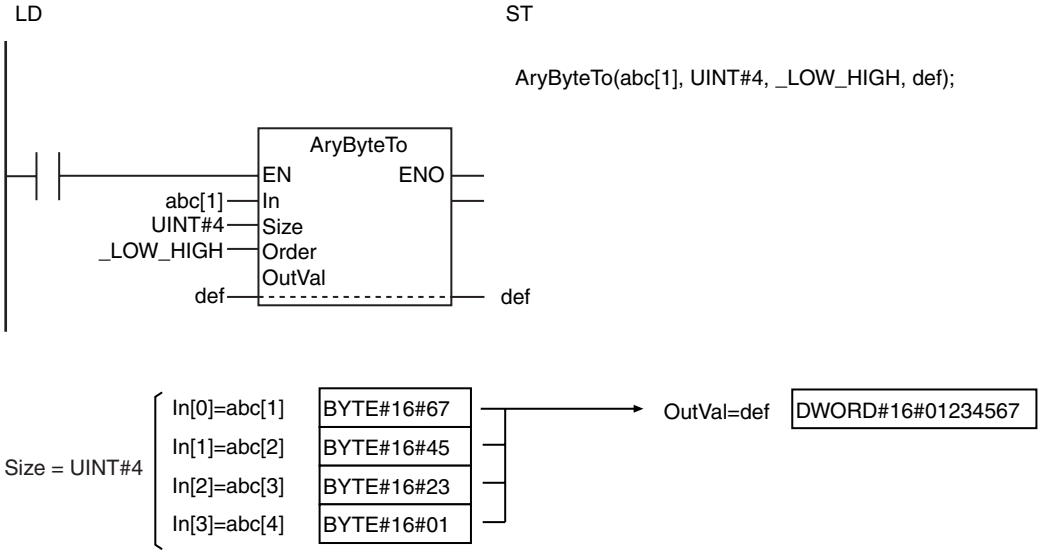
1 *In[0]* and *In[1]* are joined according to the value of *Order* to create one word (two bytes) of data. If *Order* is *_LOW_HIGH*, the higher byte is stored in *In[1]* and the lower byte is stored in *In[0]*. If *Order* is *_HIGH_LOW*, the higher byte is stored in *In[0]* and the lower byte is stored in *In[1].*

2 In the same way elements that start from *In[2]* and *In[3]* are joined to make more words of data.

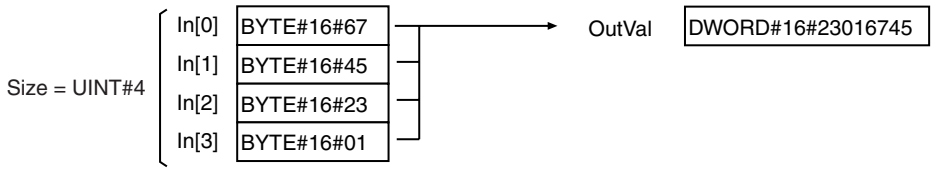
3 The words of data are joined to match the size of the data type of *OutVal*. For example, if *OutVal* is DWORD data, four individual words of data are joined.

4 The resulting data is stored in *OutVal*.

The following example is for when *OutVal* is DWORD data, *Size* is UINT#4, and *Order* is *_LOW_HIGH*.



The following example is for when *OutVal* is the same as above, *Size* is UINT#4, and *Order* is *_HIGH_LOW*.



When the Data Type of *OutVal* Is One Byte

If the data type of *OutVal* is one byte, one byte of *In[]* is stored directly in *OutVal*.

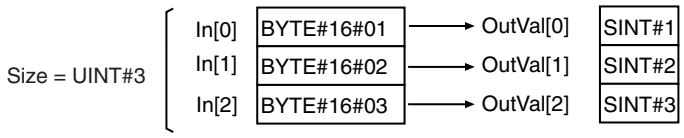
The following data types have one byte.

Classification	Data type
Bit strings	BYTE
Integers	USINT and SINT
Real numbers	None
Times, durations, dates, and text strings	STRING types with one byte
Others	An array for which the total for all elements is 1 byte, an array element that is 1 byte, a structure for which the total for all members is 1 byte, or a structure member that is 1 byte.

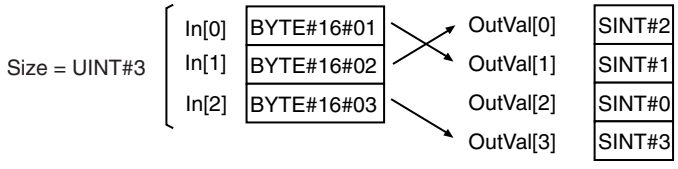
The following storage method is used.

Value of *Order*	*OutVal* (array or not)	Storage method in *OutVal*
_LOW_HIGH	Not an array	Value of *In[0]* is stored in *OutVal*
	Array	Value of *In[i]* is stored in *OutVal[i]*
_HIGH_LOW	Not an array	Value of *In[1]* is stored in *OutVal*
	Array	*In[i]* (where *i* is even) is stored in *OutVal[i+1]*. *In[i]* (where *i* is odd) is stored in *OutVal[i−1]*. If the value of *Size* is odd, data is stored up to *OutVal[Size]* and 16#00 is stored in *OutVal[Size*−1*]*.

The following example is for when *OutVal* is a SINT array with three elements, *Size* is UINT#3, and *Order* is *_LOW_HIGH*.



The following example is for when *OutVal* and *Size* are the same as above and *Order* is *_HIGH_LOW*.

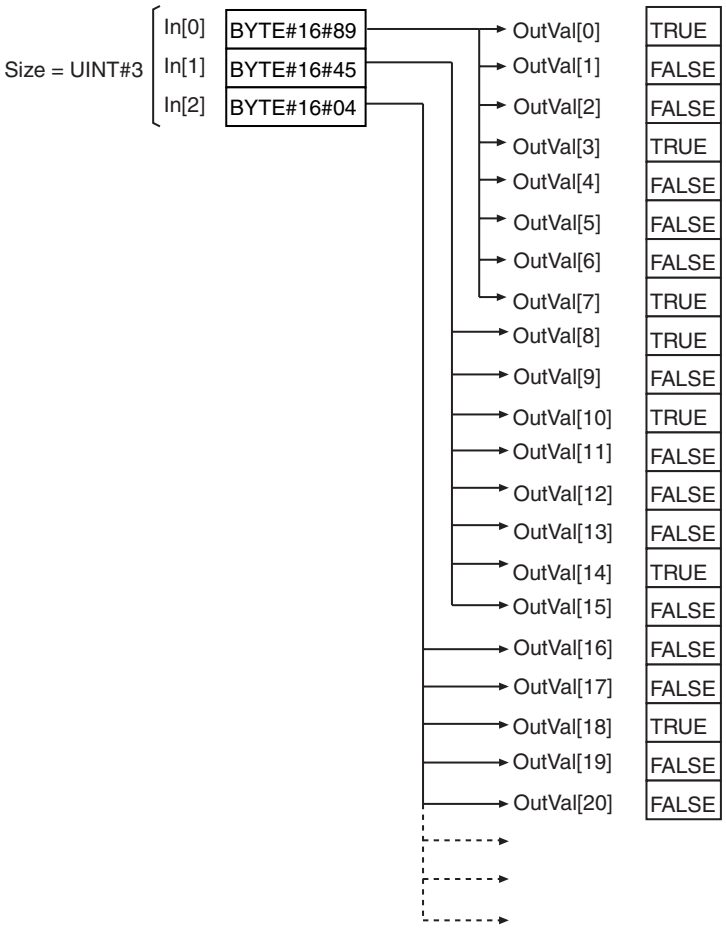


When *OutVal* Is BOOL Data

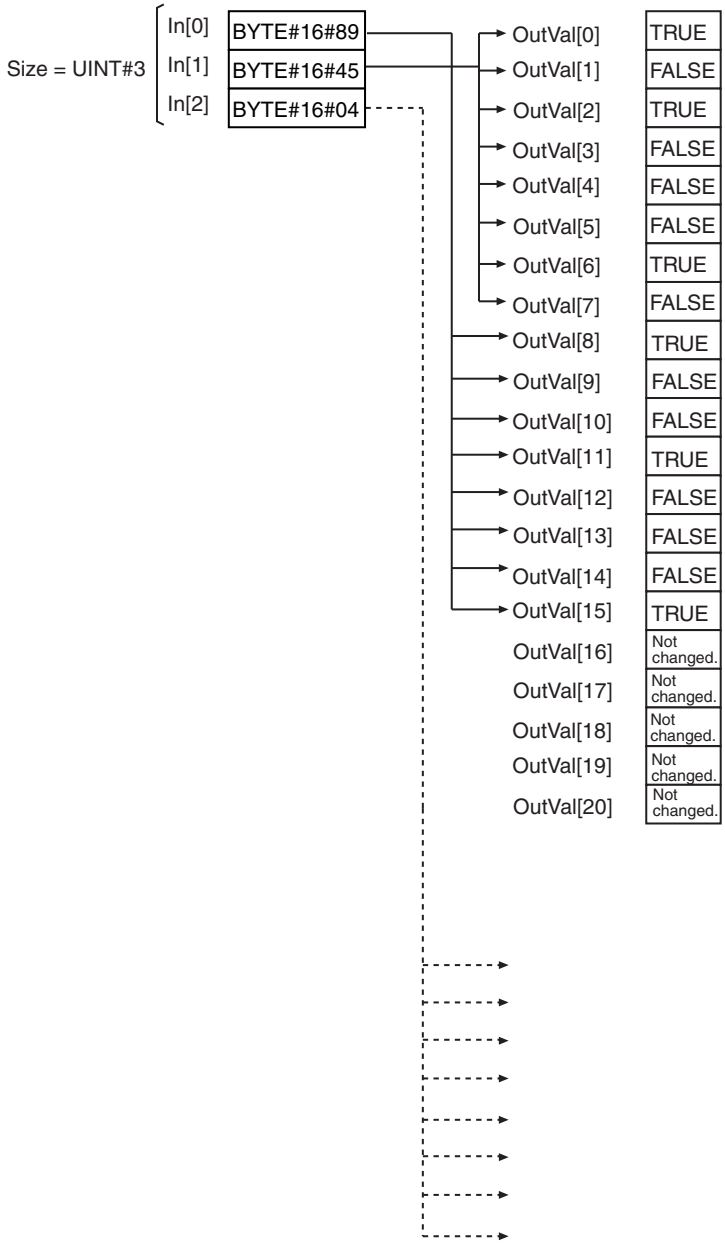
If the data type of *OutVal* is BOOL (one bit), data is stored in *OutVal* as described below.

Value of *Order*	*OutVal* (array or not)	Storage method in *OutVal*
_LOW_HIGH	Not an array	Value of bit 0 of *In[0]* is stored in *OutVal.*
	Array	Value of *In[0]* is separated and stored in *OutVal[0]* to *OutVal[7].* Value of *In[1]* is separated and stored in *OutVal[8]* to *OutVal[15].* The same process is repeated to store the rest of the data. Remaining bits are discarded.
_HIGH_LOW	Not an array	Value of bit 0 of *In[1]* is stored in *OutVal.*
	Array	Value of *In[0]* is separated and stored in *OutVal[8]* to *OutVal[15].* Value of *In[1]* is separated and stored in *OutVal[0]* to *OutVal[7].* The same process is repeated to store the rest of the data. Remaining bits are discarded.

The following example is for when *OutVal* is a BOOL array with 21 elements, *Size* is UINT#3, and *Order* is *_LOW_HIGH*.



The following example is for when *OutVal* and *Size* are the same as above and *Order* is *_HIGH_LOW*.



Precautions for Correct Use

- If *OutVal* is a structure, some of the values of *In[]* may be inserted in adjustment areas between members depending on the composition.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *OutVal* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *OutVal* will not change.
  - The value of *Order* is outside of the valid range.
  - The value of *Size* exceeds the number of elements in *In[]*.

SizeOfAry

The SizeOfAry instruction gets the number of elements in an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SizeOfAry	Get Number of Array Elements	FUN	(@)SizeOfAry EN ENO In Out	Out:=SizeOfAry(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array	Input	Array	Depends on data type.	---	*
Out	Number of elements	Output	Number of elements	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Out							OK													

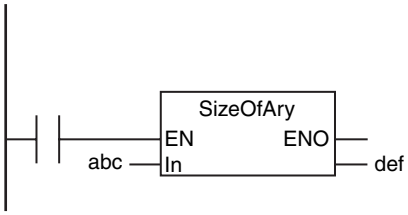
Function

The SizeOfAry instruction gets the number of elements in array *In[]*. For the input parameter, use an array name, such as *array*, and not an array element name, such as *array[0]*.

The following figure shows a programming example.

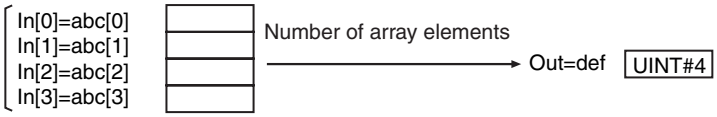
Variable	Data type
abc	ARRAY[0..3] OF INT

LD



ST

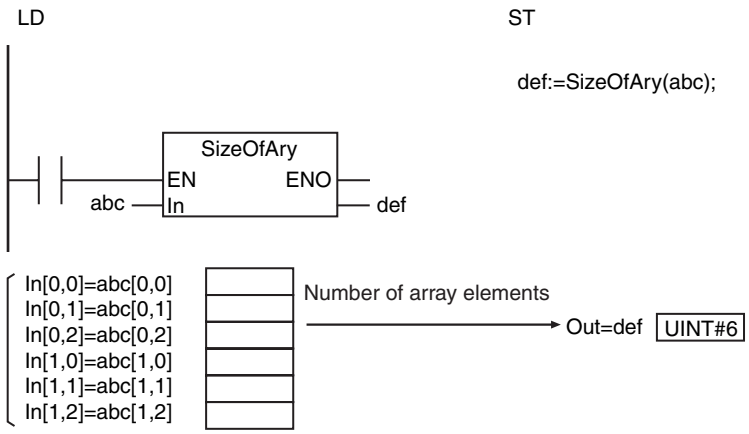
```
def:=SizeOfAry(abc);
```



Additional Information

In[] can be an array with two or more dimensions. *Out* will contain the total number of elements for all dimensions of *In[]*. For example, if the input parameter that is passed to *In[]* is *ARRAY[0..1,0..2]*, the value of *Out* will be UINT#6.

Variable	Data type
abc	ARRAY[0..1,0..2] OF BOOL

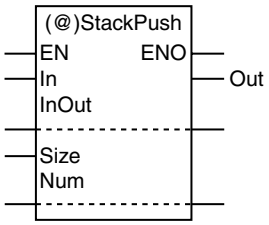


Stack and Table Instructions

Instruction	Name	Page
StackPush	Push onto Stack	2-466
StackFIFO and StackLIFO	First In First Out/Last In First Out	2-475
StackIns	Insert into Stack	2-478
StackDel	Delete from Stack	2-480
RecSearch	Record Search	2-482
RecRangeSearch	Range Record Search	2-487
RecSort	Record Sort	2-492
RecNum	Get Number of Records	2-497
RecMax and RecMin	Maximum Record Search/ Minimum Record Search	2-499

StackPush

The StackPush instruction stores a value at the top of a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackPush	Push onto Stack	FUN		StackPush(In, InOut, Size, Num);

Variables

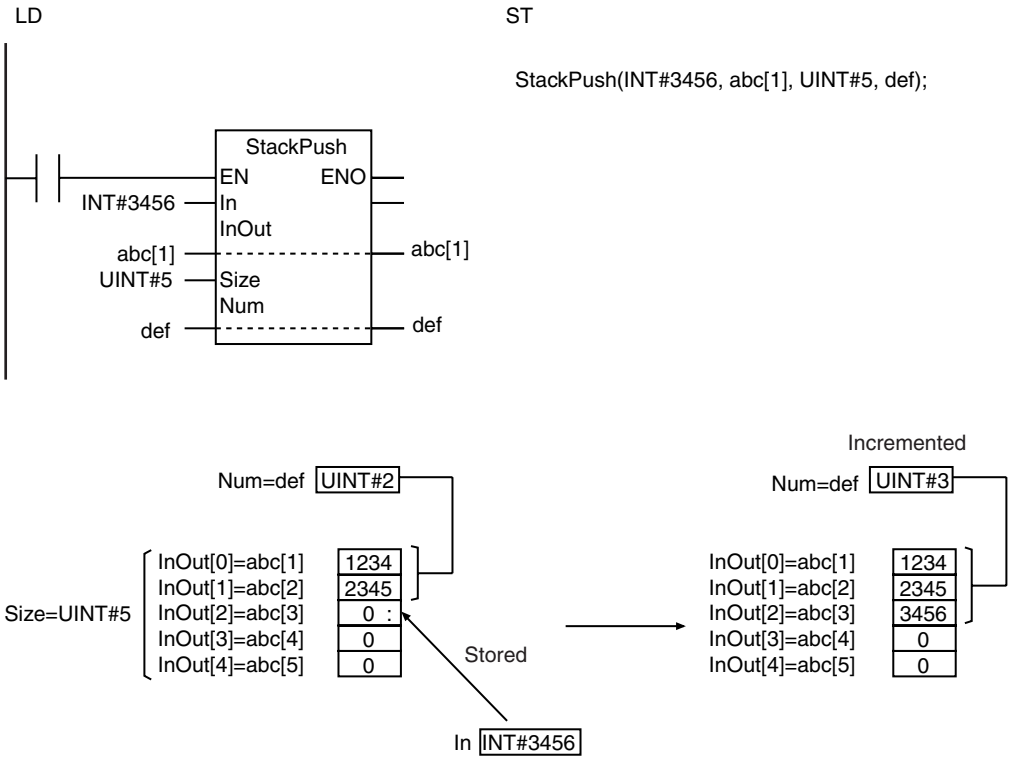
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Input value	Input	Value, structure, or structure member to place in the stack	Depends on data type.	---	---
Size	Number of stack elements		Number of stack array elements			1
InOut[] (array)	Stack array	In-out	Array that functions as stack	Depends on data type.	---	---
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
Size							OK													
InOut[] (array)	Must be an array with elements that have the same data type as *In*.																			
Num							OK													
Out	OK																			

Function

The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. Input value *In* is written to the next element, *InOut[Num]*. Then, *Num* is incremented. For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

The following example is for when *Size* is UINT#5 and *Num* is UINT#2.



Additional Information

Use the StackFIFO or StackLIFO instruction (page 2-475) to remove the bottom or top value that was stored in the stack.

Precautions for Correct Use

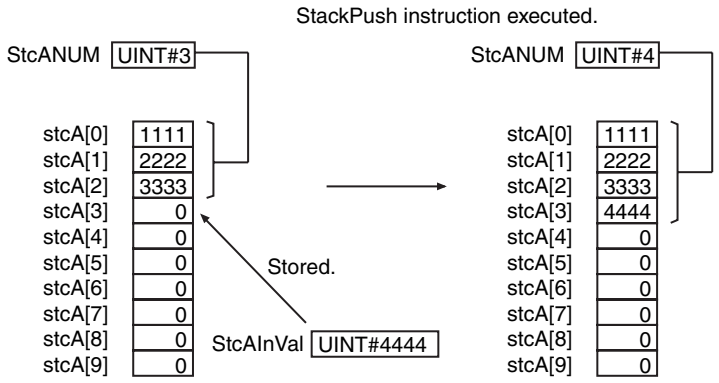
- Use the same data type for *In* and the elements of *InOut[]*.
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The value of *InOut[]* or *Num* does not change if the value of *Size* is 0.
- When *In* is an enumeration, always use a variable for the input parameter to pass to *In*. A building error will occur if a constant is passed.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - *In* and *InOut[]* contain different data types.
  - The value of *Size* is not 0 and *Num* is greater than or equal to *Size*.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *In* is STRING data and it does not end in a NULL character.
  - *In* and *InOut[]* are STRING data and the number of bytes in *In* exceeds the size of *InOut[]*.

Sample Programming

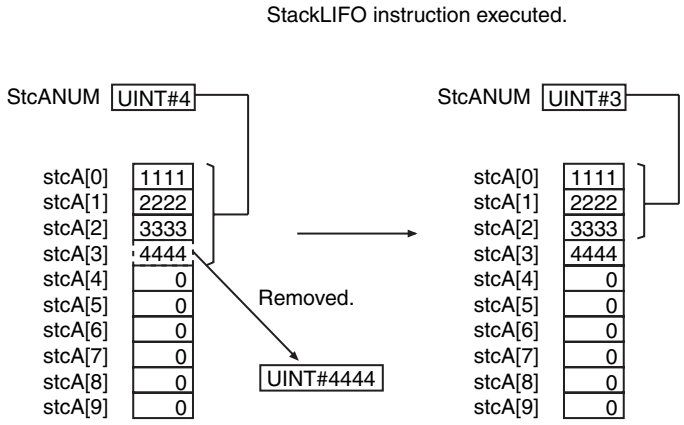
The array variable *stcA[0..9]* is used as a stack. As preparations, three values (UINT#1111, UINT#2222, and UINT#3333) are stored in the stack.

stcA[0]	1111
stcA[1]	2222
stcA[2]	3333
stcA[3]	0
stcA[4]	0
stcA[5]	0
stcA[6]	0
stcA[7]	0
stcA[8]	0
stcA[9]	0

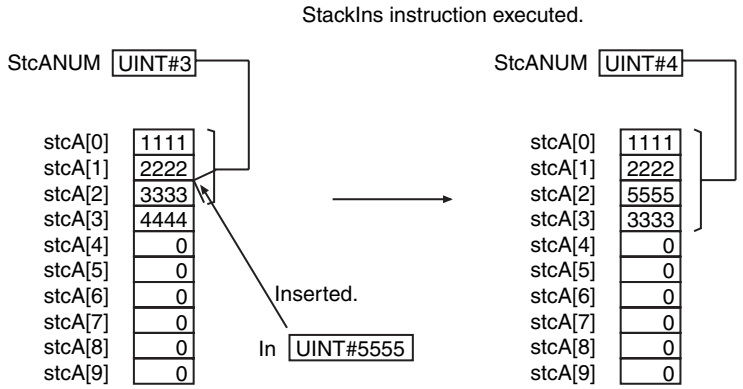
The StackPush instruction is used to store a new value (UINT#4444) at the top of the stack *stcA[3]*. That means there will be four values in the stack.



Then, the StackLIFO instruction is used to remove one value at the top of the stack *stcA[3]*. That means there will be three values in the stack.

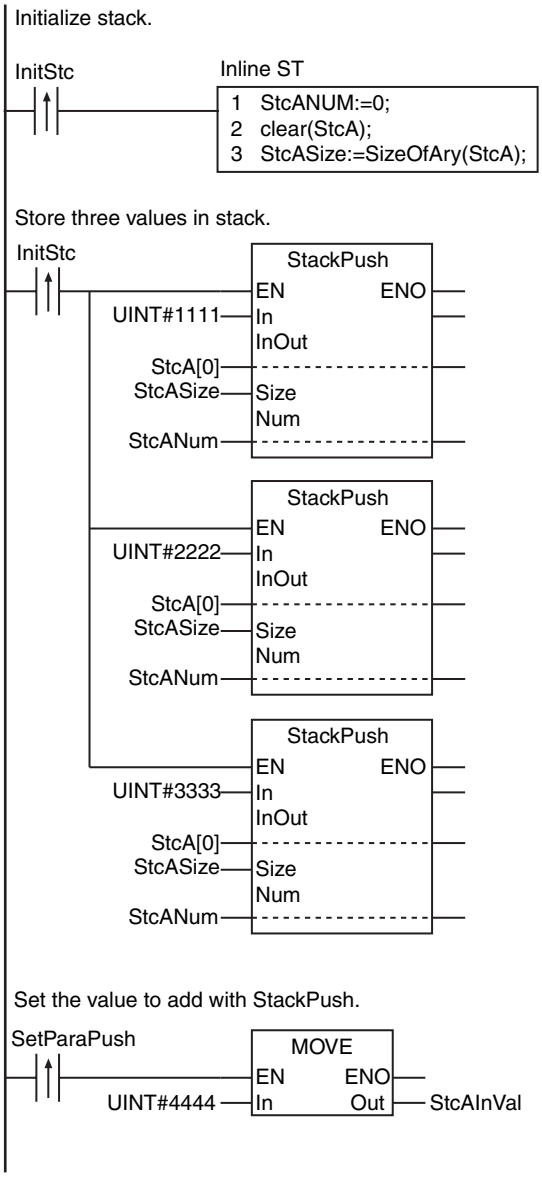


Finally, the StackIns instruction is used to insert a value (UINT#5555) between *stcA[1]* and *stcA[2]*. That means there will be four values in the stack.

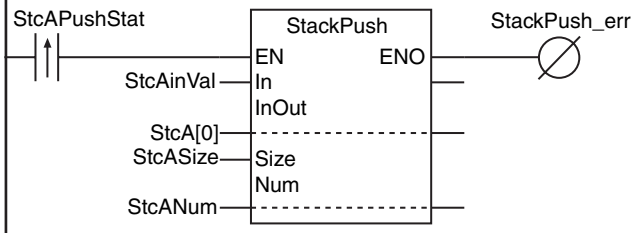


LD

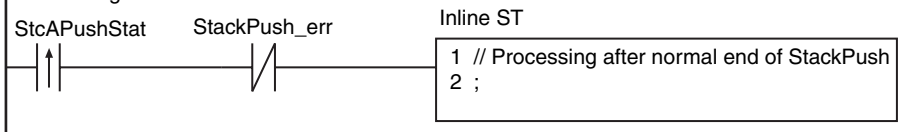
Variable	Data type	Initial value	Comment
InitStc	BOOT	False	Stack initialization condition
stcANum	UINT	0	Number of stored elements
StcA	ARRAY[0..9] OF UINT	[10(0)]	Stack array
StcASize	UINT	0	Number of stack elements
SetParaPush	BOOL	False	Execution condition to set *StcAInVal*.
StcAInVal	UINT	0	Value added by StackPush
StcAPushStat	BOOL	False	StackPush execution condition
StackPush_err	BOOL	False	StackPush error flag
StcALIFOStat	BOOL	False	StackLIFO execution condition
StcAOutVal	UINT	0	Value removed by StackLIFO
StackLIFO_err	BOOL	False	StackLIFO error flag
SetParaIns	BOOL	False	Execution condition to set *StcAInsVal* and *StcAOffset*
StcAInsVal	UINT	0	Value inserted by StackIns
StcAOffset	UINT	0	Offset for StackIns
StcAInsStat	BOOL	False	StackIns execution condition
StackIns_err	BOOL	False	StackIns error flag



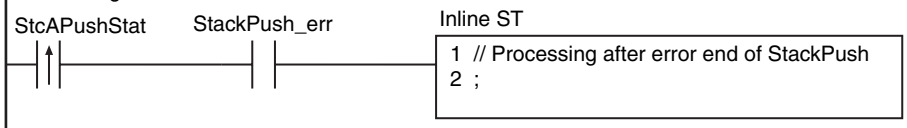
Add data with StackPush instruction.



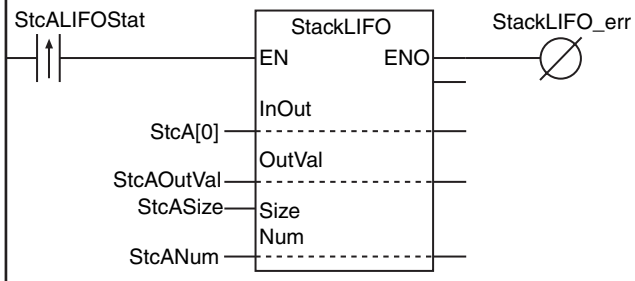
Processing after normal end of StackPush



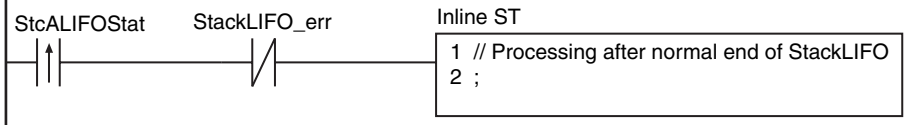
Processing after error end of StackPush



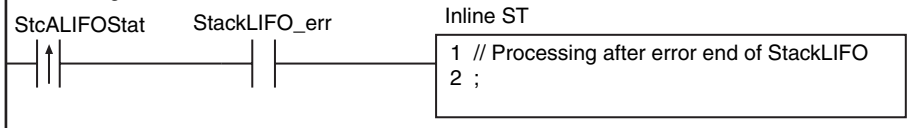
Remove data with StackLIFO instruction.



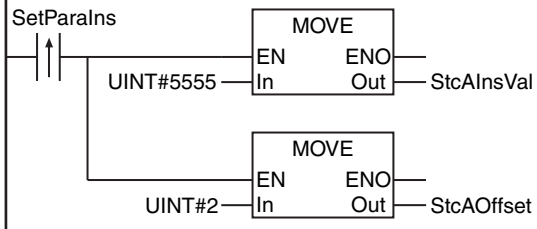
Processing after normal end of StackLIFO

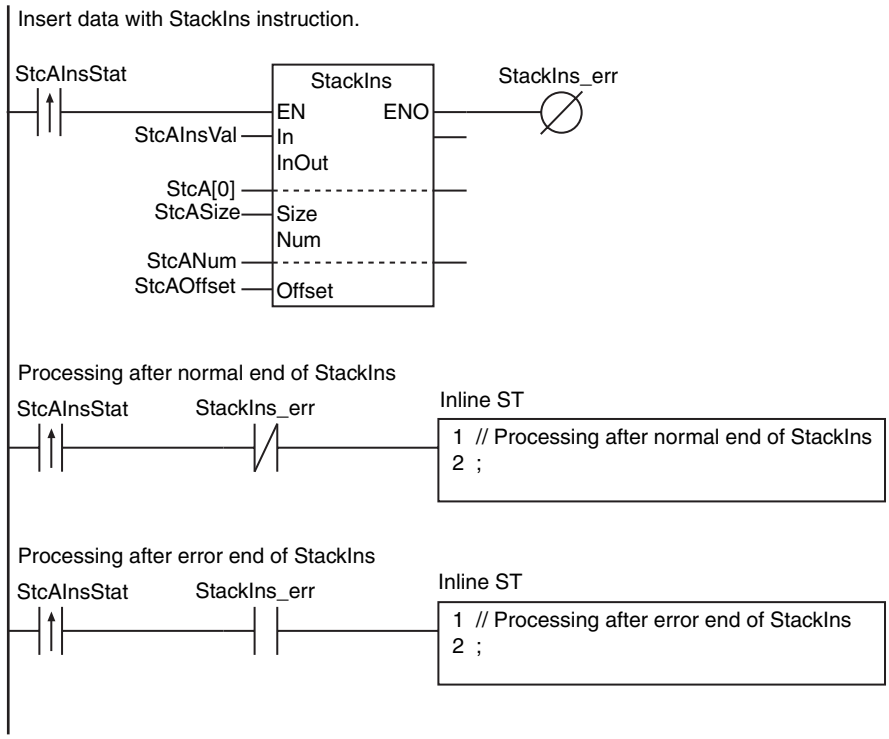


Processing after error end of StackLIFO



Set the insert value and offset with StackInsh.




Insert data with StackIns instruction.
StcAInsStat
StackIns
EN
ENO
StackIns_err
StcAInsVal
In
InOut
StcA[0]
StcASize
Size
Num
StcANum
StcAOffset
Offset
Processing after normal end of StackIns
StcAInsStat
StackIns_err
Inline ST
1 // Processing after normal end of StackIns
2 ;
Processing after error end of StackIns
StcAInsStat
StackIns_err
Inline ST
1 // Processing after error end of StackIns
2 ;

ST

Variable	Data type	Initial value	Comment
InitStc	BOOL	False	Stack initialization condition
preInitStc	BOOL	False	Value of *InitStc* from previous task period
stcANum	UINT	0	Number of stored elements
StcA	ARRAY[0..9] OF UINT	[10(0)]	Stack array
StcASize	UINT	0	Number of stack elements
StcAPushStat	BOOL	False	StackPush execution condition
preStcAPushStat	BOOL	False	Value of *StcAPushStat* from previous task period
StcAInVal	UINT	0	Value added by StackPush
StcAPush_OK	BOOL	False	StackPush normal end flag
StcAPushNormalEnd	BOOL	False	Processing after normal end of StackPush
StcAPushErrorEnd	BOOL	False	Processing after error end of StackPush
StcALIFOStat	BOOL	False	StackLIFO execution condition
preStcALIFOStat	BOOL	False	Value of *StcALIFOStat* from previous task period
StcAOutVal	UINT	0	Value removed by StackLIFO
StcALIFO_OK	BOOL	False	StackLIFO normal end flag
StcALIFONormalEnd	BOOL	False	Processing after normal end of StackLIFO
StcALIFOErrorEnd	BOOL	False	Processing after error end of StackLIFO
StcAInsStat	BOOL	False	StackIns execution condition
preStcAInsStat	BOOL	False	Value of *StcAInsStat* from previous task period
StcAInsVal	UINT	0	Value inserted by StackIns
StcAOffset	UINT	0	Offset for StackIns
StcAIns_OK	BOOL	False	StackIns normal end flag
StcAInsNormalEnd	BOOL	False	Processing after normal end of StackIns
StcAinsErrorEnd	BOOL	False	Processing after error end of StackIns

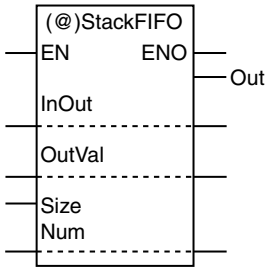
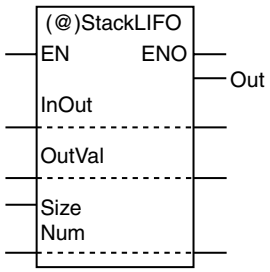
```
// Initialize stack.
IF ( (InitStc=TRUE) AND (preInitStc=FALSE) ) THEN
    StcANum:=0;
    Clear(StcA);
    StcASize:=SizeOfAry(StcA);
END_IF;

// Store three values in stack.
IF ( (InitStc=TRUE) AND (preInitStc=FALSE) ) THEN
    StackPush(In:=UINT#1111, InOut:=StcA[0], Size:=StcASize, Num:=StcANum);
    StackPush(In:=UINT#2222, InOut:=StcA[0], Size:=StcASize, Num:=StcANum);
    StackPush(In:=UINT#3333, InOut:=StcA[0], Size:=StcASize, Num:=StcANum);
END_IF;

preInitStc:=InitStc;

// Add data with StackPush instruction.
IF ( (StcAPushStat=TRUE) AND (preStcAPushStat=FALSE) ) THEN
    StcAInVal:=UINT#4444;
    StackPush(
        In     :=StcAinVal,          // Value to add
        InOut:=StcA[0],              // First element in stack array
        Size  :=StcASize,            // Number of stack elements
        Num  :=StcANum,              // Number of stored elements
        ENO  =>StcAPush_OK);  // Normal end flag
    IF (StcAPush_OK=TRUE) THEN
        StcAPushNormalEnd:=TRUE;  // Processing after normal end
    ELSE
        StcAPushErrorEnd:=TRUE;   // Processing after error end
    END_IF;
END_IF;
preStcAPushStat:=StcAPushStat;
```

```
// Remove data with StackLIFO instruction.
IF ( (StcALIFOStat=TRUE) AND (preStcALIFOStat=FALSE) ) THEN
    StackLIFO(
        InOut  :=StcA[0],              // First element in stack array
        OutVal:=StcAOutVal,            // Value removed from stack
        Size    :=StcASize,            // Number of stack elements
        Num    :=StcANum,             // Number of stored elements
        ENO    =>StcALIFO_OK);        // Normal end flag
    IF (StcALIFO_OK=TRUE) THEN
        StcALIFONormalEnd:=TRUE;    // Processing after normal end
    ELSE
        StcALIFOErrorEnd:=TRUE;       // Processing after error end
    END_IF;
END_IF;
preStcALIFOStat:=StcALIFOStat;

// Insert data with StackIns instruction.
IF ( (StcAInsStat=TRUE) AND (preStcAInsStat=FALSE) ) THEN
    StcAInsVal:=UINT#5555;
    StcAOffset:=UINT#2;
    StackIns(
        In      :=StcAInsVal,          // Value to insert into stack
        InOut:=StcA[0],                // First element in stack array
        Size  :=StcASize,              // Number of stack elements
        Num  :=StcANum,                // Number of stored elements
        Offset:=StcAOffset,            // Offset at which to insert value
        ENO  =>StcAIns_OK);            // Normal end flag
    IF (StcAIns_OK=TRUE) THEN
        StcAInsNormalEnd:=TRUE;       // Normal end flag
    ELSE
        StcAInsErrorEnd:=TRUE;         // Processing after error end
    END_IF;
END_IF;
preStcAInsStat:=StcAInsStat;
```

StackFIFO and StackLIFO

StackFIFO: Removes the bottom value from a stack.

StackLIFO: Removes the top value from a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackFIFO	First In First Out	FUN		StackFIFO(InOut, OutVal, Size, Num);
StackLIFO	Last In First Out	FUN		StackLIFO(InOut, OutVal, Size, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of stack elements	Input	Number of stack array elements	Depends on data type.	---	1
InOut[] (array)	Stack array	In-out	Array that functions as stack	Depends on data type.	---	---
OutVal	Output value		Value or structure output from stack			
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
InOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
OutVal	Must be the same data type as the elements of *InOut[]*.																			
Num							OK													
Out	OK																			

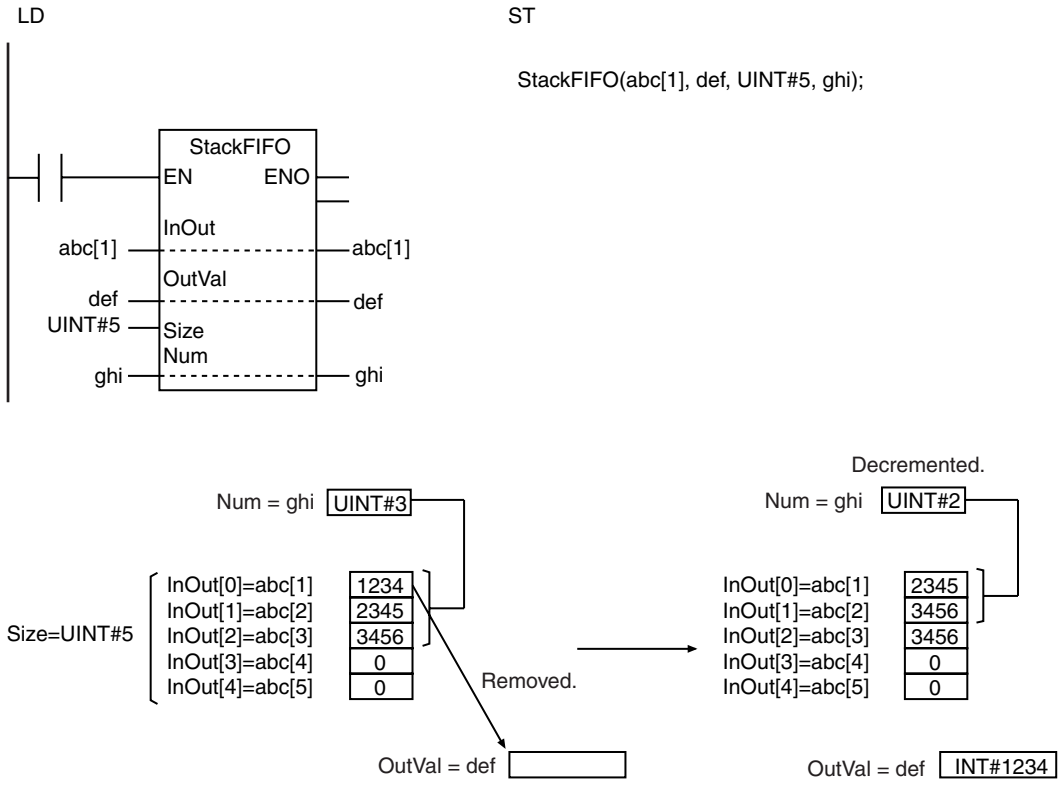
Function

The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. The instruction removes a value from the stack and assigns it to output value *OutVal.* For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

● StackFIFO

The StackFIFO removes the bottom value from a stack. Value of *InOut[0]* is assigned to *OutVal.* Then, all *Num* − 1 elements from *InOut[1]* are shifted to the next lower element in the stack array. Then 0 is stored in *InOut[Num*−1*]*. Finally, *Num* is decremented.

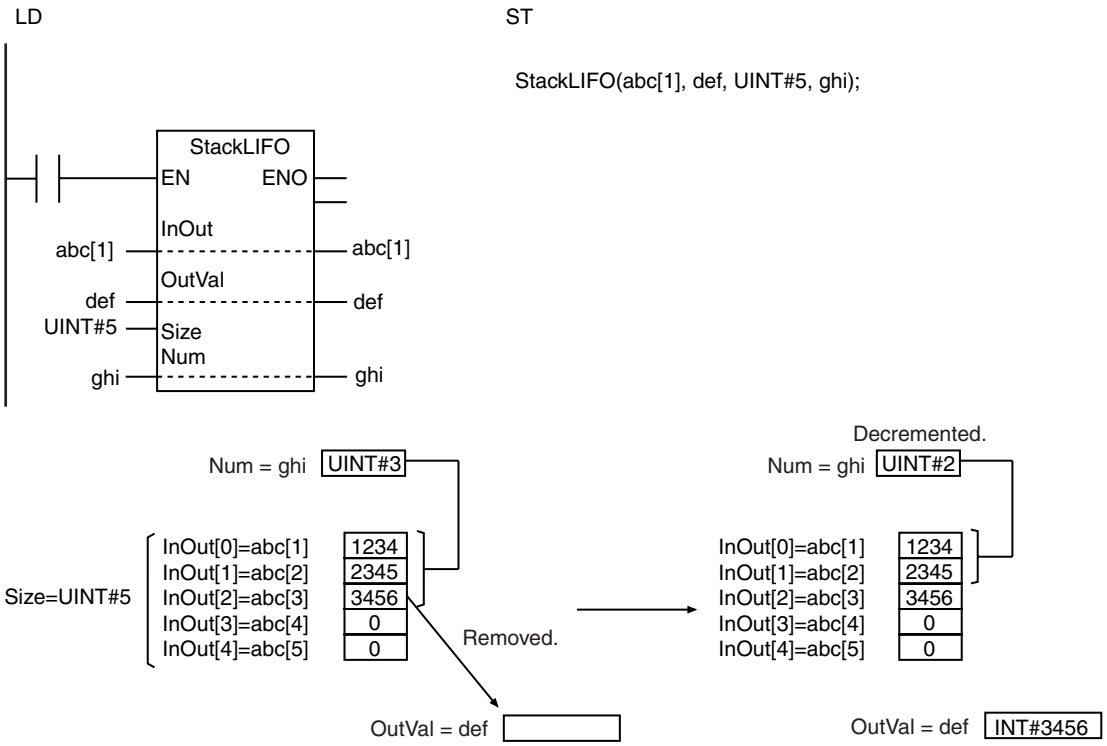
The following example is for when *Size* is UINT#5 and *Num* is UINT#3.



● StackLIFO

The StackLIFO instruction removes the top value from a stack. Value of *InOut[Num−1]* is assigned to *OutVal.* Then, *Num* is decremented.

The following example is for when *Size* is UINT#5 and *Num* is UINT#2.



Precautions for Correct Use

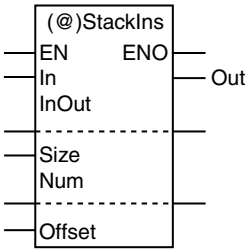
- Use the same data type for *InOut[]* and *OutVal.*
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The values in *InOut[]*, *Num*, and *OutVal* do not change if the value of *Size* or *Num* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *OutVal* will not change.
  - *InOut[]* and *OutVal* have different data types.
  - The values of *Num* and *Size* are not 0 and *Num* is greater than *Size*.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *InOut[]* is a STRING array and any of the elements does not end in a NULL character.
  - *InOut[]* is a STRING array and the number of bytes in the elements exceeds the size of *OutVal*.

Sample Programming

Refer to the sample programming that is provided for the StackPush instruction (page 2-466).

StackIns

The StackIns instruction inserts a value at a specified position in a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackIns	Insert into Stack	FUN		StackIns(In, InOut, Size, Num, Offset);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Insert value	Input	Value, structure, or structure member to insert into the stack	Depends on data type.	---	*
Size	Number of stack elements		Number of stack array elements			1
Offset	Offset		Position in stack at which to insert *In*			0
InOut[] (array)	Stack array	In-out	Array that functions as stack	Depends on data type.	---	---
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

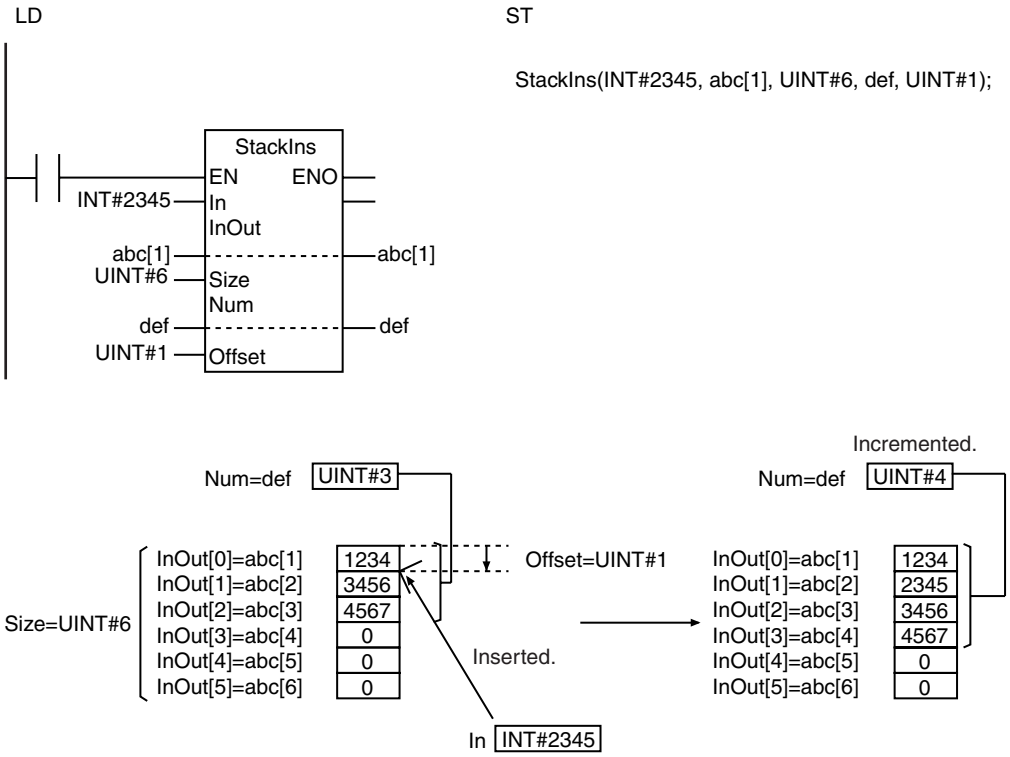
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
Size							OK													
Offset							OK													
InOut[] (array)	Must be an array with elements that have the same data type as *In*.																			
Num							OK													
Out	OK																			

Function

The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. Insert value *In* is inserted at the position specified by the offset *Offset* (*InOut[Offset]*). All higher elements, i.e., *InOut[Offset]* to *InOut[Num–1]*, are moved to the next higher element in the stack array. Then, *Num* is incremented. For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

The following example is for when *Size* is UINT#6, *Num* is UINT#3 and *Offset* is UINT#1.



Precautions for Correct Use

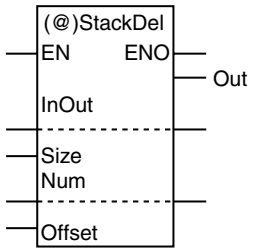
- Use the same data type for *In* and *InOut[].*
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The values in *InOut[]* and *Num* do not change if the value of *Size* is 0.
- When *In* is an enumeration, always use a variable for the input parameter to pass to *In*. A building error will occur if a constant is passed.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - *In* and *InOut[]* contain different data types.
  - The value of *Size* is not 0 and *Size* is not greater than *Num* and *Num* is not greater than or equal to *Offset*.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *In* is STRING data and it does not end in a NULL character.
  - *InOut[]* is a STRING array and the number of bytes in the elements exceeds the size of *OutVal*.

Sample Programming

Refer to the sample programming that is provided for the StackPush instruction (page 2-466).

StackDel

The StackDel instruction deletes a value from a specified position in a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackDel	Delete from Stack	FUN		StackDel(InOut, Size, Num, Offset);

Variables

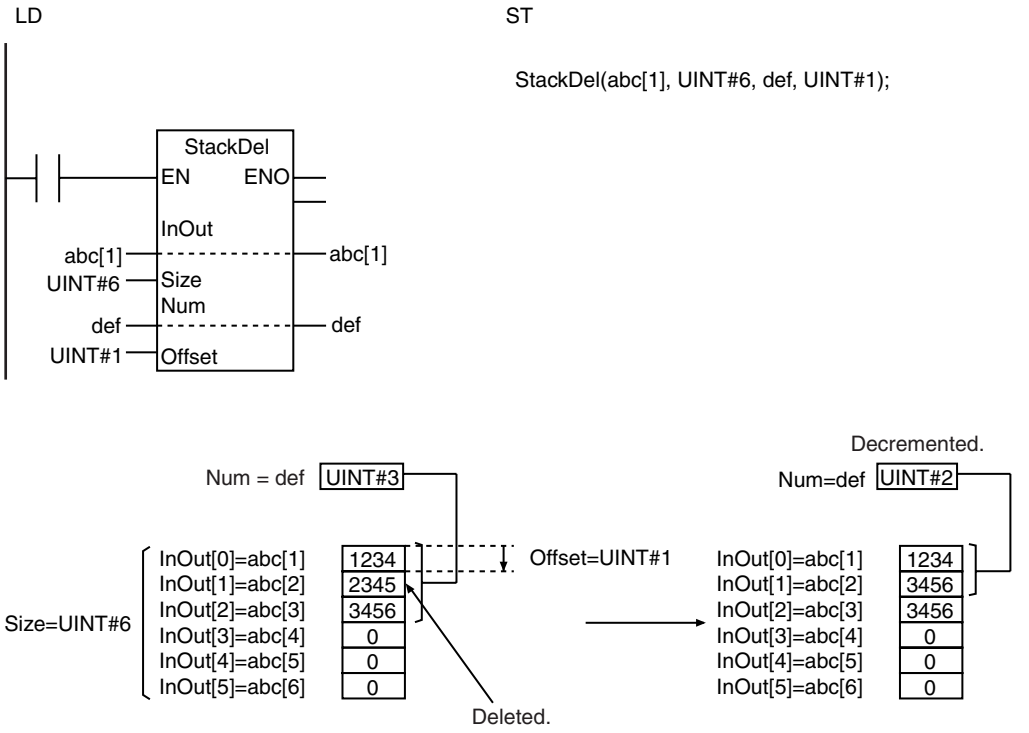
Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of stack elements	Input	Number of stack array elements	Depends on data type.	---	1
Offset	Offset		Offset of value to delete from stack			0
InOut[] (array)	Stack array	In-out	Array that functions as stack	Depends on data type.	---	---
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
Offset							OK													
InOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Num							OK													
Out	OK																			

Function

The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. The value is deleted from the position specified by the offset *Offset* (*InOut[Offset]*). All higher elements, i.e., *InOut[Offset*+1*]* to *InOut[Num*–1*]*, are moved to the next lower element in the stack array. Then, *Num* is decremented. For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

The following example is for when *Size* is UINT#6, *Num* is UINT#3 and *Offset* is UINT#1.

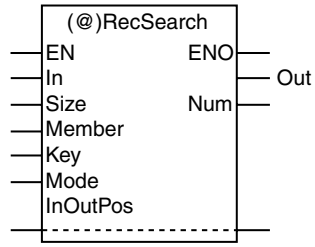


Precautions for Correct Use

- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The values in *InOut[]* and *Num* do not change if the value of *Size* or *Num* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - The values of *Num* and *Size* are not 0 and *Size* is not greater than or equal to *Num* and *Num* is not greater than *Offset*.
  - The value of *Size* exceeds the array area of *InOut[]*.

RecSearch

The RecSearch instruction searches an array of structures for elements that match the search key with the specified method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecSearch	Record Search	FUN		Out:=RecSearch(In, Size, Member, Key, Mode, InOut-Pos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array of structures to search	---	---	*
Size	Number of elements to search		Number of array elements to search	Depends on data type.		1
Member	Member to search		Member of In[] structure to search			*
Key	Search key		Search value			
Mode	Search method		Search method	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[] (array)	Element numbers of matching elements	In-out	Element numbers of matching elements	Depends on data type.	---	---
Out	Search result	Output	TRUE: There are elements that match conditions FALSE: There are no elements that match conditions	Depends on data type.	---	---
Num	Number of matches		Number of matches			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Size							OK													
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
	Specify the same data type as the search member of *In[]*																			
Key	Must be the same data type as *Member*.																			
Mode	Refer to *Function* for the enumerators of the enumerated type _eSEARCH_MODE.																			
InOutPos[] (array)							OK													
Out	OK																			
Num							OK													

Function

The RecSearch instruction searches *Size* elements in the array of structures *In[]*. The search range is therefore from *In[0]* to *In[Size−1]*. The instruction searches member to search Member in the structures for members that match the search key *Key*.
One of the members to search in the elements of *In[]* is passed as an argument to *Member*.
If any matching elements are found, the value of search result *Out* changes to TRUE. The element number of the matching element is assigned to *InOutPos[0]* and the number of matching elements is assigned to *Num*. If there is more than one matching element, the element number of the lowest matching element in *In[]* is assigned to *InOutPos[0]*. If there are no matching elements, the value of *Out* will be FALSE and *InOutPos[0]* and *Num* will be 0.

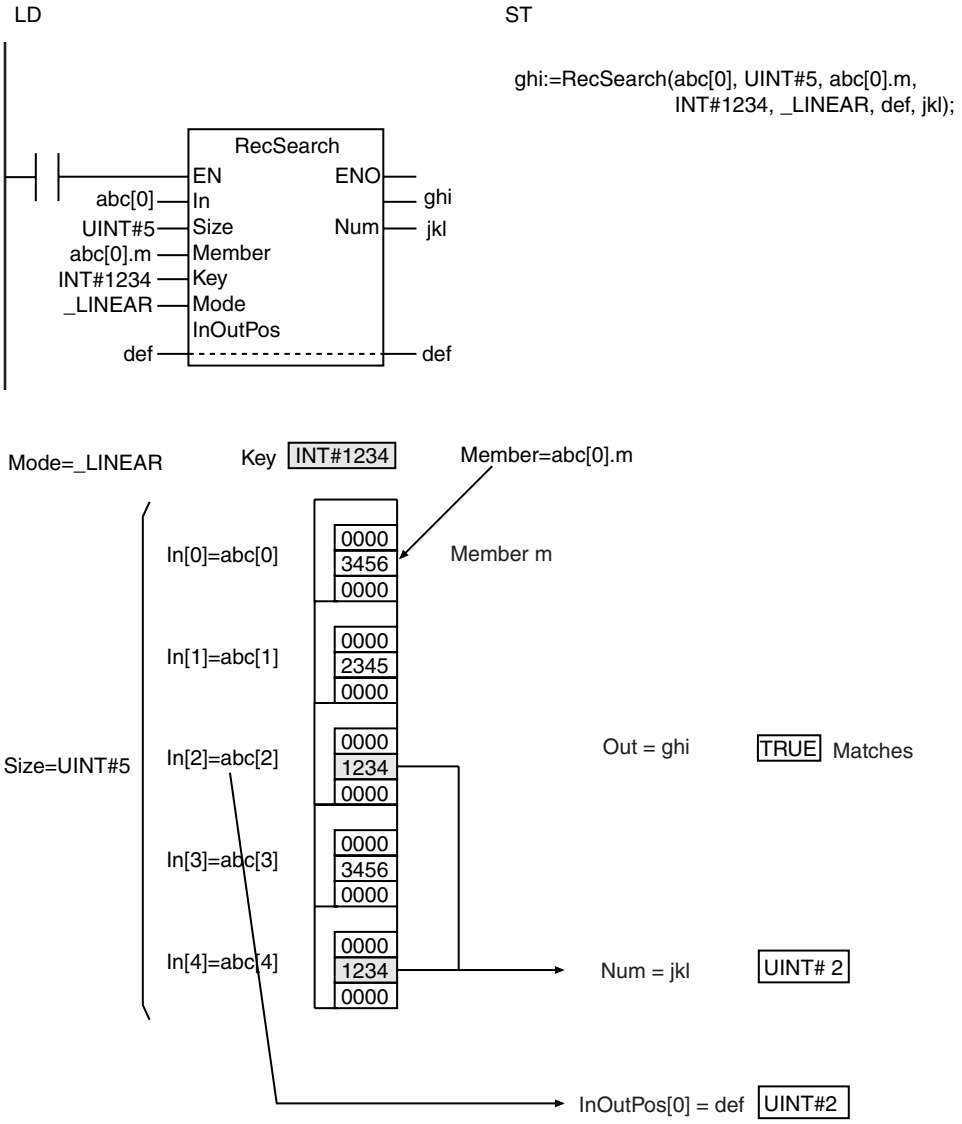
Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

The data type of search method *Mode* is enumerated type _eSEARCH_MODE. The meanings of the enumerators are as follows:

Enumerator	Meaning
_LINEAR	Linear search
_BIN_ASC	Ascending binary search
_BIN_DESC	Descending binary search

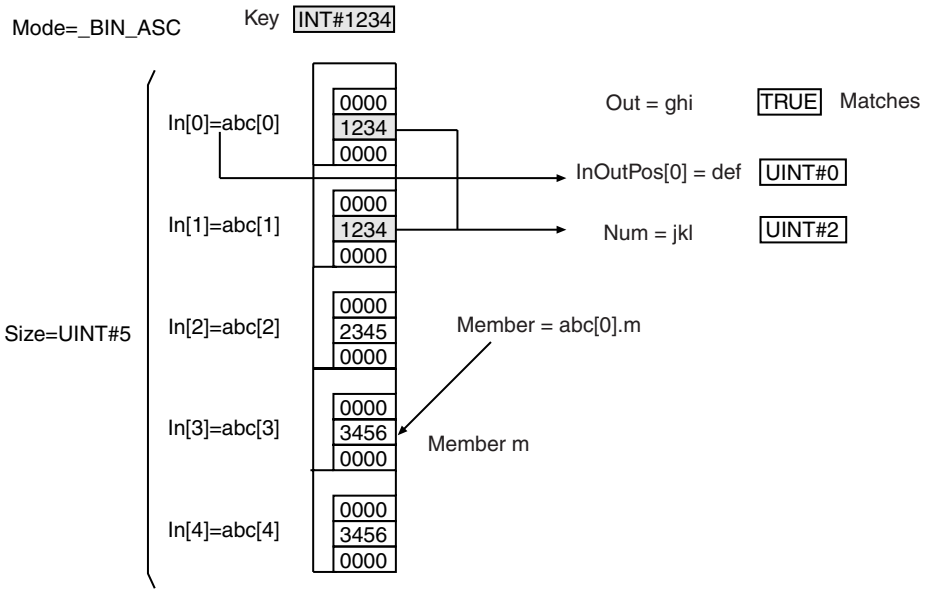
For a linear search, the search is performed in order from the first element of *In[].*

The following example is for when *Size* is UINT#5, *Key* is INT#1234 and *Mode* is _LINEAR.



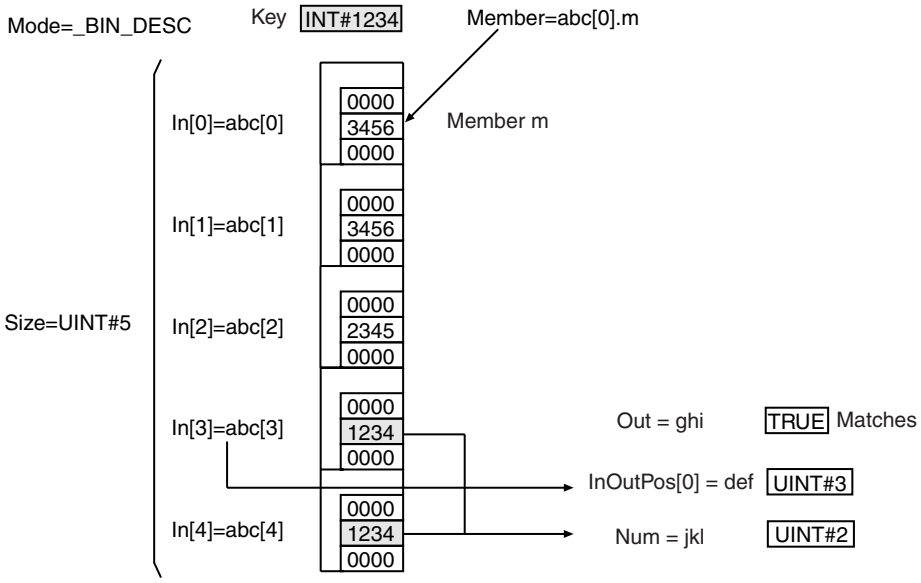
For an ascending binary search, the array elements in the input parameter that is passed to *In[]* must be in ascending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for an ascending binary search.



For a descending binary search, the array elements in the input parameter that is passed to *In[]* must be in descending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for a descending binary search.



Additional Information

- *In[]* can be a member of a higher-level structure.

  Example: In[0]=str0.str1[0]

- *In[]* can be an array with two or more dimensions. If *In[]* is a two-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]* and the element number in the second dimension is assigned to *InOutPos[1]*.

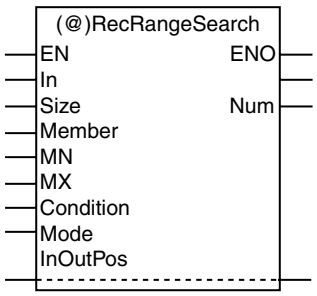
- If *In[]* is a three-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]*, the element number in the second dimension is assigned to *InOutPos[1]*, and the element number in the third dimension is assigned to *InOutPos[2]*.

Precautions for Correct Use

- When an element in the array is passed to *In[]*, all elements below the passed element are processed.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- If *Key* is a real number, do not specify nonnumeric data for *Key*.
- If the value of *Size* is 0, the value of *Out* is FALSE and the value of *Num* is 0. *InOutPos[]* does not change.
- The correct result is not obtained if the value of *Mode* is _BIN_ASC or _BIN_DESC and the elements of *In[]* are not in ascending or descending order. Place the elements in ascending or descending order before executing this instruction.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *InOutPos[]*, and *Num* will not change.
  - The value of *Mode* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - *Member* is not a member of *In[]*.
  - The array size of *InOutPos[]* is smaller than the number of dimensions of *In[]*.
  - *Member* is not integer or real number data.
  - *Key* and *Member* have different data types.
  - *In[]* is not an array of structures.

RecRangeSearch

The RecRangeSearch instruction searches an array of structures for elements that match the search condition range with the specified method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecRangeSearch	Range Record Search	FUN		Out:=RecRangeSearch(In, Size, Member, MN, MX, Condition, Mode, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array of structures to search	---	---	*
Size	Number of elements to search		Number of array elements to search	Depends on data type.		1
Member	Member to search		Member of I*n[]* structure to search			*
MN	Search condition lower limit		Search condition lower limit			
MX	Search condition upper limit		Search condition upper limit			
Condition	Search condition		Search condition	_EQ_BOTH, _EQ_MIN, _EQ_MAX, _NE_BOTH		_EQ_BOTH
Mode	Search method		Search method	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[] (array)	Element numbers of matching elements	In-out	Element numbers of matching elements	Depends on data type.	---	---
Out	Search result	Output	TRUE: There are elements that match conditions FALSE: There are no elements that match conditions	Depends on data type.	---	---
Num	Number of matches		Number of matches			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Size							OK													
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	Specify the same data type as the search member of *In[]*.																			
MN	Must be the same data type as *Member*.																			
MX	Must be the same data type as *Member*.																			
Condition	Refer to *Function* for the enumerators for the enumerated type _eSEARCH_CONDITION.																			
Mode	Refer to *Function* for the enumerators for the enumerated type _eSEARCH_MODE.																			
InOutPos[] (array)							OK													
Out	OK																			
Num							OK													

Function

The RecRangeSearch instruction searches Size elements in the array of structures *In[]*. The search range is therefore from *In[0]* to *In[Size−1]*. The instruction searches member to search *Member* in the structures for members that match the search condition.

Condition specifies the search condition. *Mode* specifies the search method. Details are provided below. One of the members to search in the elements of *In[]* is passed as an argument to *Member*.

If any elements that match the search condition are found, the value of search result *Out* changes to TRUE. The element number of the matching element is assigned to *InOutPos[0]* and the number of matching elements is assigned to *Num*. If there is more than one matching element, the element number of the lowest matching element in *In[]* is assigned to *InOutPos[0]*. If there are no matching elements, the value of *Out* will be FALSE and *InOutPos[0]* and *Num* will be 0.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

The data type of search condition *Condition* is enumerated type _eSEARCH_CONDITION. The meanings of the enumerators are as follows:

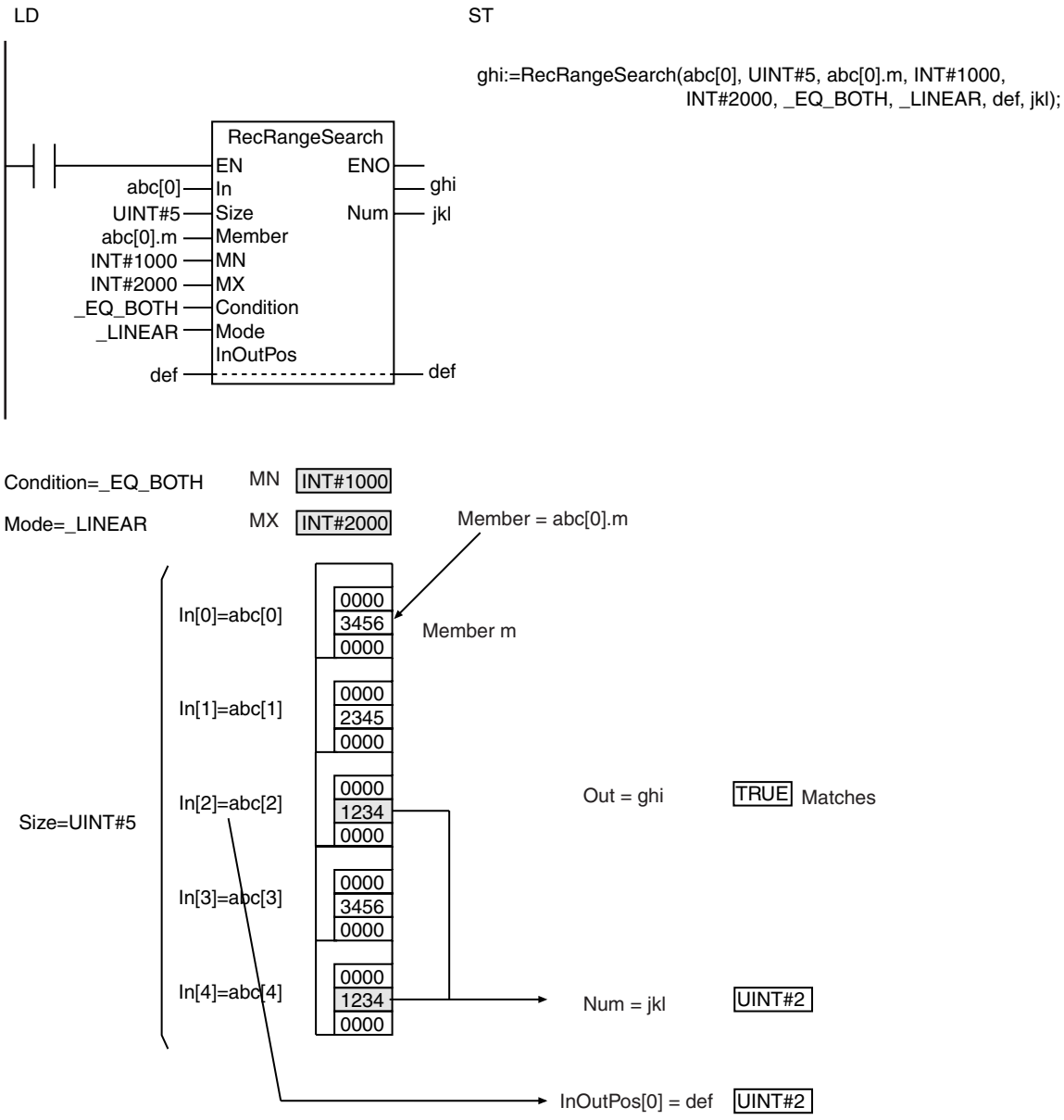
Enumerator	Meaning
_EQ_BOTH	MN ≤ Member ≤ MX
_EQ_MIN	MN ≤ Member < MX
_EQ_MAX	MN < Member ≤ MX
_NE_BOTH	MN < Member < MX

The data type of search method *Mode* is enumerated type _eSEARCH_MODE. The meaning of the enumerators are as follows:

Enumerator	Meaning
_LINEAR	Linear search
_BIN_ASC	Ascending binary search
_BIN_DESC	Descending binary search

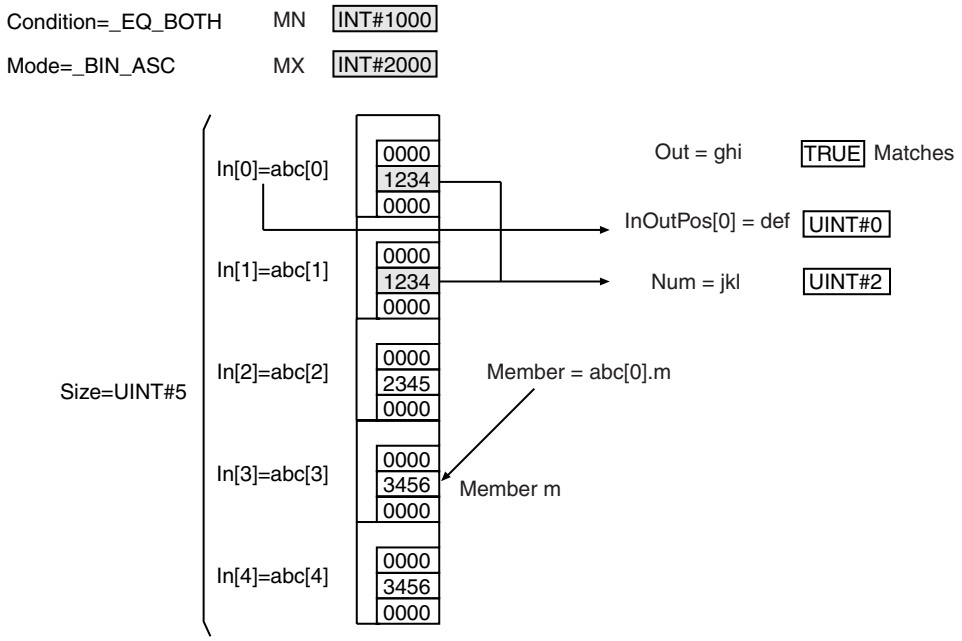
For a linear search, the search is performed in order from the first element of *In[]*.

The following example is for when *Size* is UINT#5, *MN* is INT#1000, *MX* is INT#2000, *Condition* is _EQ_BOTH and *Mode* is _LINEAR.



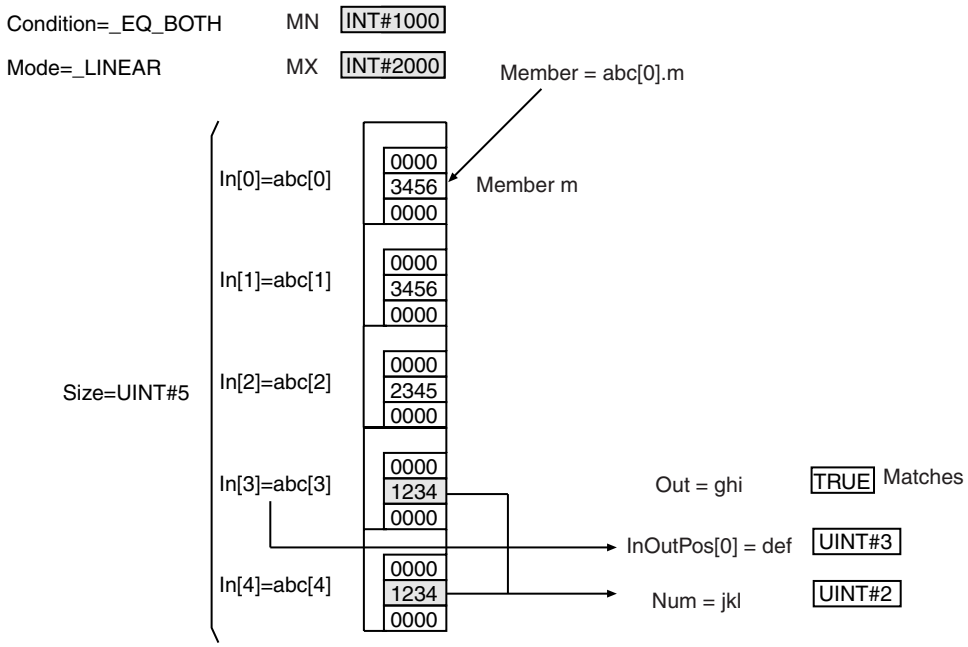
For an ascending binary search, the array elements in the input parameter that is passed to *In[]* must be in ascending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for an ascending binary search.



For a descending binary search, the array elements in the input parameter that is passed to *In[]* must be in descending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for a descending binary search.



Additional Information

- *In[]* can be a member of a higher-level structure.

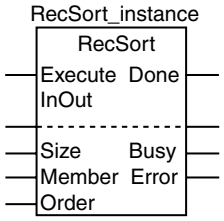
  Example: In[0]=str0.str1[0]
- *In[]* can be an array with two or more dimensions. If *In[]* is a two-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]* and the element number in the second dimension is assigned to *InOutPos[1]*.
- If *In[]* is a three-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]*, the element number in the second dimension is assigned to *InOutPos[1]*, and the element number in the third dimension is assigned to *InOutPos[2]*.

Precautions for Correct Use

- Make the data types of *Member*, *MN*, and *MX* the same as the data type of the members that are searched in *In[]*.
- When an element in the array is passed to *In[]*, all elements below the passed element are processed.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- If *MN* or *MX* is a real number, do not specify nonnumeric data for *MN* or *MX*.
- If the value of *Size* is 0, the value of *Out* is FALSE and the value of *Num* is 0. *InOutPos[]* does not change.
- The correct result is not obtained if the value of *Mode* is _BIN_ASC or _BIN_DESC and the elements of *In[]* are not in ascending or descending order. Place the elements in ascending or descending order before executing this instruction.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *InOutPos[]*, and *Num* will not change.
  - The data types of the member to search in *In[]*, *MN*, and *MX* are different.
  - *MN* is greater than *MX*.
  - The value of *Condition* is outside of the valid range.
  - The value of *Mode* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - *Member* is not a member of *In[]*.
  - The array size of *InOutPos[]* is smaller than the number of dimensions of *In[]*.
  - *Member* is not integer or real number data.
  - *MN*, *MX*, and *Member* have different data types.
  - *In[]* is not an array of structures.

RecSort

The RecSort instruction sorts the elements of an array of structures.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecSort	Record Sort	FB		RecSort_instance(Execute, InOut, Size, Member, Order, Done, Busy, Error);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of elements to sort	Input	Number of array elements to sort	Depends on data type.	---	1
Member	Member to sort		Member of I*n[]* structure to sort			*
Order	Sort order		Sort order	_ASC, _DESC		_ASC
InOut[] (array)	Sort array	In-out	Array of structures to sort	---	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	Specify the same data type as the sort member of *InOut[]*																			
Order	Refer to *Function* for the enumerators of the enumerated type _eSORT_ORDER.																			
InOut[] (array)	Specify an array of structures.																			

Function

When the value of *Execute* is TRUE, *Size* elements of *InOut[]* (a structure array) is sorted. Specifically, the elements from *InOut[0]* to *InOut[Size−1]* are sorted. Specifically, the elements from *InOut[0]* to *InOut[Size−1]* are sorted. *Order* specifies the sort order. Details are provided below. One of the members to sort in the elements of *In[]* is passed as an argument to *Member*.

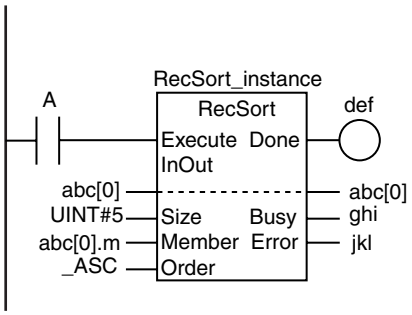
Always attach the element number to the in-out parameter that is passed to *InOut[]*, e.g., *array[3]*.

The data type of sort order *Order* is enumerated type _eSORT_ORDER. The meaning of the enumerators are as follows:

Enumerator	Meaning
_ASC	Ascending
_DESC	Descending

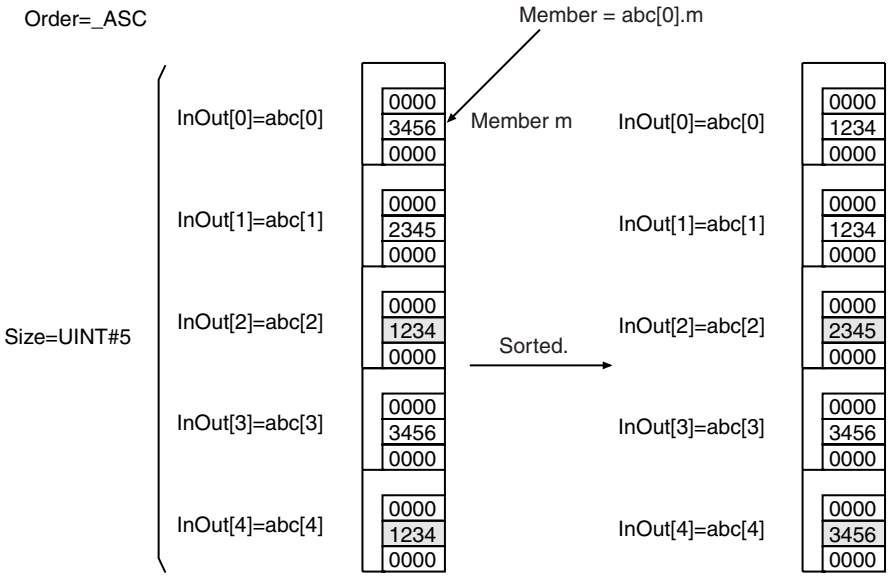
The following example is for when *Size* is UINT#5, *Member* is 3456 and *Order* is *_Asc*.

LD



ST

```
RecSort_instance(A, abc[0], UINT#5, abc[0].m, _ASC, def, ghi, jkl);
```



Additional Information

If the power supply is interrupted during execution of this instruction, the contents of *InOut* may be corrupted. If you back up the contents of *InOut[]* each time the instruction is completed normally, you can restore the data if it is corrupted. Refer to *Sample Programming*.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to *FALSE* or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- If the value of *Size* is 0, the value of *Done* will be TRUE and *InOut[]* will not change.
- An error occurs in the following cases. *Done* and *Busy* will be FALSE and *Error* will be TRUE.
  - The value of *Order* is outside of the valid range.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *Member* is not a member of *InOut[]*.
  - *Member* is not integer or real number data.
  - *InOut[]* is not an array of structures.

Sample Programming

In this sample, the RecSort instruction is used to sort an array *Abc[]* of *MyStr* structures in ascending order. The member to sort is *Abc[].m*. To prevent loosing data even if power is interrupted during processing, *Abc[]* is backed up in a variable named *Abc_backup[]* before sorting. If a power interruption occurs, the contents of *Abc_backup[]* is restored to *Abc[]* and the sort operation is redone.

Definitions of Global Variables

Data Types

Variable	Data type	Comment
MyStr	STRUCT	Structure
l	BOOL	Member
m	INT	Member
n	REAL	Member

Global Variables

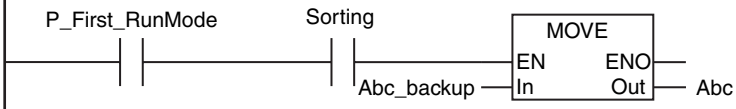
Variable	Data type	Initial value	Retain	Comment
Abc	ARRAY[0..4] OF MyStr	[5(l:=False, m:=0, n:=0.0)]	☑	Sort array
Abc_backup	ARRAY[0..4] OF MyStr	[5(l:=False, m:=0, n:=0.0)]	☑	Backup of *Abc[]*

LD

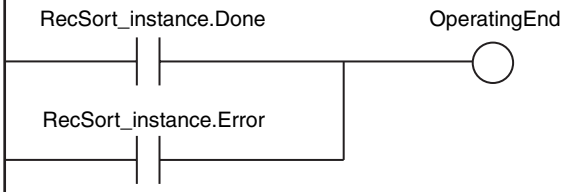
Internal Variables	Variable	Data type	Initial value	Retain	Comment
	Sorting	BOOL	False	☑	Processing (retained)
	OperatingEnd	BOOL	False	☐	Processing completed
	Trigger	BOOL	False	☐	Execution condition
	Operating	BOOL	False	☐	Processing
	RS_instance	RS		☐	
	RecSort_instance	RecSort		☐	

External Variables	Variable	Data type	Comment
	Abc	ARRAY[0..4] OF MyStr	Sort array
	Abc_backup	ARRAY[0..4] OF MyStr	Backup of *Abc[]*

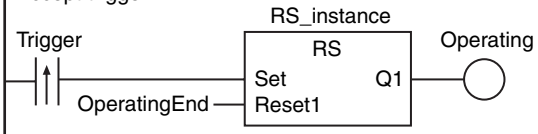
Restore *Abc_backup[]* to *Abc[]* after power interruption.



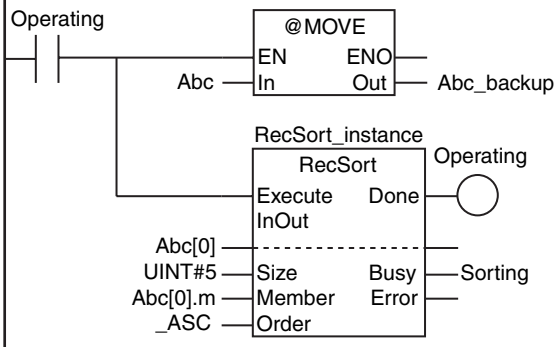
Determine if execution of the RecSort instruction is completed.



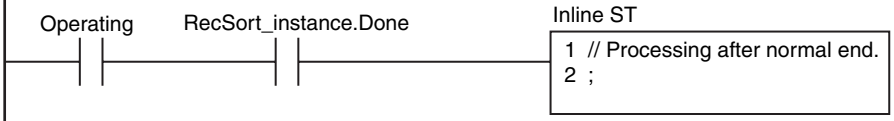
Accept trigger.



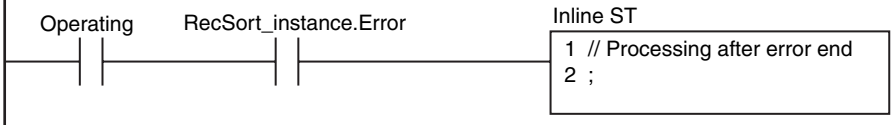
Make backup and execute RecSort instruction.



Processing after normal end.



Processing after error end



ST

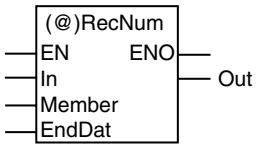
Internal Variables	Variable	Data type	Initial value	Retain	Comment
	Sorting	BOOL	False	☑	Processing (retained)
	Trigger	BOOL	False	☐	Execution condition
	LastTrigger	BOOL	False	☐	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	☐	Processing started
	Operating	BOOL	False	☐	Processing
	RS_instance	RS		☐	
	RecSort_instance	RecSort		☐	

External Variables	Variable	Data type	Comment
	Abc	ARRAY[0..4] OF MyStr	Sort array
	Abc_backup	ARRAY[0..4] OF MyStr	Backup of *Abc[]*

```
// Restore Abc_backup[] to Abc[] after power interruption.
IF ( (P_First_RunMode = TRUE) AND (Sorting = TRUE) ) THEN
    Abc:=Abc_backup;
END_IF;

// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize RecSort instruction.
IF (OperatingStart=TRUE) THEN
    Abc_backup:=Abc;
    RecSort_instance(
        Execute:=FALSE,          // Start condition
        InOut   :=Abc[0],        // Sort array
        Member:=Abc[0].m);       // Member to sort
    OperatingStart:=FALSE;
END_IF;

// Execute RecSort instruction.
IF (Operating=TRUE) THEN
    RecSort_instance(
        Execute:=TRUE,
        InOut   :=Abc[0],
        Size    :=UINT#5,
        Member:=Abc[0].m,
        Order   :=_ASC,
        Busy    :=>Sorting);

    IF (RecSort_instance.Done=TRUE) THEN
        // Processing after normal end.
        Operating:=FALSE;
    END_IF;

    IF (RecSort_instance.Error=TRUE) THEN
        // Processing after error end.
        Operating:=FALSE;
    END_IF;
END_IF;
```

RecNum

The RecNum instruction finds the number of records in an array of structures to the end data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecNum	Get Number of Records	FUN		Out:=RecNum(In, Member, EndDat);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array of structures to process	---	---	*
Member	Member to process		Member of I*n[]* structure to process	Depends on data type.		
EndDat	End data		End data			
Out	Number of records	Output	Number of records	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Member	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
	Must be the same data type as the members to process in *In[]*.																			
EndDat	Must be the same data type as *Member*.																			
Out							OK													

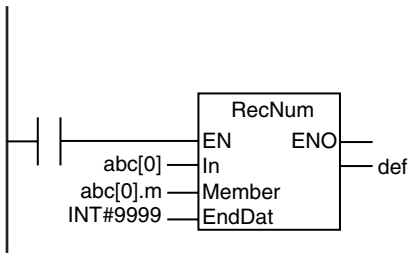
Function

The RecNum instruction searches from the start of an array *In[]* (whose elements are structures). The instruction searches for elements for which the value of member to process *Member* matches end data *EndDat*. As the result, it assigns the number of elements (records) up to the element just before the element with an *EndDat* match to *Out*. One of the members to process in the elements of *In[]* is passed as an argument to *Member*.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

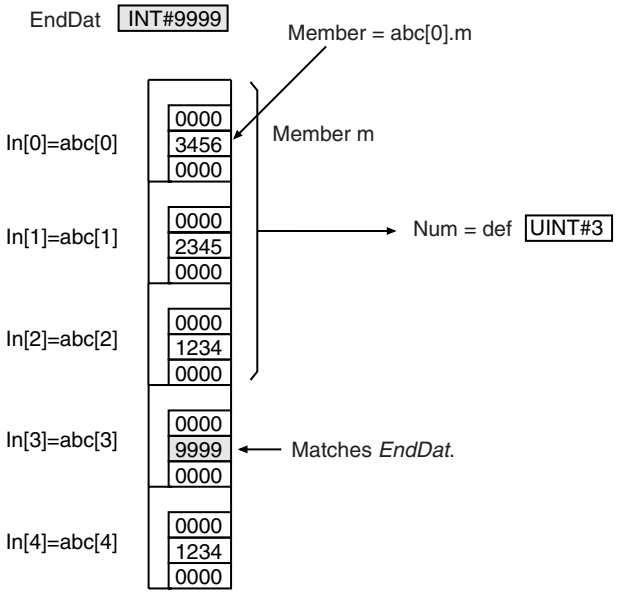
The following example is for when *EndDat* is INT#9999.

LD



ST

```
def:=RecNum(abc[0], abc[0].m, INT#9999);
```



Additional Information

In[] can be a member of a higher-level structure.

Example: In[0]=str0.str1[0]

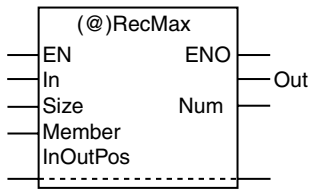
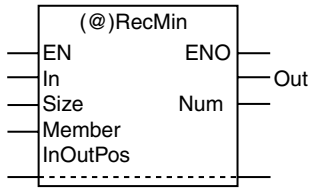
Precautions for Correct Use

- If there are no members in *In[]* that match *EndDat*, the total number of elements in *In[]* is assigned to *Out*.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- If *EndDat* is a real number, do not specify nonnumeric data for it.
- When an element in the array is passed to *In[]*, all elements below the passed element are processed.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *Member* is not a member of *In[]*.
  - *Member* or *EndDat* is STRING data and it does not end in a NULL character.
  - *Member* is not Boolean, integer, bit string, real number, or text string data.
  - *Member* and *EndDat* have different data types.
  - *In[]* is not an array of structures.

RecMax and RecMin

RecMax: Searches the specified member in the structures of an array of structures for the maximum value.

RecMin: Searches the specified member in the structures of an array of structures for the minimum value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecMax	Maximum Record Search	FUN		Out:=RecMax(In, Size, Member, InOutPos, Num);
RecMin	Minimum Record Search	FUN		Out:=RecMin(In, Size, Member, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array of structures to search	---	---	*
Size	Number of elements to search		Number of array elements to search	Depends on data type.		1
Member	Member to search		Member of I*n[]* structure to search			*
InOutPos[] (array)	Found element number	In-out	Found element number	Depends on data type.	---	---
Out	Search result	Output	Search result	Depends on data type.	---	---
Num	Number found		Number found			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Size							OK													
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	Specify the same data type as the search member of *In[].*																			

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOutPos[] (array)							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Num							OK													

Function

These instructions search *Size* elements in an array of structures *In[]*. The search range is therefore from *In[0]* to *In[Size−1]*. The instruction searches member to search Member in the structures.

One of the members to search in the elements of *In[]* is passed as an argument to *Member*. The element number of the element with the maximum or minimum value is assigned to *InOutPos[0]* and the number of elements that were found is assigned to *Num*. If more than one element was found, the element number of the lowest element with the maximum or minimum value in *In[]* is assigned to *InOutPos[0]*.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

● RecMax

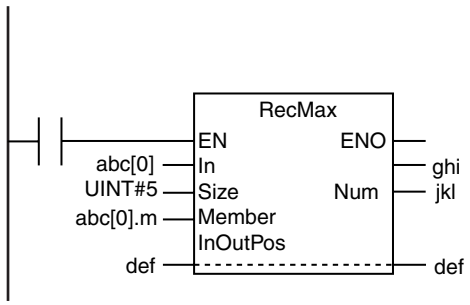
The RecMax instruction searches for the maximum value. The maximum value of the member to search is assigned to search result *Out*.

● RecMin

The RecMin instruction searches for the minimum value. The minimum value of the member to search is assigned to search result *Out*.

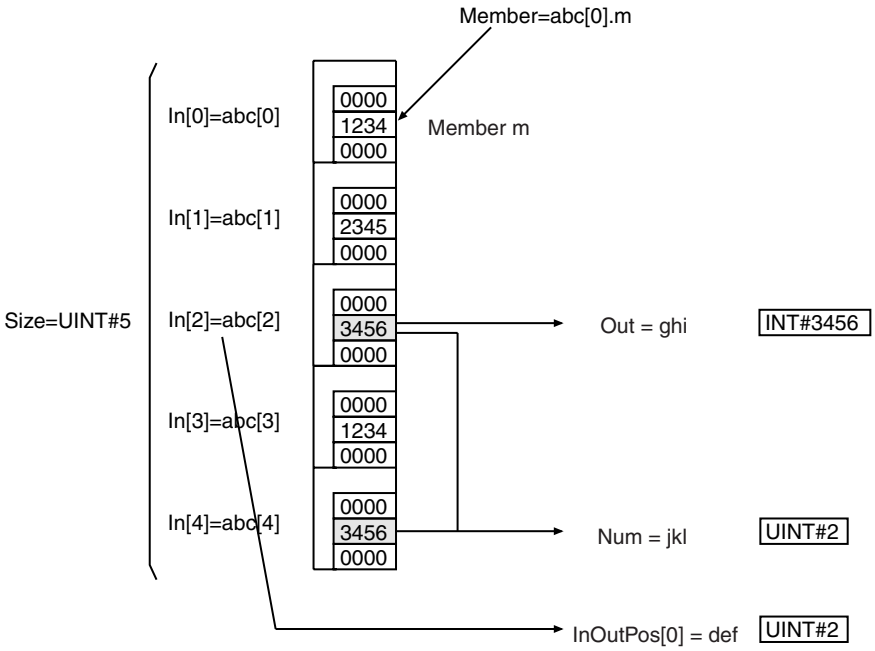
The following example shows the RecMax instruction when *Size* is UINT#5.

LD



ST

```
ghi:=RecMax(abc[0], UINT#5, abc[0].m, def, jkl);
```



Additional Information

- *In[]* can be a member of a higher-level structure.

  Example: In[0]=str0.str1[0]
- *In[]* can be an array with two or more dimensions. If *In[]* is a two-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]* and the element number in the second dimension is assigned to *InOutPos[1]*.
- If *In[]* is a three-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]*, the element number in the second dimension is assigned to *InOutPos[1]*, and the element number in the third dimension is assigned to *InOutPos[2]*.

Precautions for Correct Use

- If you use a different data type for *Member* and *Out*, use only the following data types and make sure the valid range of *Out* includes the valid range of *Member*.
  - USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, and LREAL
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- When an element in the array is passed to *In[]*, all elements below the passed element are processed.
- If the value of *Size* is 0, the values of *Out* and Num are 0. The values in *InOutPos[]* do not change.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *InOutPos[]*, and *Num* will not change.
  - The value of *Size* exceeds the array area of *In[]*.
  - *Member* is not a member of *In[]*.
  - The array size of *InOutPos[]* is smaller than the number of dimensions of *In[]*.
  - An array without a subscript is passed to *In[]*.
  - *Member* is not integer or real number data.

FCS Instructions

Instruction	Name	Page
StringSum	Checksum Calculation	2-504
StringLRC	Calculate Text String LRC	2-506
StringCRCCCITT	Calculate Text String CRC-CCITT	2-508
StringCRC16	Calculate Text String CRC-16	2-510
AryLRC_**	Calculate Array LRC Group	2-512
AryCRCCCITT	Calculate Array CRC-CCITT	2-514
AryCRC16	Calculate Array CRC-16	2-516

StringSum

The StringSum instruction calculates the checksum for a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringSum	Checksum Calculation	FUN	(@)StringSum EN ENO In Out Size	Out:=StringSum(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	''
Size	Byte size		Byte size of checksum	1 or 2	Bytes	1
Out	Checksum	Output	Checksum	Number of bytes specified by *Size*	Bytes	---

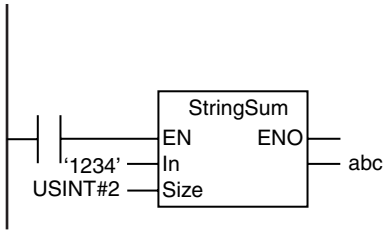
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Size						OK														
Out																				OK

Function

The StringSum instruction calculates the checksum of text string to process *In*. Checksum *Out* will be the number of bytes specified with byte size *Size*. *Out* is given as a hexadecimal text string with a NULL character stored at the end.

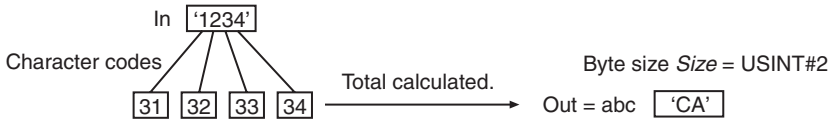
The following example is for when *In* is '1234' and *Size* is USINT#2.

LD



ST

```
abc:=StringSum('1234', USINT#2);
```



If *Size* was USINT#1 in the above example, *Out* would be 'A'.

Precautions for Correct Use

- If the sum of the character codes in *In* exceeds the number of digits of *Size*, the upper digits are discarded.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - *In* does not end in a NULL character.
  - The number of bytes in *In* is 0 (i.e., the NULL character only).
  - The size of the processing result exceeds the size of *Out*.

StringLRC

The StringLRC instruction calculates the LRC value (horizontal parity).

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringLRC	Calculate Text String LRC	FUN	(@)StringLRC EN ENO In Out	Out:=StringLRC(In);

Variables

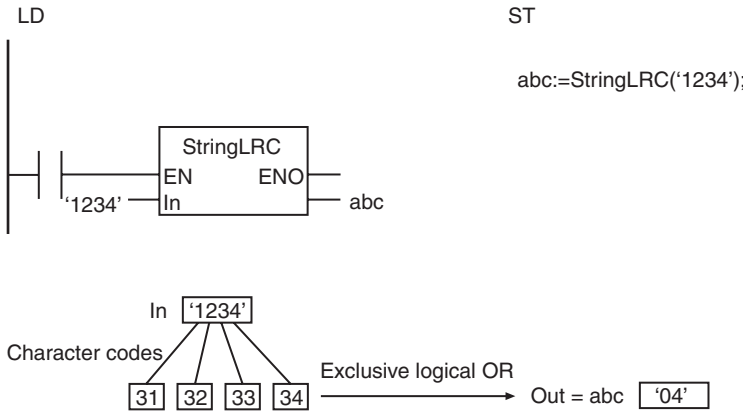
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	"
Out	LRC value	Output	LRC value	3 bytes max. (two single-byte alphanumeric characters plus the final NULL character)	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out																				OK

Function

The StringLRC instruction calculates the LRC value (horizontal parity) of text string to process *In*. The LRC value is the exclusive logical OR of the character codes for the text string in *In.* The LRC value (*Out*) is given as a hexadecimal text string with a NULL character stored at the end.

The following example is for when *In* is '1234'.



Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- *In* does not end in a NULL character.
- The number of bytes in *In* is 0 (i.e., the NULL character only).
- The number of bytes for *Out* is outside of the valid range.

StringCRCCCITT

The StringCRCCCITT instruction calculates the CRC-CCITT value using the XMODEM method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringCRCCCITT	Calculate Text String CRC-CCITT	FUN	(@)StringCRCCCITT EN ENO In Out Initial OutOrder	Out:=StringCRCCCITT(In, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	"
Initial	Initial value		Initial value of CRC-CCITT value			0
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_HIGH _LOW
Out	CRC-CCITT value	Output	CRC-CCITT value	5 bytes (four single-byte alphanumeric characters plus the final NULL character)	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators of the enumerated type _eBYTE_ORDER.																			
Out																				OK

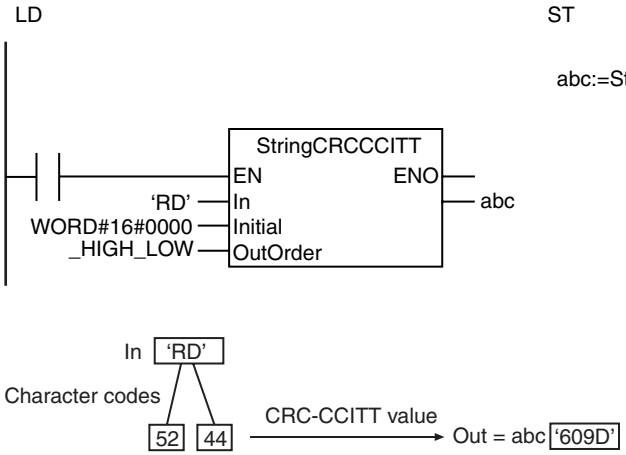
Function

The StringCRCCCITT instruction calculates the CRC-CCITT value of text string to process *In* using the XMODEM method. CRC-CCITT value *Out* is given as a hexadecimal text string with a NULL character stored at the end.

Set *Initial* to the initial value for CRC-CCITT value calculation. *OutOrder* specifies the byte order.
The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meanings of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

The following example is for when *In* is 'RD', *Initial* is WORD#16#0000, and *OutOrder* is _HIGH_LOW.



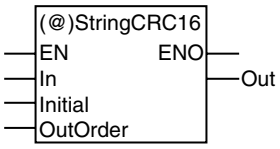
Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *OutOrder* is outside of the valid range.
- *In* does not end in a NULL character.
- The number of bytes in *In* is 0 (i.e., the NULL character only).
- The number of bytes for *Out* is outside of the valid range.

StringCRC16

The StringCRC16 instruction calculates the CRC-16 value using the MODBUS method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringCRC16	Calculate Text String CRC-16	FUN		Out:=StringCRC16(In, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	"
Initial	Initial value		Initial value of CRC-16 value			16#FFFF
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_LOW _HIGH
Out	CRC-16 value	Output	CRC-16 value	5 bytes (four single-byte alphanumeric characters plus the final NULL character)	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators of the enumerated type _eBYTE_ORDER.																			
Out																				OK

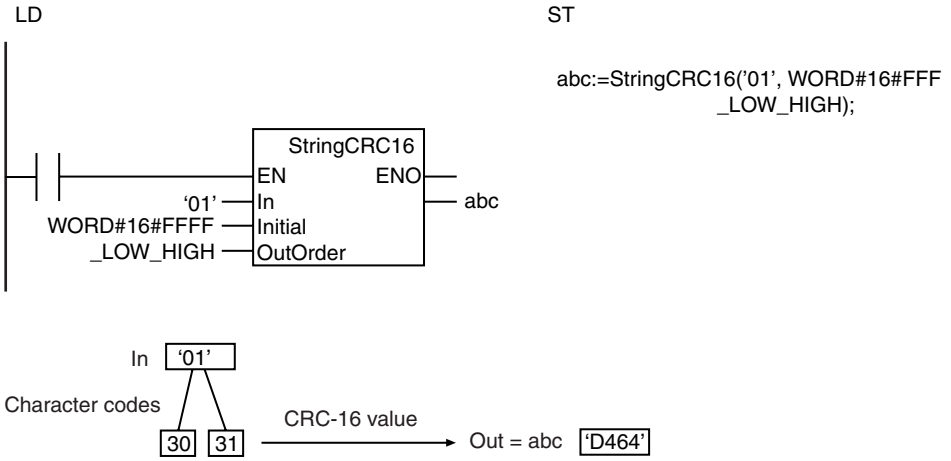
Function

The StringCRC16 instruction calculates the CRC-16 value of text string to process *In* using the MODBUS method. CRC-16 value *Out* is given as a hexadecimal text string with a NULL character stored at the end.

Set *Initial* to the initial value for CRC-16 value calculation. *OutOrder* specifies the byte order.
The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meanings of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

The following example is for when *In* is '01', *Initial* is WORD#16#FFFF and *OutOrder* is _LOW_HIGH.



Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *OutOrder* is outside of the valid range.
- *In* does not end in a NULL character.
- The number of bytes in *In* is 0 (i.e., the NULL character only).
- The number of bytes for *Out* is outside of the valid range.

AryLRC_**

The AryLRC_** instructions calculate the LRC value for an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryLRC_**	Calculate Array LRC Group	FUN	(@)AryLRC_** EN ENO In Out Size "**" must be a bit string data type.	Out:=AryLRC_**(In, Size); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Out	LRC value	Output	LRC value	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

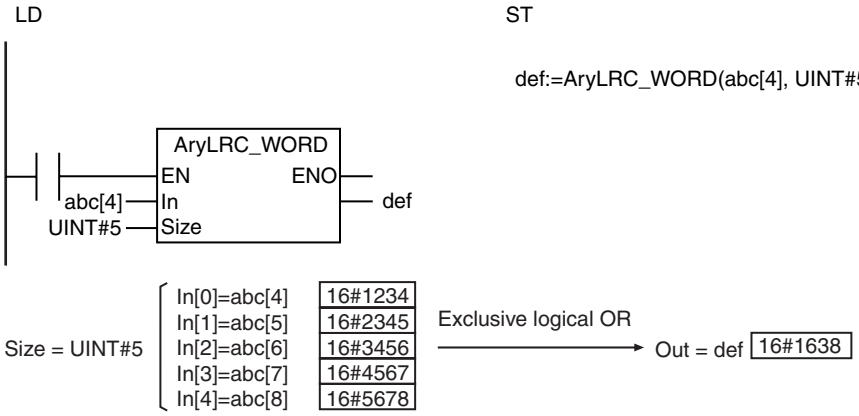
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK															
Size							OK													
Out	Must be same data type as *In[]*																			

Function

The AryLRC_** instructions calculate the LRC value (exclusive logical OR) of *Size* array elements of array to process *In[]* starting from *In[0]*. The name of the instruction is determined by the data type of *In[]*. For example, if *In[]* is the WORD data type, the instruction is AryLRC_WORD.

Always attach the element number to in-out parameter that is passed to *In[]*, e.g., *array[3]*.

The following example shows the AryLRC_WORD instruction when *Size* is UINT#5.



Precautions for Correct Use

- Use the same data type for *In[]* and *Out.*
- If the value of *Size* is 0, the value of *Out* is 16#00.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* exceeds the array area of *In[]*.
  - An array without a subscript is passed to *In[]*.
  - *In[]* is not an array of bit strings.

AryCRCCCITT

The AryCRCCCITT instruction calculates the CRC-CCITT value using the XMODEM method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCRCCCITT	Calculate Array CRC-CCITT	FUN	(@)AryCRCCCITT EN ENO In Out Size Initial OutOrder	Out:=AryCRCCCITT(In, Size, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Initial	Initial value		Initial value of CRC-CCITT value			0
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_HIGH _LOW
Out	CRC-CCITT value	Output	CRC-CCITT value	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Size							OK													
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
Out			OK																	

Function

The AryCRCCCITT instruction calculates the CRC-CCITT value of *Size* elements of array to process *In[]* starting from *In[0]*. The XMODEM method is used.

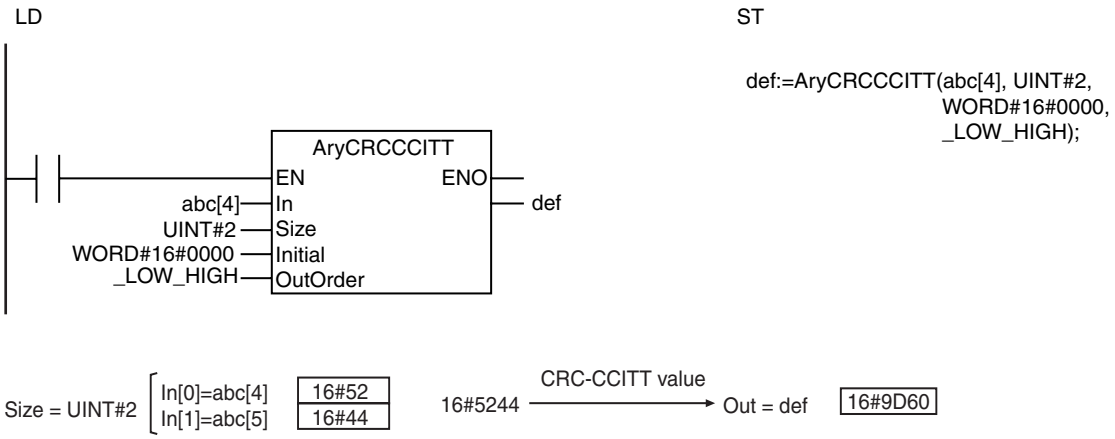
Set *Initial* to the initial value for CRC-CCITT value calculation. *OutOrder* specifies the byte order.

The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

Always attach the element number to in-out parameter that is passed to *In[]*, e.g., *array[3]*.

The following example is for when *Size* is UINT#2, *Initial* is WORD#16#0000, and *OutOrder* is _LOW_HIGH.



Precautions for Correct Use

- If the value of *Size* is 0, the value of *Out* is WORD#16#00.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *OutOrder* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - An array without a subscript is passed to *In[]*.
  - The elements in *In[]* are not bit string, integer, real number, time, duration, date, or date and time data.

AryCRC16

The AryCRC16 instruction calculates the CRC-16 value using the MODBUS method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCRC16	Calculate Array CRC-16	FUN	(@)AryCRC16 EN ENO In Out Size Initial OutOrder	Out:=AryCRC16(In, Size, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Initial	Initial value		Initial value of CRC-16 value			16#FFFF
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_LOW _HIGH
Out	CRC-16 value	Output	CRC-16 value	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Size							OK													
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
Out			OK																	

Function

The AryCRC16 instruction calculates the CRC-16 value of *Size* array elements of array to process *In[]* starting from *In[0]*. The MODBUS method is used.

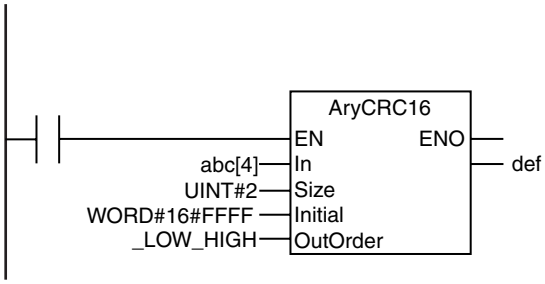
Set *Initial* to the initial value for CRC-16 value calculation. *OutOrder* specifies the byte order.

The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerator	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

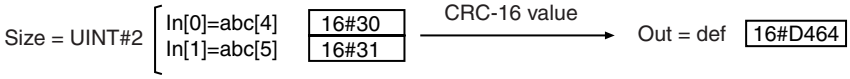
The following example is for when *Size* is UINT#2, *Initial* is WORD#16#FFFF and *OutOrder* is _LOW_HIGH.

LD



ST

```
def:=AryCRC16(abc[4], UINT#2,
              WORD#16#FFFF,
              _LOW_HIGH);
```



Precautions for Correct Use

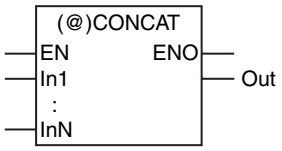
- If the value of *Size* is 0, the value of *Out* is WORD#16#0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *OutOrder* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - An array without a subscript is passed to *In[]*.
  - The elements in *In[]* are not bit string, integer, real number, time, duration, date, or date and time data.

Text String Instructions

Instruction	Name	Page
CONCAT	Concatenate String	2-520
LEFT and RIGHT	Get String Left/Get String Right	2-522
MID	Get String Any	2-524
FIND	Find String	2-526
LEN	String Length	2-528
REPLACE	Replace String	2-529
DELETE	Delete String	2-531
INSERT	Insert String	2-533
GetByteLen	Get Byte Length	2-535
ClearString	Clear String	2-537
ToUCase and ToLCase	Convert to Uppercase/ Convert to Lowercase	2-538
TrimL and TrimR	Trim String Left/Trim String Right	2-540

CONCAT

The CONCAT instruction joins two to five text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CONCAT	Concatenate String	FUN		Out:=CONCAT(In1,···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Strings to join	Input	Text strings to join, where N is 2 to 5	Depends on data type.	---	"*
Out	Result of joining	Output	Text string that resulted from joining	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

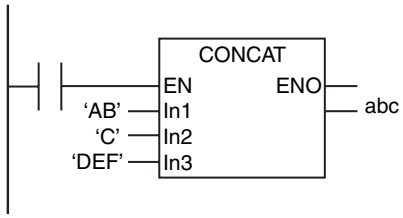
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN																				OK
Out																				OK

Function

The CONCAT instruction joins 2 to 5 text strings in strings to join *In1* to *InN* in that order. It adds a NULL character to the end.

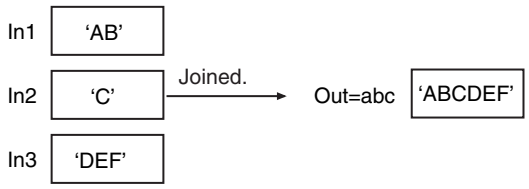
The following example is for when *In1* is 'AB', *In2* is 'C' and *In3* is 'DEF'. The value of variable *abc* will be 'ABCDEF'.

LD



ST

```
abc:=CONCAT('AB', 'C', 'DEF');
```



Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

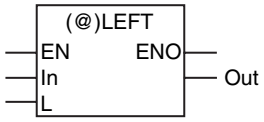
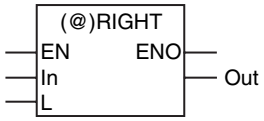
- One of *In1* to *InN* does not end in a NULL character.
- The length of the joined character strings exceeds the size of *Out*.

LEFT and RIGHT

These instructions extract a text string with the specified number of characters.

LEFT: Extracts characters from the left (beginning) of the text string.

RIGHT: Extracts characters from the right (end) of the text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LEFT	Get String Left	FUN		Out:=LEFT(In, L);
RIGHT	Get String Right	FUN		Out:=RIGHT(In, L);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Source string	Input	Text string from which to extract characters	Depends on data type.	---	"
L	Number of characters		Number of characters to extract	0 to 1985		1
Out	Extraction result	Output	Extracted text string	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
L							OK													
Out																				OK

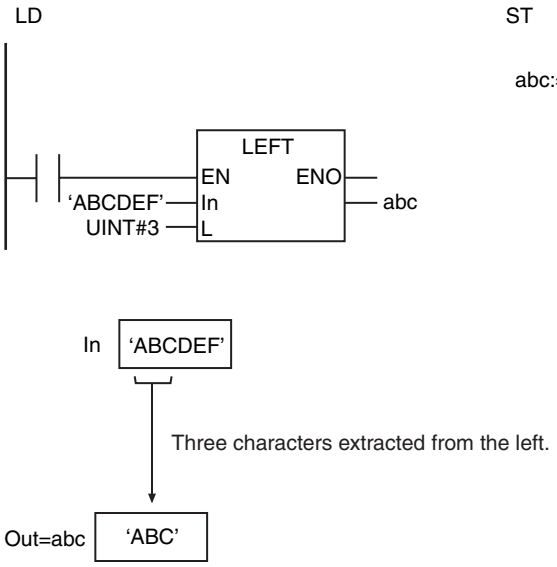
Function

These instructions extract a text string with the number of characters specified by number of characters *L* from the source string *In*. A NULL character is placed at the end of extraction result *Out*.

● LEFT

Extracts characters from the left (beginning) of *In*.

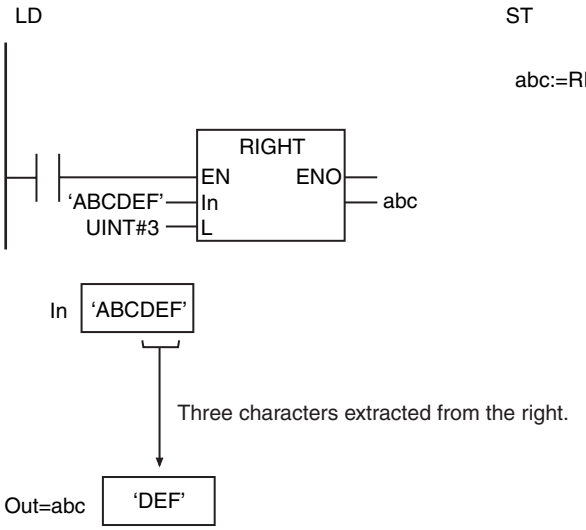
The following example is for when *In* is ‘ABCDEF’ and *L* is UINT#3. The value of variable *abc* will be ‘ABC’.



● RIGHT

Extracts characters from the right (end) of *In*.

The following example is for when *In* is ‘ABCDEF’ and *L* is UINT#3. The value of variable *abc* will be ‘DEF’.

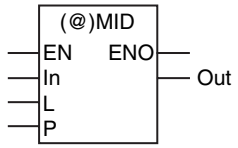


Precautions for Correct Use

- If the value of *L* is larger than the number of characters in *In* or it is within the valid range, an error does not occur and all of the characters in *In* are copied to *Out*.
- If the value of *L* is 0, an error does not occur and only the NULL character is assigned to *Out*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* does not end in a NULL character.
  - *In* results in a character code error.
  - The execution result exceeds the size of *Out.*

MID

The MID instruction extracts a text string with the specified number of characters from the specified character position.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MID	Get String Any	FUN		Out:=MID(In, L, P);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Source string	Input	Text string from which to extract characters	Depends on data type.	---	"
L	Number of characters		Number of characters to extract	0 to 1985		1
P	First character		First character to extract	1 to 1985		
Out	Extraction result	Output	Extracted text string	Depends on data type.	---	---

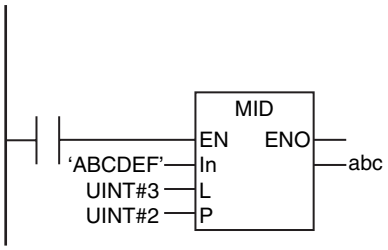
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
L							OK													
P							OK													
Out																				OK

Function

The MID instruction extracts a text string with the number of characters specified by number of characters *L* from the source string *In*. The first character to extract is specified by first character *P*. A NULL character is placed at the end of extraction result *Out*.

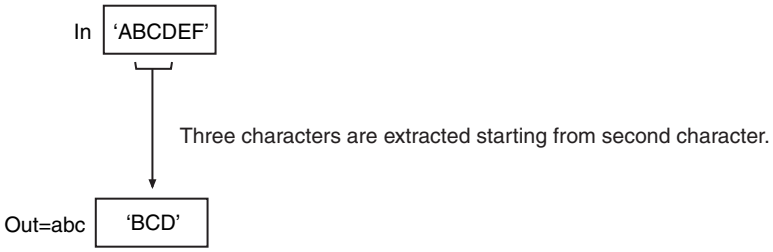
The following example is for when *In* is ‘ABCDEF’, *L* is UINT#3, and *P* is UINT#2. The value of variable *abc* will be ‘BCD’.

LD



ST

```
abc:=MID('ABCDEF', UINT#3, UINT#2);
```

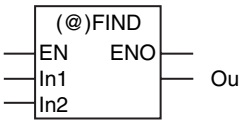


Precautions for Correct Use

- If the value of *L* is 0, an error does not occur and only the NULL character is assigned to *Out*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* does not end in a NULL character.
  - *In* results in a character code error.
  - *In* does not have enough characters for the number of characters specified by *L* from the position specified by *P*.
  - The value of *P* is 0.
  - The execution result exceeds the size of *Out.*

FIND

The FIND instruction searches a specified text string for the position of a specified text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FIND	Find String	FUN		Out:=FIND(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	String to search	Input	Text string to search	Depends on data type.	---	"
In2	Search key		Text string to search for			
Out	Search result	Output	Search result	0 to 1985	---	---

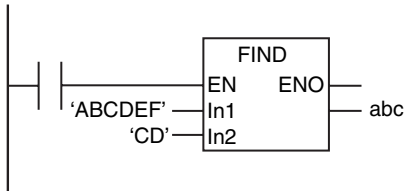
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				OK
In2																				OK
Out							OK													

Function

The FIND instruction searches for search key *In2* in string to search *In1*. The position of *In2* from the start of *In1* is assigned to search result *Out*. If *In2* is not found in *In1*, *Out* is 0.

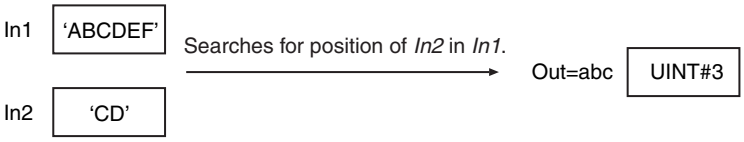
The following example is for when *In1* is ‘ABCDEF’ and *In2* is ‘CD’. The value of variable *abc* will be UINT#3.

LD



ST

```
abc:=FIND(‘ABCDEF’, ‘CD’);
```



Precautions for Correct Use

- Make sure the number of characters in *In2* is less than the number of characters in *In1*. Otherwise, the value of *Out* will be 0.
- If *In2* exists more than once in *In1*, the position of the first *In2* from the beginning of *In1* is assigned to *Out*.
- If the value of *In1* and *In2* is only the NULL character, the value of *Out* is 1.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In1* or *In2* does not end in a NULL character.
  - *In1* or *In2* results in a character code error.

LEN

The LEN instruction finds the number of characters in a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LEN	String Length	FUN	(@)LEN EN ENO In Out	Out:=LEN(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Length string	Input	Text string to find length	Depends on data type.	---	''
Out	Find result	Output	Length detection result	0 to 1985	---	---

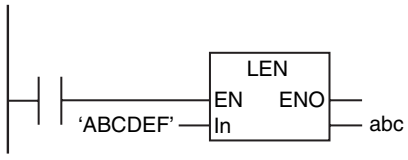
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out							OK													

Function

The LEN instruction finds the number of characters in length string *In*. A NULL character at the end of *In* is not counted.

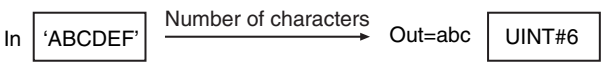
The following example is for when *In* is 'ABCDEF'. The value of variable *abc* will be UINT#6.

LD



ST

```
abc:=LEN('ABCDEF');
```

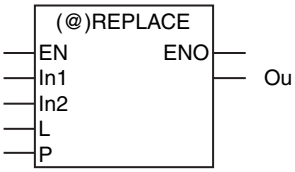


Precautions for Correct Use

- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* does not end in a NULL character.
  - *In* results in a character code error.

REPLACE

The REPLACE instruction replaces part of a text string with another text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
REPLACE	Replace String	FUN		Out:=REPLACE(In1, In2, L, P);

Variables

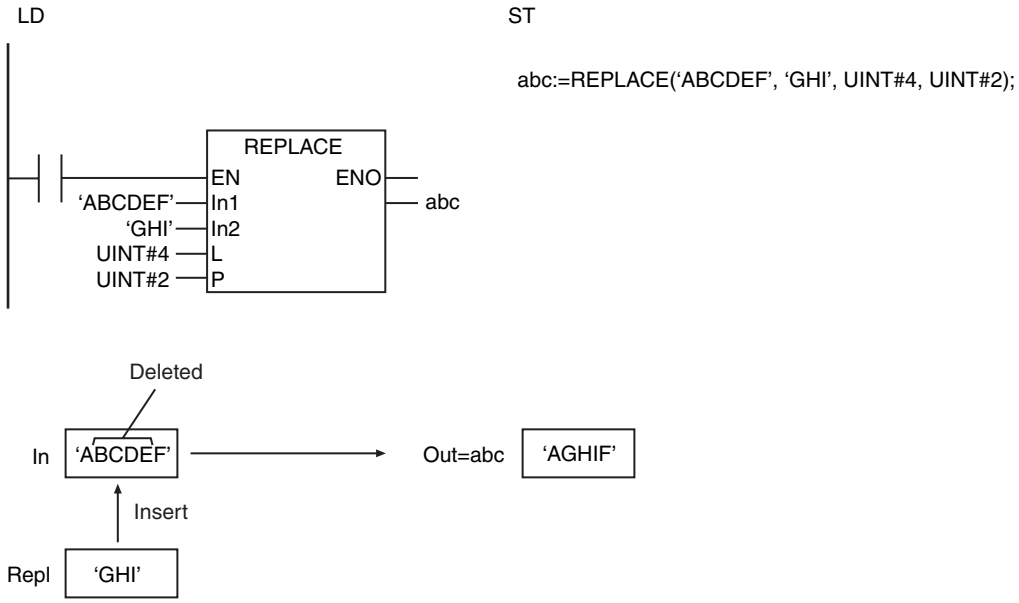
Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	String for replacement	Input	Text string for replacement	Depends on data type.	---	"
In2	Insert string		Text string to insert			
L	Number of characters		Number of characters to delete	0 to 1985		1
P	Replacement start position		Replacement start position	1 to 1985		
Out	Replacement result	Output	Text string after replacement	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				OK
In2																				OK
L							OK													
P							OK													
Out																				OK

Function

The REPLACE instruction replaces part of string for replacement *In1* with string to insert *In2*. First the number of characters specified by *L* from the position specified by *P* are deleted from *In1*. *In2* is then inserted for the deleted characters. A NULL character is placed at the end of replacement result *Out.*

The following example is for when *In1* is ‘ABCDEF’, *In2* is ‘GHI’, *P* is UINT#2, and *L* is UINT#4. The value of variable *abc* will be ‘AGHIF’.

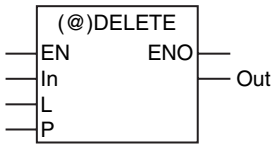


Precautions for Correct Use

- If *L* is 0, an error will not occur and all of the characters in *In1* are inserted to *Out*.
- If the value of *In2* is 0, *L* characters are deleted from *P* in *In1*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In1* or *In2* does not end in a NULL character.
  - *In1* results in a character code error.
  - *In1* does not have enough characters for the number of characters specified by *L* from the position specified by *P*.
  - The value of *P* is 0.
  - The length of the replacement result exceeds the size of *Out*.

DELETE

The DELETE instruction deletes all or part of a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DELETE	Delete String	FUN		Out:=DELETE(In, L, P);

Variables

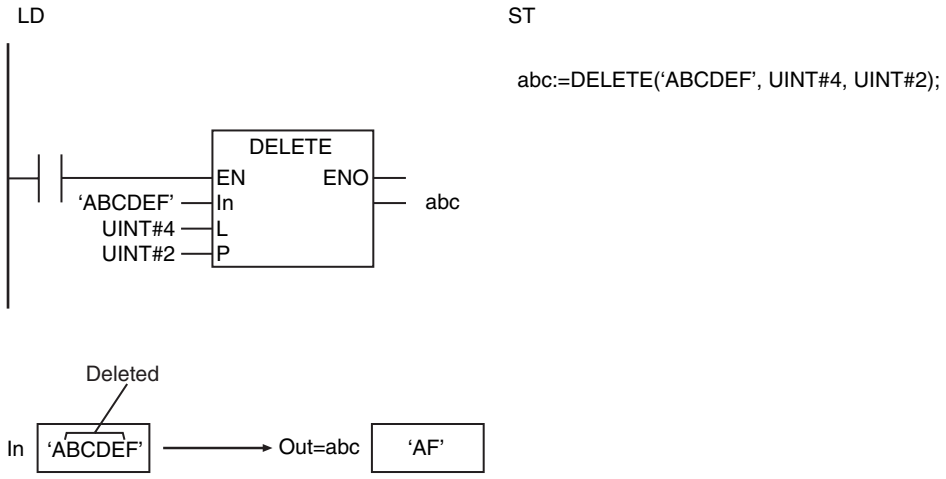
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	String for deletion	Input	Text string for deletion	Depends on data type.	---	''
L	Number of characters		Number of characters to delete	0 to 1985		1
P	Deletion start position		Deletion start position	1 to 1985		
Out	Deletion result	Output	Text string after deletion	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
L							OK													
P							OK													
Out																				OK

Function

The DELETE string deletes the number of characters specified by *L* from the position specified by *P* from *In*. A NULL character is placed at the end of deletion result *Out*.

The following example is for when *In* is 'ABCDEF', *L* is UINT#4, and *P* is UINT#2. The value of variable *abc* will be 'AF'.

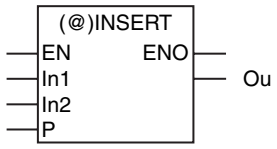


Precautions for Correct Use

- If *L* is 0, an error will not occur and all of the characters in *In* are inserted to *Out*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* does not end in a NULL character.
  - *In* results in a character code error.
  - *In* does not have enough characters for the number of characters specified by *L* from the position specified by *P*.
  - The value of *P* is 0.
  - The execution result exceeds the size of *Out.*

INSERT

The INSERT instruction inserts a text string into another text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
INSERT	Insert String	FUN		Out:=INSERT(In1, In2, P);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original string	Input	Text string into which to insert string	Depends on data type.	---	"
In2	Insert string		Text string to insert			
P	Insertion start position		Insertion start position	0 to 1985		0
Out	Insertion result	Output	Text string after insertion	Depends on data type.	---	---

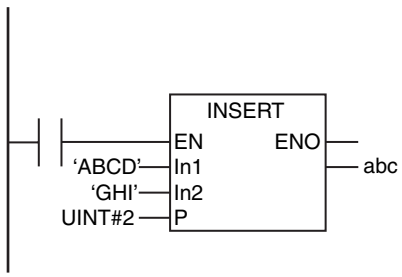
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				OK
In2																				OK
P							OK													
Out																				OK

Function

The INSERT instruction inserts insertion string *In2* into original string *In1* at insertion start position *P*. A NULL character is placed at the end of insertion result *Out*.

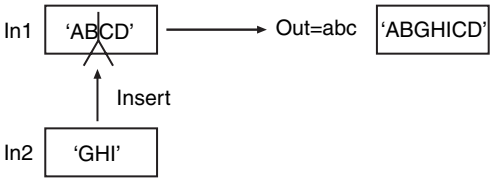
The following example is for when *In1* is 'ABCD', *In2* is 'GHI', and *P* is UINT#2. The value of variable *abc* will be 'ABGHICD'.

LD



ST

```
abc:=INSERT('ABCD', 'GHI', UINT#2);
```



Additional Information

If *P* is 0, *In1* is inserted at the end of *In2*.

Precautions for Correct Use

- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In1* or *In2* does not end in a NULL character.
  - *In1* results in a character code error.
  - The value of *P* is greater than the number of characters in *In1*.
  - The length of the insertion result exceeds the size of *Out*.

GetByteLen

The GetByteLen instruction counts the number of bytes in a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetByteLen	Get Byte Length	FUN	(@)GetByteLen EN ENO In Out	Out:=GetByteLen(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Count string	Input	Text string to count number of bytes	Depends on data type.	---	''
Out	Number of bytes	Output	Number of bytes	0 to 1985	Bytes	---

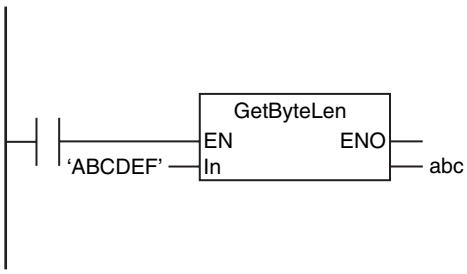
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out							OK													

Function

The GetByteLen instruction counts the number of bytes in count string *In.* A NULL character at the end of the text string is not counted.

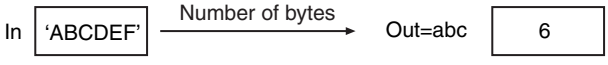
The following example is for when *In* is 'ABCDEF'. The value of variable *abc* will be 6.

LD



ST

```
abc:=GetByteLen('ABCDEF');
```



Additional Information

If *In* contains only ASCII characters, the result will be the same as the result of the LEN instruction.

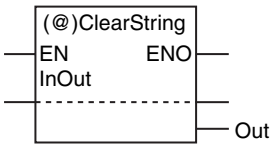
Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- *In* does not end in a NULL character.

ClearString

The ClearString instruction clears a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ClearString	Clear String	FUN		ClearString(InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Clear string	In-out	Text string to clear	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

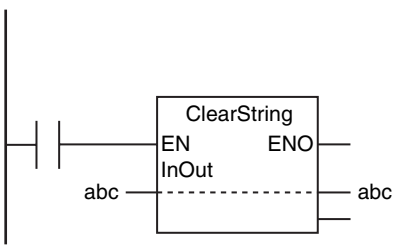
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut																				OK
Out	OK																			

Function

The ClearString instruction clears clear string *InOut*. NULL characters are stored in the entire range of *InOut*.

The following figure shows a programming example. The content of STRING variable will be all NULL characters.

LD



ST

```
ClearString(abc);
```

Precautions for Correct Use

Return value *Out* is not used when the instruction is used in ST.

ToUCase and ToLCase

ToUCase: Converts all single-byte letters in a text string to uppercase.

ToLCase: Converts all single-byte letters in a text string to lowercase.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ToUCase	Convert to Uppercase	FUN	(@)ToUCase EN ENO In Out	Out:=ToUCase(In);
ToLCase	Convert to Lowercase	FUN	(@)ToLCase EN ENO In Out	Out:=ToLCase(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Text string to convert	Depends on data type.	---	''
Out	Conversion result	Output	Converted text string	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out																				OK

Function

● ToUCase

The ToUCase instruction converts all single-byte letters in data to convert *In* to uppercase.

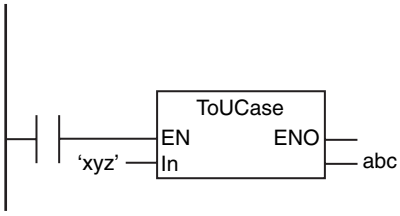
● ToLCase

The ToLCase instruction converts all single-byte letters in data to convert *In* to lowercase.

Both instructions output a NULL character at the end of the text string. Only single-byte characters are changed.

The following example for the ToUCase instruction is for when *In* is ‘xyz’. The value of variable *abc* will be ‘XYZ’.

LD



ST

```
abc:=ToUCase(‘xyz’);
```

Precautions for Correct Use

- Two-byte letters are not converted.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* does not end in a NULL character.
  - *In* results in a character code error.
  - The conversion result exceeds the size of *Out*.

TrimL and TrimR

TrimL: Removes blank space from the beginning of a text string.

TrimR: Removes blank space from the end of a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TrimL	Trim String Left	FUN		Out:=TrimL(In);
TrimR	Trim String Right	FUN		Out:=TrimR(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	String to trim	Input	Text string to trim	Depends on data type.	---	''
Out	Trimming result	Output	Text string after trimming	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out																				OK

Function

TrimL

The TrimL instruction deletes blank characters from the beginning of string to trim *In*. If there are no blank characters at the beginning of the text string, nothing is done.

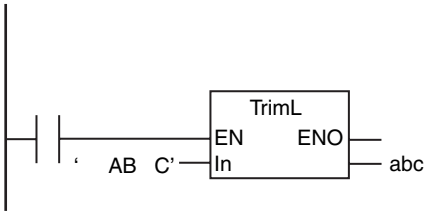
TrimR

The TrimR instruction deletes blank characters from the end of string to trim *In*. If there are no blank characters at the end of the text string, nothing is done.

Both instructions output a NULL character at the end of the text string. Both ASCII spaces (16#20) and two-byte Japanese spaces (16#E38080) are treated as blank characters.

The following example for the TrimL instruction is for when *In* is ‘ AB C’. The value of variable *abc* will be ‘ AB C’.

LD



ST

```
abc:=TrimL(‘ AB C’);
```

Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- *In* does not end in a NULL character.
- *In* results in a character code error.
- The conversion result exceeds the size of *Out*.

Time and Time of Day Instructions

Instruction	Name	Page	Instruction	Name	Page
ADD_TIME	Add Time	2-544	DateToSec	Convert Date to Seconds	2-576
ADD_TOD_TIME	Add Time to Time of Day	2-546	TodToSec	Convert Time of Day to Seconds	2-577
ADD_DT_TIME	Add Time to Date and Time	2-548	SecToDt	Convert Seconds to Date and Time	2-578
SUB_TIME	Subtract Time	2-550	SecToDate	Convert Seconds to Date	2-580
SUB_TOD_TIME	Subtract Time from Time of Day	2-552	SecToTod	Convert Seconds to Time of Day	2-582
SUB_TOD_TOD	Subtract Time of Day	2-554	TimeToNanoSec	Convert Time to Nanoseconds	2-583
SUB_DATE_DATE	Subtract Date	2-555	TimeToSec	Convert Time to Seconds	2-584
SUB_DT_DT	Subtract Date and Time	2-556	NanoSecToTime	Convert Nanoseconds to Time	2-585
SUB_DT_TIME	Subtract Time from Date and Time	2-558	SecToTime	Convert Seconds to Time	2-586
MULTIME	Multiply Time	2-560	ChkLeapYear	Check for Leap Year	2-588
DIVTIME	Divide Time	2-562	GetDaysOfMonth	Get Days in Month	2-589
CONCAT_DATE_TOD	Concatenate Date and Time of Day	2-564	DaysToMonth	Convert Days to Month	2-591
DT_TO_TOD	Extract Time of Day from Date and Time	2-566	GetDayOfWeek	Get Day of Week	2-593
DT_TO_DATE	Extract Date from Date and Time	2-568	GetWeekOfYear	Get Week Number	2-595
SetTime	Set Time	2-570	DtToDateStruct	Break Down Date and Time	2-597
GetTime	Get Time of Day	2-572	DateStructToDt	Join Time	2-599
DtToSec	Convert Date and Time to Seconds	2-574			

ADD_TIME

The ADD_TIME instruction adds two times.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD_TIME	Add Time	FUN	(@)ADD_TIME EN ENO In1 Out In2	Out:=ADD_TIME(In1, In2);

Variables

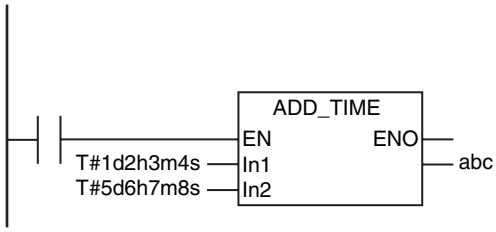
Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Add time 1	Input	Add time 1	Depends on data type.	ns	T#0s
In2	Add time 2		Add time 2			
Out	Total time	Output	Total time	Depends on data type.	ns	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2																OK				
Out																OK				

Function

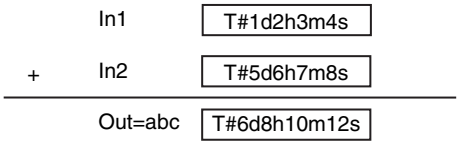
The ADD_TIME instruction adds two times, *In1* and *In2*. The result of addition in *Out* is also a time.
The following example is for when *In1* is T#1d2h3m4s and *In2* is T#5d6h7m8s.

LD



ST

```
abc:=ADD_TIME(T#1d2h3m4s, T#5d6h7m8s);
```



Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*.

- T#106751d_23h_47m_16s_854.775807ms + T#0.000001ms
  → T#-106751d_23h_47m_16s_854.775808ms
- T#-106751d_23h_47m_16s_854.775808ms + T#-0.000001ms
  → T#106751d_23h_47m_16s_854.775807ms

ADD_TOD_TIME

The ADD_TOD_TIME instruction adds a time to a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD_TOD_TIME	Add Time to Time of Day	FUN	(@)ADD_TOD_TIME EN ENO In1 Out In2	Out:=ADD_TOD_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Add time of day	Input	Add time of day	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
In2	Add time		Add time		ns	T#0s
Out	Resulting time of day	Output	Resulting time of day	Depends on data type.	Hour, minutes, seconds	---

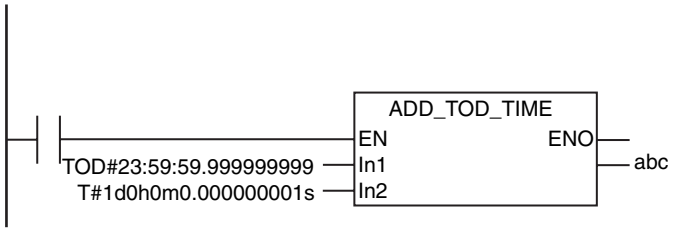
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		OK		
In2																OK				
Out																		OK		

Function

The ADD_TOD_TIME instruction adds a time, *In2,* to a time of day *In1.* The result of addition in *Out* is also a time of day.

The following example is for when *In1* is TOD#23:59:59.999999999 and *In2* is T#1d0h0m0.000000001s.

LD



	In1	TOD#23:59:59.999999999
+	In2	T#1d0h0m0.000000001s
	Out=abc	TOD#0:0:0.000000000

ST

```
abc:=ADD_TOD_TIME(TOD#23:59:59.999999999,
                  T#1d0h0m0.000000001s);
```

Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*.

- TOD#23:59:59.999999999 + T#0.000001ms → TOD#0:0:0.000000000
- TOD#0:0:0.000000000 + T#-0.000001ms → TOD#23:59:59.999999999

ADD_DT_TIME

The ADD_DT_TIME instruction adds a time to a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD_DT_TIME	Add Time to Date and Time	FUN	(@)ADD_DT_TIME EN ENO In1 Out In2	Out:=ADD_DT_TIME(In1, In2);

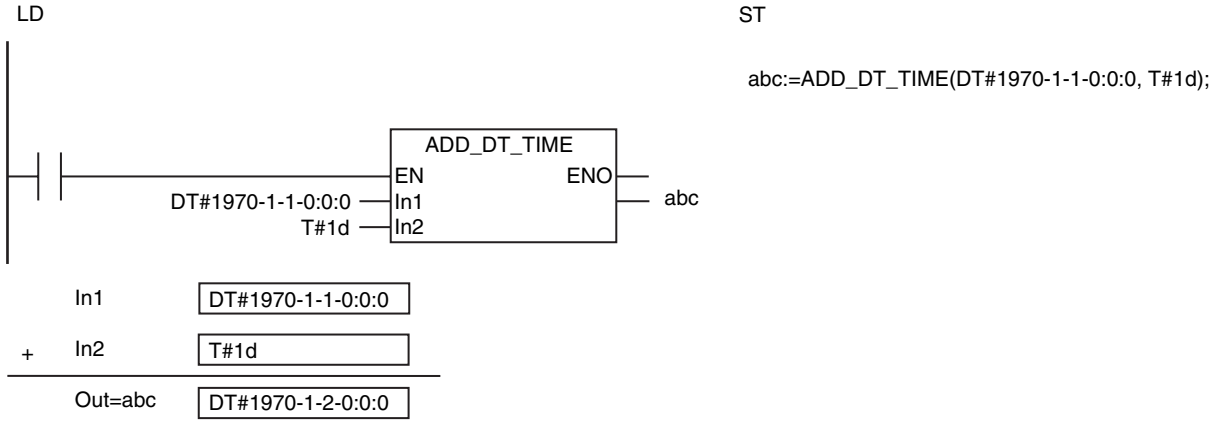
Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Add date and time	Input	Add date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
In2	Add time		Add time		ns	T#0s
Out	Addition result date and time	Output	Addition result date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			OK	
In2																OK				
Out																			OK	

Function

The ADD_DT_TIME instruction adds a time, *In2,* to a date and time *In1.* The result of addition in *Out* is also a date and time. Leap years are also accounted for. The following example is for when *In1* is DT#1970-1-1-0:0:0 and *In2* is T#1d.



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*.

- DT#2554-7-21-23:34:33.709551615 + T#0.000001ms → DT#1970-1-1-0:0:0
- DT#1970-1-1-0:0:0 + T#-0.000001ms → DT#2554-7-21-23:34:33.709551615

SUB_TIME

The SUB_TIME instruction subtracts one time from another.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_TIME	Subtract Time	FUN	(@)SUB_TIME EN ENO In1 Out In2	Out:=SUB_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original time	Input	Original time	Depends on data type.	ns	T#0s
In2	Time to subtract		Time to subtract			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

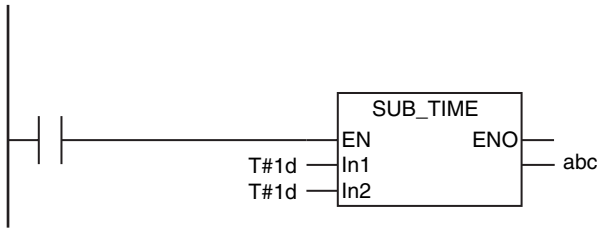
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2																OK				
Out																OK				

Function

The SUB_TIME instruction subtracts a time *In2* from another time *In1.* The result of subtraction in *Out* is also a time.

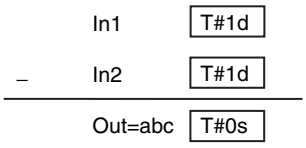
The following example is for when *In1* and *In2* are T#1d.

LD



ST

```
abc:=SUB_TIME(T#1d, T#1d);
```



Precautions for Correct Use

An error will not occur even if the subtraction result exceeds the valid range of *Out*.

- T#106751d_23h_47m_16s_854.775807ms − T#-0.000001ms
  → T#-106751d_23h_47m_16s_854.775808ms
- T#-106751d_23h_47m_16s_854.775808ms − T#0.000001ms
  → T#106751d_23h_47m_16s_854.775807ms

SUB_TOD_TIME

The SUB_TOD_TIME instruction subtracts a time from a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_TOD_TIME	Subtract Time from Time of Day	FUN	(@)SUB_TOD_TIME EN In1 In2 ENO Out	Out:=SUB_TOD_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Time of day	Input	Time of day	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
In2	Time to subtract		Time to subtract		ns	T#0s
Out	Resulting time of day	Output	Resulting time of day	Depends on data type.	Hour, minutes, seconds	---

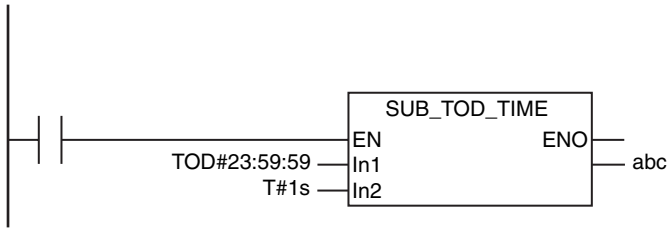
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		OK		
In2																OK				
Out																		OK		

Function

The SUB_TOD_TIME instruction subtracts a time *In2* from a time of day *In1.* The result of subtraction in *Out* is also a time of day.

The following example is for when *In1* is TOD#23:59:59 and *In2* is T#1s.

LD



ST

```
abc:=SUB_TOD_TIME(TOD#23:59:59, T#1s);
```

	In1	TOD#23:59:59
–	In2	T#1s
	Out=abc	TOD#23:59:58

Precautions for Correct Use

An error will not occur even if the subtraction result exceeds the valid range of *Out*.

- TOD#23:59:59.999999999 − T#-0.000001ms → TOD#0:0:0
- TOD#0:0:0 − T#0.000001ms → TOD#23:59:59.999999999

SUB_TOD_TOD

The SUB_TOD_TOD instruction subtracts a time of day from another time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_TOD_TOD	Subtract Time of Day	FUN	(@)SUB_TOD_TOD EN ENO In1 Out In2	Out:=SUB_TOD_TOD(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Time of day 1	Input	Time of day 1	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
In2	Time of day 2		Time of day 2			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

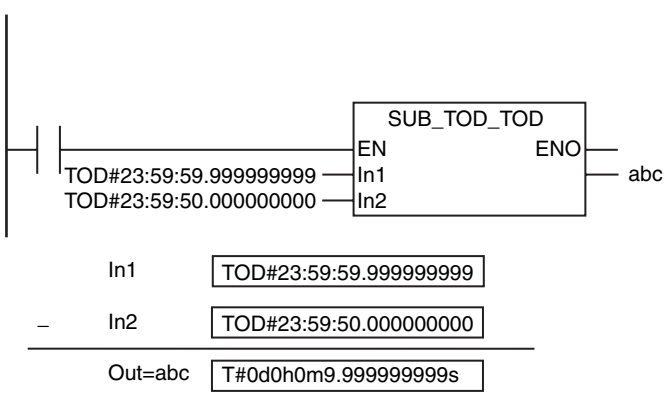
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		OK		
In2																		OK		
Out																OK				

Function

The SUB_TOD_TOD instruction subtracts time of day *In2* from time of day *In1*. The result of subtraction in *Out* is a time.

The following example is for when *In1* is TOD#23:59:59.999999999 and *In2* is TOD#23:59:50.000000000.

LD



ST

```
abc:=SUB_TOD_TOD(TOD#23:59:59.999999999,
                 TOD#23:59:50.000000000);
```

SUB_DATE_DATE

The SUB_DATE_DATE instruction subtracts another date from another date.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_DATE_DATE	Subtract Date	FUN	(@)SUB_DATE_DATE EN ENO In1 Out In2	Out:=SUB_DATE_DATE(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date 1	Input	Date 1	Depends on data type.	Year, month, day	D#1970-1-1
In2	Date 2		Date 2			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

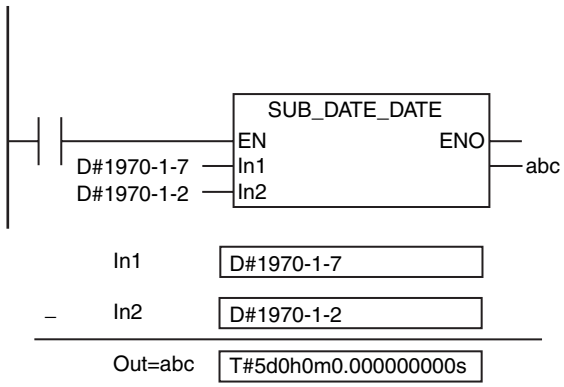
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	OK			
In2																	OK			
Out																OK				

Function

The SUB_DATE_DATE instruction subtracts date *In2* from date *In1*. The result of subtraction in *Out* is a time.

The following example is for when *In1* is D#1970-1-7 and *In2* is D#1970-1-2.

LD



	In1	D#1970-1-7
–	In2	D#1970-1-2
	Out=abc	T#5d0h0m0.000000000s

ST

```
abc:=SUB_DATE_DATE(D#1970-1-7, D#1970-1-2);
```

SUB_DT_DT

The SUB_DT_DT instruction subtracts another date and time from another date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_DT_DT	Subtract Date and Time	FUN	(@)SUB_DT_DT EN ENO In1 Out In2	Out:=SUB_DT_DT(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date and time 1	Input	Date and time 1	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
In2	Date and time 2		Date and time 2			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

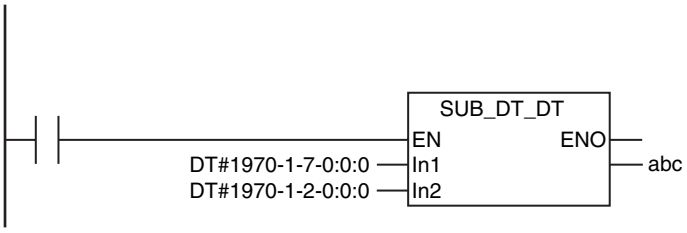
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			OK	
In2																			OK	
Out																OK				

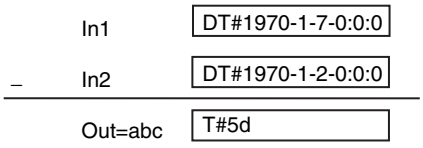
Function

The SUB_DT_DT instruction subtracts date and time *In2* from date and time *In1.* The result of subtraction in *Out* is a time.

The following example is for when *In1* is DT#1970-1-7-0:0:0 and *In2* is DT#1970-1-2-0:0:0.

LD





ST

```
abc:=SUB_DT_DT(DT#1970-1-7-0:0:0,
               DT#1970-1-2-0:0:0);
```

Related System-defined Variables

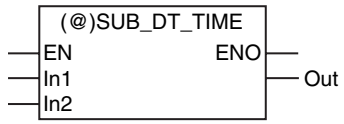
Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

If the processing result exceeds the valid range of *Out*, *Out* will contain an illegal value.

SUB_DT_TIME

The SUB_DT_TIME instruction subtracts a time from a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_DT_TIME	Subtract Time from Date and Time	FUN		Out:=SUB_DT_TIME(In1, In2);

Variables

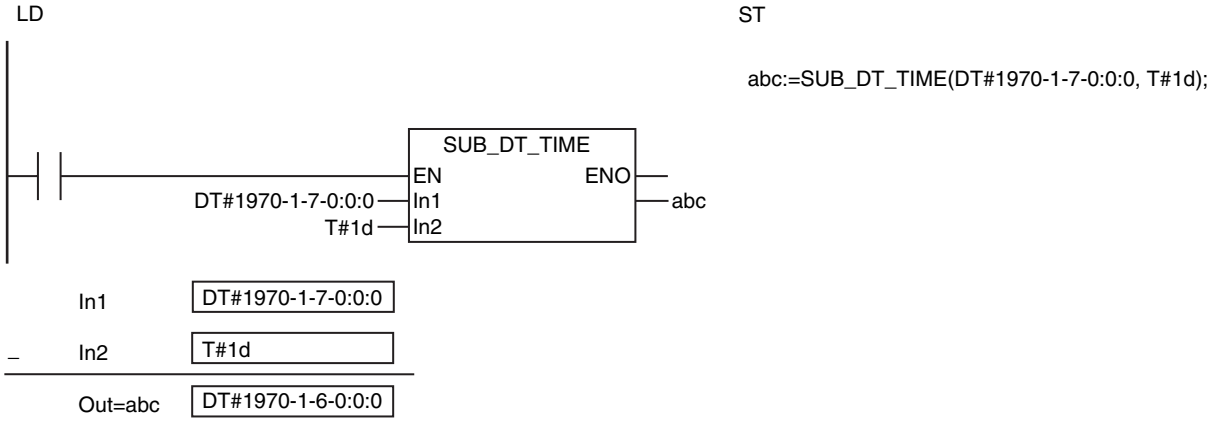
Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
In2	Time to subtract		Time to subtract		ns	T#0s
Out	Resulting date and time	Output	Resulting date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			OK	
In2																OK				
Out																			OK	

Function

The SUB_DT_TIME instruction subtracts a time *In2* from a date and time *In1.* The result of subtraction in *Out* is a date and time. Leap years are also accounted for.

The following example is for when *In1* is DT#1970-1-1-0:0:0 and *In2* is T#1d.



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error will not occur even if the subtraction result exceeds the valid range of *Out*.

- DT#2554-7-21-23:34:33.709551615 − T#-0.000001ms → DT#1970-1-1-0:0:0
- DT#1970-1-1-0:0:0 − T#0.000001ms → DT#2554-7-21-23:34:33.709551615

MULTIME

The MULTIME instruction multiplies a time by a specified number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MULTIME	Multiply Time	FUN	(@)MULTIME EN ENO In1 Out In2	Out:=MULTIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original time	Input	Original time	Depends on data type.	ns	T#0s
In2	Multiplier		Multiplier		---	*
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

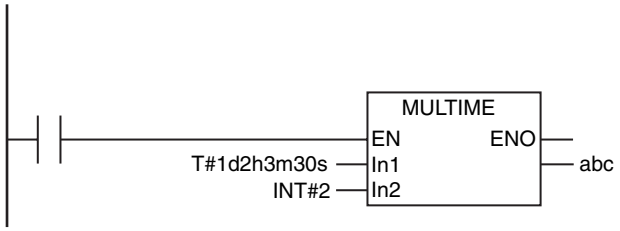
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out																OK				

Function

The MULTIME instruction multiplies a time *In1* by multiplier *In2.* The result of multiplication in *Out* is also a time.

The following example is for when *In1* is T#1d2h3m30s and *In2* is INT#2.

LD



	In1	T#1d2h3m30s
×	In2	INT#2
	Out=abc	T#2d4h7m

ST

```
abc:=MULTIME(T#1d2h3m30s, INT#2);
```

Precautions for Correct Use

- If *In2* is a real number, the multiplication result is rounded to the nearest nanosecond. The following table shows how values are rounded.

Value below nanoseconds	Treatment	Examples
Less than 0.5	The value is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the value is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The value is rounded up.	1.51 → 2

- If the value of *In2* is 0, positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In2*	Value of *Out*
0	T#0s
+∞	T#-106751d23h47m16.854775808s
−∞	T#-106751d23h47m16.854775808s
Nonnumeric data	T#-106751d23h47m16.854775808s

- An error will not occur even if the multiplication result exceeds the valid range of *Out*.
  - T#53375d_23h_53m_38s_427.387904ms * USINT#2
    → T#-106751d_23h_47m_16s_854.775808ms
  - T#-53375d_23h_53m_38s_427.387905ms * USINT#2
    → T#106751d_23h_47m_16s_854.775806ms

DIVTIME

The DIVTIME instruction divides a time by a specified number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DIVTIME	Divide Time	FUN	(@)DIVTIME EN ENO In1 Out In2	Out:=DIVTIME(In1, In2);

Variables

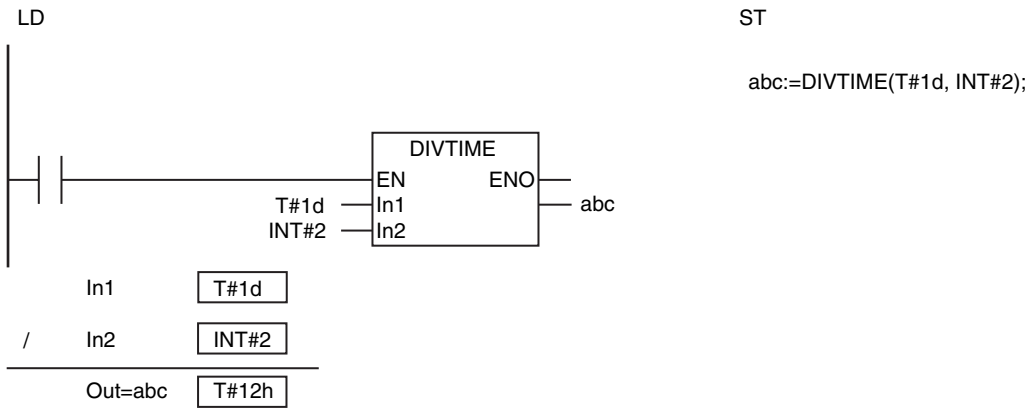
Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original time	Input	Original time	Depends on data type.	ns	T#0s
In2	Number to divide by		Number to divide by		---	*
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out																OK				

Function

The DIVTIME instruction divides a time *In1* by a number *In2.* The result of division in *Out* is also a time. The following example is for when *In1* is T#1d and *In2* is INT#2.



Precautions for Correct Use

- If the value of *In2* is 0, positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In2*	Value of *Out*
0	T#-106751d23h47m16.854775808s
+∞	T#0s
−∞	T#0s
Nonnumeric data	Nonnumeric data

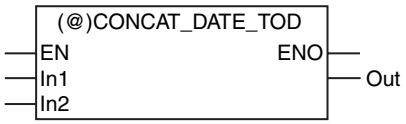
- If *In2* is a real number, there may be error of up to several nanoseconds.
- If *In2* is a real number, the division result is rounded to the nearest nanosecond. The following table shows how values are rounded.

Value below nanoseconds	Description	Example
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the value is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - *In2* is an integer with a value of 0.

CONCAT_DATE_TOD

The CONCAT_DATE_TOD instruction combines a date and a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CONCAT_DATE_TOD	Concatenate Date and Time of Day	FUN		Out:=CONCAT_DATE_TOD (In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date	Input	Date	Depends on data type.	Year, month, day	D#1970-1-1
In2	Time of day		Time of day		Hour, minutes, seconds	TOD#0:0:0
Out	Combined date and time	Output	Combined date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

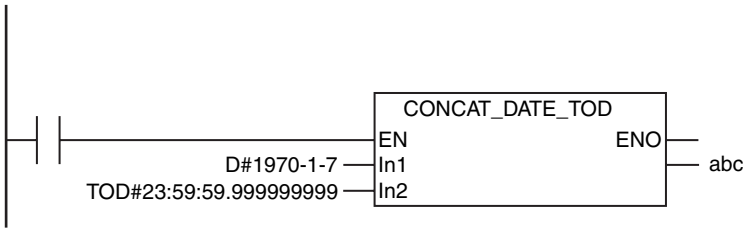
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	OK			
In2																		OK		
Out																			OK	

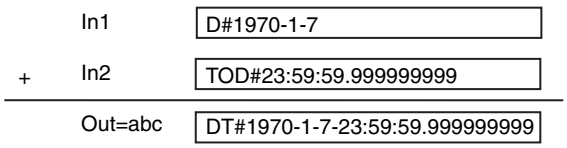
Function

The CONCAT_DATE_TOD instruction combines a date *In1* and a time of day *In2.* The result of combining in *Out* is also a date and time.

The following example is for when *In1* is D#1970-1-7 and *In2* is TOD#23:59:59.999999999.

LD





ST

```
abc:=CONCAT_DATE_TOD(D#1970-1-7,
                     TOD#23:59:59.999999999);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The results of combining exceeds the valid range of *Out* (e.g., the value of *In1* is D#2554-7-21 and the value of *In2* is larger than TOD#23:34:33.709551615).

DT_TO_TOD

The DT_TO_TOD instruction extracts the time of day from a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DT_TO_TOD	Extract Time of Day from Date and Time	FUN	(@)DT_TO_TOD EN ENO In Out	Out:=DT_TO_TOD(In);

Variables

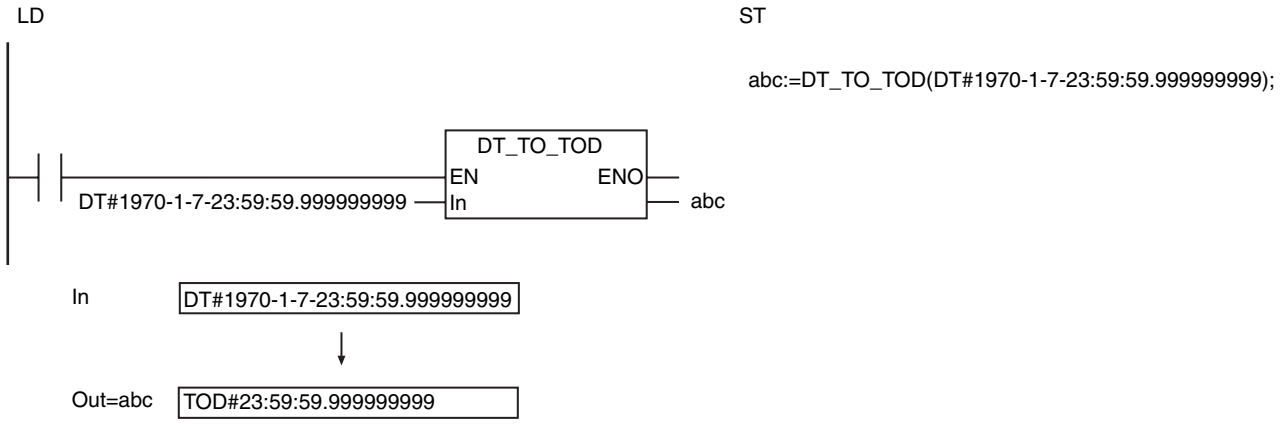
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Time of day	Output	Time of day	Depends on data type.	Hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out																		OK		

Function

The DT_TO_TOD instruction extracts the time of day from date and time *In.*

The following example is for when *In* is DT#1970-1-7-23:59:59.999999999.



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

DT_TO_DATE

The DT_TO_DATE instruction extracts the date from a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DT_TO_DATE	Extract Date from Date and Time	FUN	(@)DT_TO_DATE EN ENO In Out	Out:=DT_TO_DATE(In);

Variables

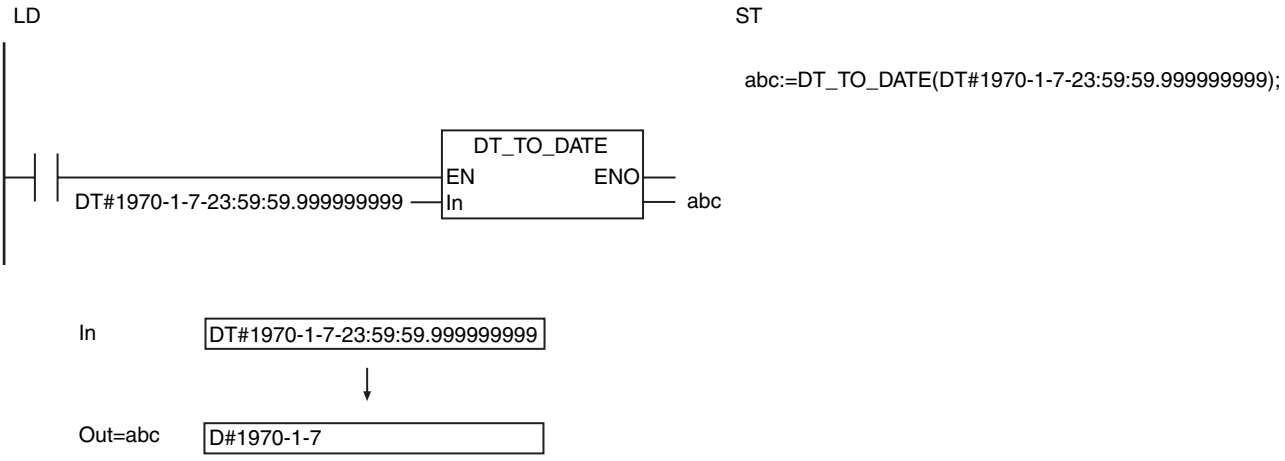
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Date	Output	Date	Depends on data type.	Year, month, day	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out																	OK			

Function

The DT_TO_DATE instruction extracts the date from date and time *In.*

The following example is for when *In* is DT#1970-1-7-23:59:59.999999999.

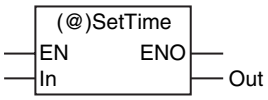


Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

SetTime

The SetTime instruction sets the system time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetTime	Set Time	FUN		SetTime(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time data	Input	Current time to set system time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Return value	Output	Always TRUE	TRUE only	---	---

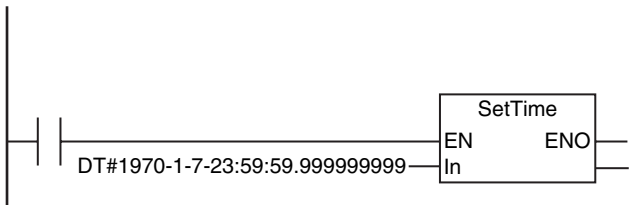
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out	OK																			

Function

The SetTime instruction sets the system time to date and time *In*.

The following programming example is for when *In* is DT#1970-1-7:23:59:59.999999999.

LD



ST

```
SetTime(DT#1970-1-7-23:59:59.999999999);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

The following methods can also be used to set the system time.

- Sysmac Studio
- NTP function

Precautions for Correct Use

- For *In*, specify the time for the set time zone (do not specify Greenwich mean time (GMT)).
- You cannot set a time in *In* that is lower than 1970-1-1-0:0:0.000000000 GMT.
- A time lag will occur when updating the internal time. If the time is read immediately after executing this instruction, the old time may be read.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In* is outside of the valid range.
  - The value of *In* is below 1970-1-1-0:0:0.000000000 GMT.

GetTime

The GetTime instruction reads the current time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetTime	Get Time of Day	FUN	(@)GetTime EN ENO Out	Out:=GetTime();

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Current time	Output	Current time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

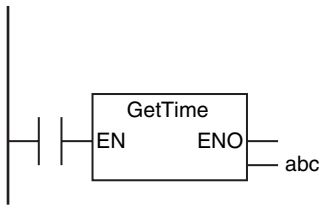
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out																			OK	

Function

The GetTime instruction reads the current time. The current time of day is the time for the set time zone (not Greenwich mean time (GMT)).

The following figure shows a programming example. The current time is assigned to variable *abc.*

LD



ST

```
abc:=GetTime();
```

Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

- Use the DtToSec instruction (page 2-574) to convert the current time of day to the system time of day (number of seconds from 00:00:00 on January 1,1970).
- Use the DtToDateStruct instruction (page 2-597) to convert the current time of day to a date (year, month, day, minutes, and seconds).
- Use the GetDayOfWeek instruction (page 2-593) to read the day of the week.

DtToSec

The DtToSec instruction converts a date and time to the number of seconds from 00:00:00 on January 1, 1970.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DtToSec	Convert Date and Time to Seconds	FUN	(@)DtToSec EN ENO In Out	Out:=DtToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Seconds	Output	Number of seconds from 00:00:00 on January 1, 1970	0 to 18446744073	Seconds	---

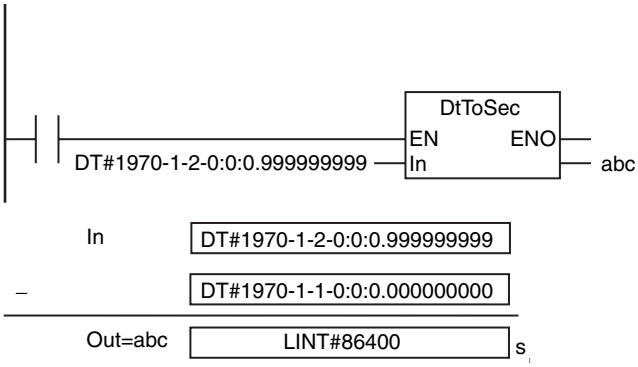
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out													OK							

Function

The DtToSec instruction converts the date and time in *In* to the number of seconds from 00:00:00 on January 1, 1970. The converted value is in seconds. The value is truncated below the seconds.

The following example is for when *In* is DT#1970-1-2-0:0:0.999999999.

LD



ST

```
abc:=DtToSec(DT#1970-1-2-0:0:0.999999999);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

Use the SecToDt instruction (page 2-578) to convert the number of seconds from 00:00:00 on January 1,1970 to a date and time.

DateToSec

The DateToSec instruction converts a date to the number of seconds from 00:00:00 on January 1, 1970.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DateToSec	Convert Date to Seconds	FUN	(@)DateToSec EN ENO In Out	Out:=DateToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date	Input	Date	Depends on data type.	Year, month, day	DT#1970-1-1
Out	Seconds	Output	Number of seconds from 00:00:00 on January 1, 1970	0 to 18446659200	Seconds	---

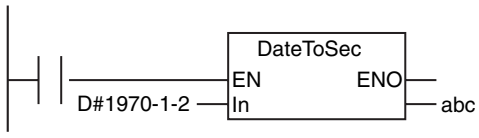
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK			
Out													OK							

Function

The DateToSec instruction converts 00:00:00 on date *In* to the number of seconds from 00:00:00 on January 1, 1970. The converted value is in seconds.

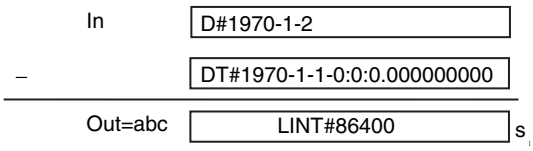
The following example is for when *In* is D#1970-1-2.

LD



ST

```
abc:=DateToSec(D#1970-1-2);
```



Additional Information

Use the SecToDate instruction (page 2-580) to convert the number of seconds from 00:00:00 on January 1,1970 to a date.

TodToSec

The TodToSec instruction converts a time of day to the number of seconds from 00:00:00.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TodToSec	Convert Time of Day to Seconds	FUN	(@)TodToSec EN ENO In Out	Out:=TodToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time of day	Input	Time of day	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
Out	Seconds	Output	Number of seconds from 00:00:00	0 to 86399	Seconds	---

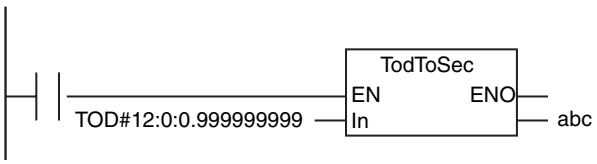
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		OK		
Out													OK							

Function

The TodToSec instruction converts the time of day in *In* to the number of seconds from 00:00:00. The converted value is in seconds. The value is truncated below the seconds.

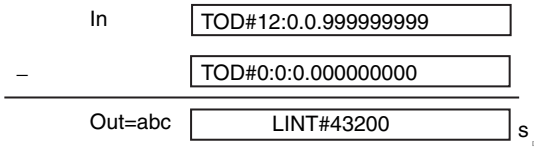
The following example is for when *In* is TOD#12:0:0.999999999.

LD



ST

```
abc:=TodToSec(TOD#12:0:0.999999999);
```



Additional Information

Use the SecToTod instruction (page 2-582) to convert the number of seconds from 00:00:00 on January 1,1970 to a time of day.

SecToDt

The SecToDt instruction converts the number of seconds from 00:00:00 on January 1, 1970 to a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToDt	Convert Seconds to Date and Time	FUN	(@)SecToDt EN ENO In Out	Out:=SecToDt(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Number of seconds from 00:00:00 on January 1, 1970	0 to 18446744073	Seconds	0
Out	Date and time	Output	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

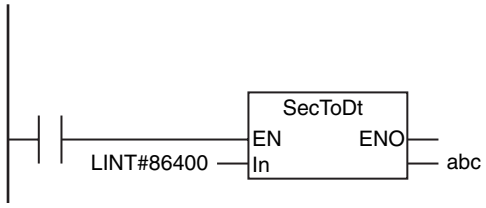
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																			OK	

Function

The SecToDt instruction converts the number of seconds from 00:00:00 on January 1, 1970 in *In* to a date and time.

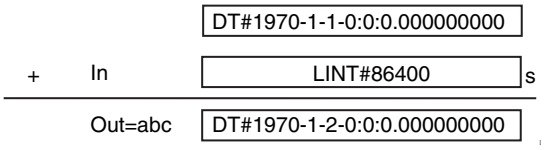
The following example is for when *In* is LINT#86400.

LD



ST

```
abc:=SecToDt(LINT#86400);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

Use the DtToSec instruction (page 2-574) to convert the current time of day to the number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.

SecToDate

The SecToDate instruction converts the number of seconds from 00:00:00 to a date.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToDate	Convert Seconds to Date	FUN	(@)SecToDate EN ENO In Out	Out:=SecToDate(In);

Variables

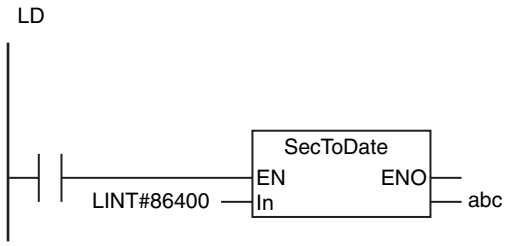
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Number of seconds from 00:00:00	0 to 18446744073	Seconds	0
Out	Date	Output	Date	Depends on data type.	Year, month, day	---

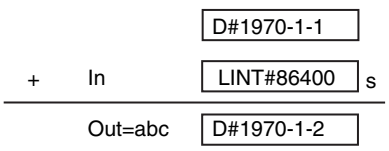
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																	OK			

Function

The SecToDate instruction converts the number of seconds from 00:00:00 in *In* to a date. The value is truncated below date.

The following example is for when *In* is LINT#86400.





Additional Information

Use the DateToSec instruction (page 2-576) to convert a date to the number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.

SecToTod

The SecToTod instruction converts the number of seconds from 00:00:00 to a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToTod	Convert Seconds to Time of Day	FUN	(@)SecToTod EN ENO In Out	Out:=SecToTod(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Number of seconds from 00:00:00	Depends on data type.*	Seconds	0
Out	Time of day	Output	Time of day	Depends on data type.	Hour, minutes, seconds	---

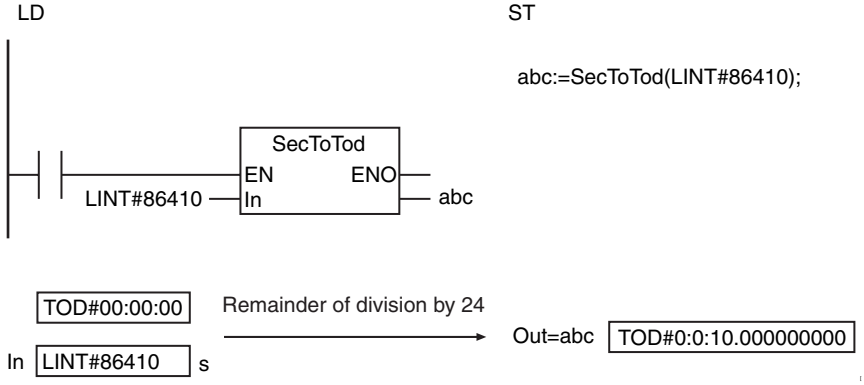
* Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																		OK		

Function

The SecToTod instruction converts the number of seconds from 00:00:00 in *In* to a time of day. If the value of *In* is 24 hours or longer, *In* is divided by 24 and the remainder is converted to the time of day.

The following example is for when *In* is LINT#86410.



Additional Information

Use the TodToSec instruction (page 2-577) to convert a time of day to the number of seconds from 00:00:00 on January 1,1970.

TimeToNanoSec

The TimeToNanoSec instruction converts a time to nanoseconds.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TimeToNanoSec	Convert Time to Nanoseconds	FUN	(@)TimeToNanoSec EN ENO In Out	Out:=TimeToNanoSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time	Input	Time	Depends on data type.	ns	T#0s
Out	Nanosec-onds	Output	Nanoseconds	*	ns	---

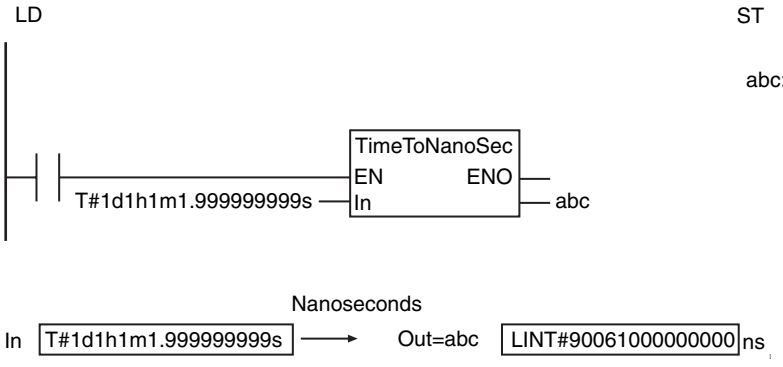
* −9223372036854775808 to 9223372036854775807

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																OK				
Out													OK							

Function

The TimeToNanoSec instruction converts the time in *In* to nanoseconds.

The following example is for when *In* is T#1d1h1m1.999999999s.



Additional Information

Use the NanoSecToTime instruction (page 2-585) to convert nanoseconds to a time.

TimeToSec

The TimeToSec instruction converts a time to seconds.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TimeToSec	Convert Time to Seconds	FUN	(@)TimeToSec EN ENO In Out	Out:=TimeToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time	Input	Time	Depends on data type.	ns	T#0s
Out	Seconds	Output	Seconds	–9223372036 to 9223372036	Seconds	---

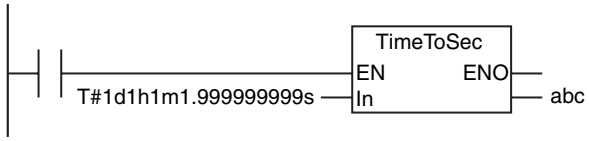
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																OK				
Out							OK													

Function

The TimeToSec instruction converts the time in *In* to seconds. The value is truncated below the seconds.

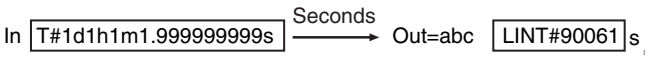
The following example is for when *In* is T#1d1h1m1.999999999s.

LD



ST

```
abc:=TimeToSec(T#1d1h1m1.999999999s);
```



Additional Information

Use the SecToTime instruction (page 2-586) to convert seconds to a time.

Precautions for Correct Use

In is in nanoseconds. *Out* is in seconds.

NanoSecToTime

The NanoSecToTime instruction converts nanoseconds to a time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NanoSecToTime	Convert Nanoseconds to Time	FUN	(@)NanoSecToTime EN ENO In Out	Out:=NanoSecToTime(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Nanoseconds	Input	Nanoseconds	*	ns	0
Out	Time	Output	Time	Depends on data type.	ns	---

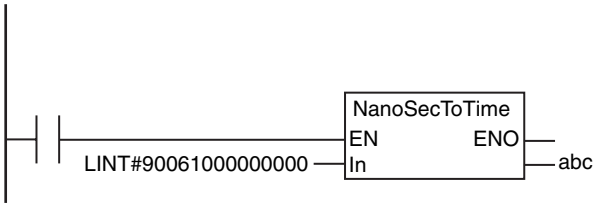
* −9223372036854775808 to 9223372036854775807

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																OK				

Function

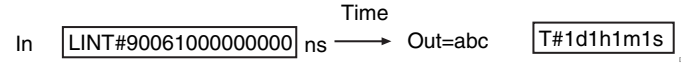
The NanoSecToTime instruction converts the number of nanoseconds in *In* to a time.
The following example is for when *In* is LINT#90061000000000.

LD



ST

```
abc:=NanoSecToTime(LINT#90061000000000);
```



Additional Information

Use the TimeToNanoSec instruction (page 2-583) to convert a time to nanoseconds.

SecToTime

The SecToTime instruction converts seconds to a time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToTime	Convert Seconds to Time	FUN	(@)SecToTime EN ENO In Out	Out:=SecToTime(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Seconds	–9223372036 to 9223372036	Seconds	0
Out	Time	Output	Time	Depends on data type.	ns	---

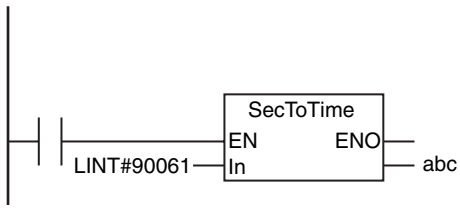
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																OK				

Function

The SecToTime instruction converts the number of seconds in *In* to a time.

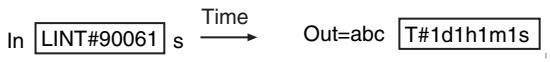
The following example is for when *In* is LINT#90061.

LD



ST

```
abc:=SecToTime(LINT#90061);
```



Additional Information

Use the TimeToSec instruction (page 2-584) to convert a time to seconds.

Precautions for Correct Use

- *In* is in seconds. *Out* is in nanoseconds.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In* is outside of the valid range.

ChkLeapYear

The ChkLeapYear instruction is used to check for a leap year.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ChkLeapYear	Check for Leap Year	FUN	(@)ChkLeapYear EN ENO In Out	Out:=ChkLeapYear(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Year	Input	Year	1970 to 2554	Year	1970
Out	Result	Output	TRUE: Leap year FALSE: Not leap year	Depends on data type.	---	---

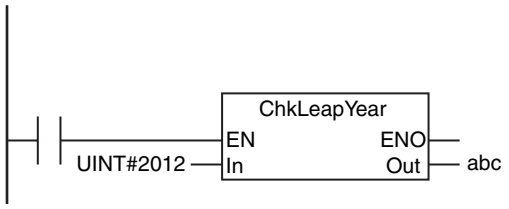
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In							OK													
Out	OK																			

Function

The ChkLeapYear instruction is used to check to see if year *In* is a leap year. If it is a leap year, the value of result *Out* is TRUE. If it is not a leap year, *Out* is FALSE.

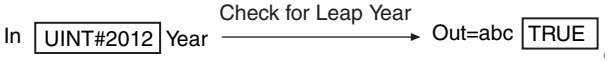
The following example is for when *In* is UINT#2012.

LD



ST

```
abc:=ChkLeapYear(UINT#2012);
```



Precautions for Correct Use

If the value of *In* exceeds the valid range, an error will not occur and the value of *Out* will be an illegal value.

GetDaysOfMonth

The GetDaysOfMonth instruction gets the number of days in the specified month.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetDaysOfMonth	Get Days in Month	FUN	(@)GetDaysOfMonth EN ENO Year Out Month	Out:=GetDaysOfMonth(Year, Month);

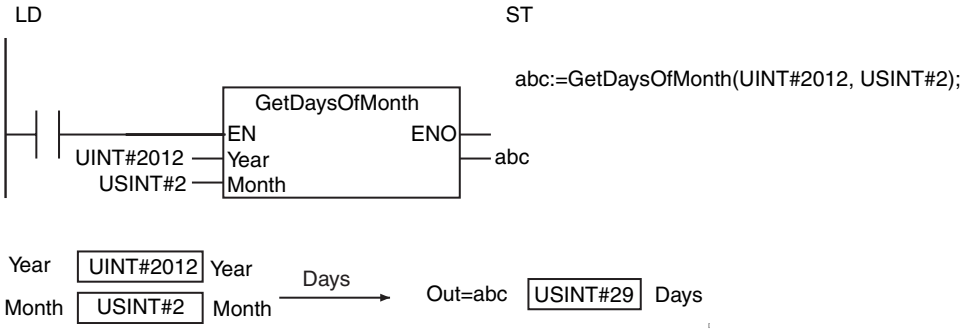
Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Year	Year	Input	Year	1970 to 2554	Year	1970
Month	Month		Month	1 to 12	Month	1
Out	Days	Output	Days	28 to 31	Days	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							OK													
Month						OK														
Out						OK														

Function

The GetDaysOfMonth instruction gets the number of days in month *Month* of year *Year*.
The following example is for when *Year* is UINT#2012 and *Month* is USINT#2.



Precautions for Correct Use

- If the value of *Year* exceeds the valid range, an error will not occur and the value of *Out* will be an illegal value.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Month* is outside of the valid range.

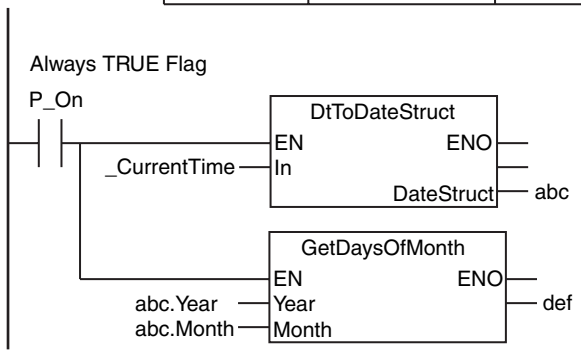
Sample Programming

This sample gets the number of days in the current month.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	abc	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	Date and time
	def	USINT	0	Days in current month

External Variables	Variable	Data type	Constant	Comment
	_CurrentTime	DATE_AND_TIME	☑	System Time of Day



ST

Internal Variables	Variable	Data type	Initial value	Comment
	abc	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	Date and time
	def	USINT	0	Days in current month

External Variables	Variable	Data type	Constant	Comment
	_CurrentTime	DATE_AND_TIME	☑	System Time of Day

```
DtToDateStruct(_CurrentTime, abc);
def:=GetDaysOfMonth(abc.Year, abc.Month);
```

DaysToMonth

The DaysToMonth instruction calculates the month based on the number of days from January 1.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DaysToMonth	Convert Days to Month	FUN	(@)DaysToMonth EN ENO Year Out Days	Out:=DaysToMonth(Year, Days);

Variables

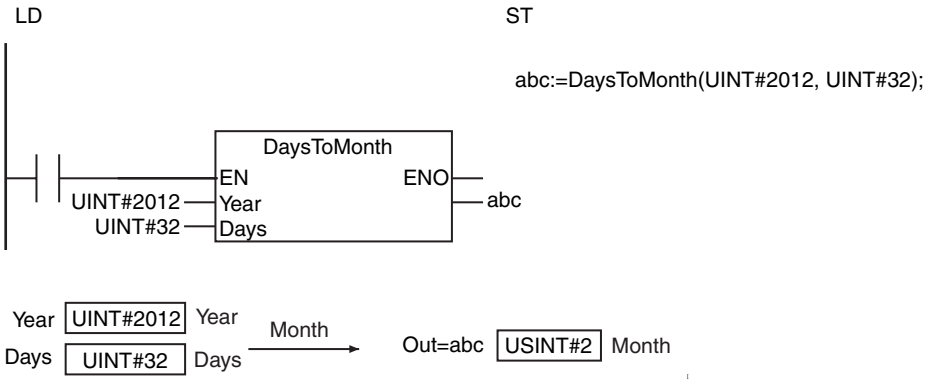
Name	Meaning	I/O	Description	Valid range	Unit	Default
Year	Year	Input	Year	1970 to 2554	Year	1970
Days	Days		Number of days from January 1	1 to 365 1 to 366 for a leap year	Days	1
Out	Month	Output	Month	1 to 12	Month	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							OK													
Days							OK													
Out						OK														

Function

The DaysToMonth instruction calculates the month based on the number of days in *Days* from January 1 in year *Year*.

The following example is for when *Year* is UINT#2012 and *Days* is UINT#32.

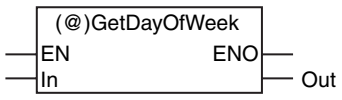


Precautions for Correct Use

- If the value of *Year* exceeds the valid range, an error will not occur and the value of *Out* will be an illegal value.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Days* is outside of the valid range.

GetDayOfWeek

The GetDayOfWeek instruction gets the day of the week for the specified year, month, and day of month.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetDayOfWeek	Get Day of Week	FUN		Out:=GetDayOfWeek(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Year, month, day	Input	Year, month, day	Depends on data type.	Year, month, day	*
Out	Day of the week	Output	Day of the week	_MON, _TUE, _WED, _THU, _FRI, _SAT, _SUN	Day of the week	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK		OK	
Out	Refer to *Function* for the enumerators for the enumerated type _eDAYOFWEEK.																			

Function

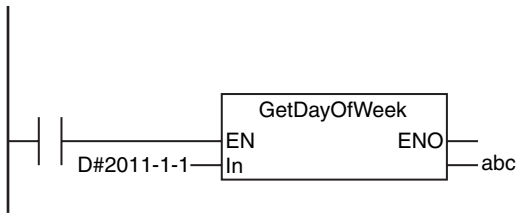
The GetDayOfWeek instruction gets the day of the week for the year, month, and day of month specified in *In*.

The data type of *Out* is enumerated type _eDAYOFWEEK. The meanings of the enumerators are as follows:

Enumerator	Meaning
_MON	Monday
_TUE	Tuesday
_WED	Wednesday
_THU	Thursday
_FRI	Friday
_SAT	Saturday
_SUN	Sunday

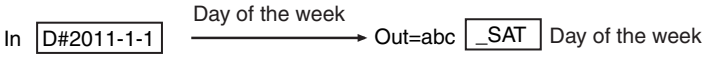
The following example is for when *In* is D#2011-1-1.

LD



ST

```
abc:=GetDayOfWeek(D#2011-1-1);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

GetWeekOfYear

The GetWeekOfYear instruction gets the week number for the specified year, month, and day of month.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetWeekOfYear	Get Week Number	FUN	(@)GetWeekOfYear EN ENO In Out	Out:=GetWeekOfYear(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Year, month, day	Input	Year, month, day	Depends on data type.	Year, month, day	*
Out	Week	Output	Week number	1 to 54	Week	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

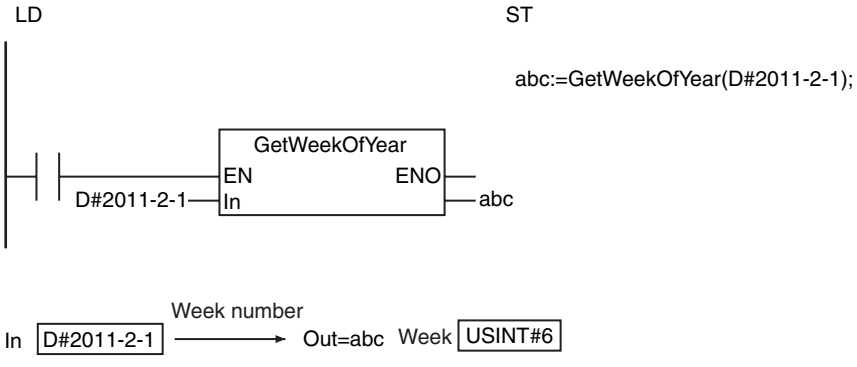
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK		OK	
Out						OK														

Function

The GetWeekOfYear instruction gets the week number for the year, month, and day of month specified in *In*. Weeks are counted from Monday to Sunday. The count is incremented when changing from Sunday to Monday.

January 1 is always in week 1. For example, if January 1 is a Thursday, January 1 to January 4 (Sunday) is week 1 and January 5 (Monday) to January 11 (Sunday) is week 2.

The following example is for when *In* is D#2011-2-1.

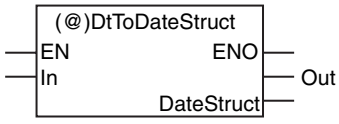


Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

DtToDateStruct

The DtToDateStruct instruction converts a date and time to the year, month, day, hour, minutes, seconds, and nanoseconds.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DtToDateStruct	Break Down Date and Time	FUN		Out:=DtToDateStruct(In, DateStruct);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Return value	Output	Always TRUE	TRUE only	---	---
DateStruct	Date and time		Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	---		

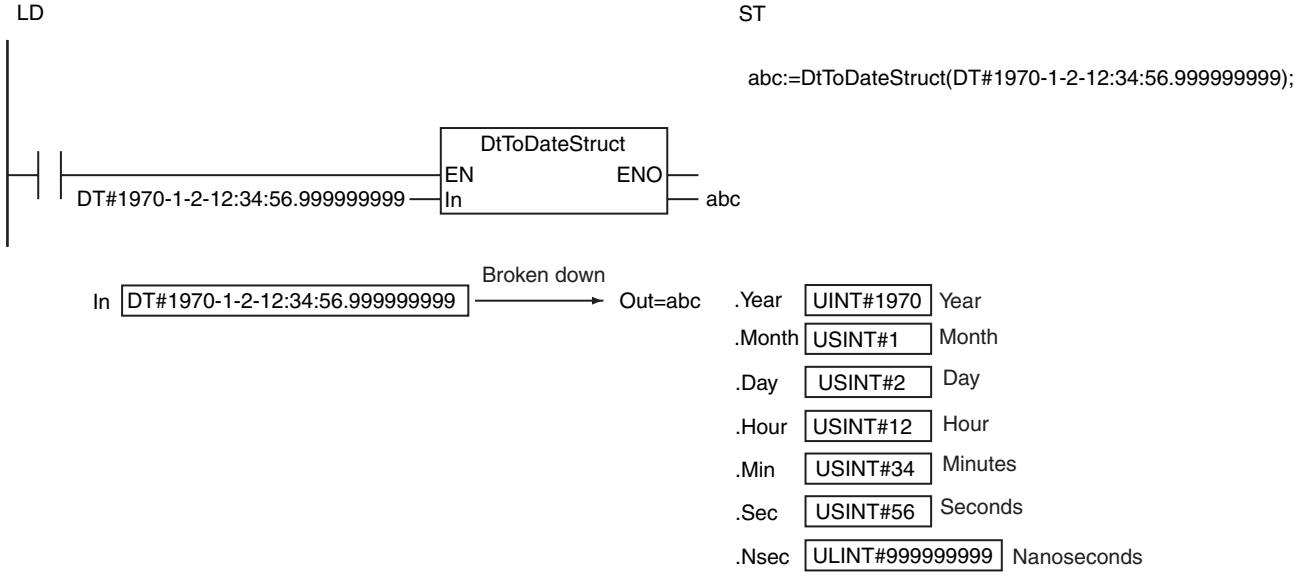
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out	OK																			
DateStruct	Refer to *Function* for details on the structure _sDT.																			

Function

The DtToDateStruct instruction converts the date and time in *In* to the year, month, day, hour, minutes, seconds, and nanoseconds. The data in the broken down date and time in *Out* is the structure _sDT. The meanings of the members are as follows:

Name	Meaning	Content	Data type	Valid range	Unit	Default
Out	Date and time	Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	_sDT	---	---	---
Year	Year	Year	UINT	1970 to 2554	Year	---
Month	Month	Month	USINT	1 to 12	Month	
Day	Day	Day	USINT	1 to 31	Day	
Hour	Hour	Hour	USINT	0 to 23	Hour	
Min	Minutes	Minutes	USINT	0 to 59	Minutes	
Sec	Seconds	Seconds	USINT	0 to 59	Seconds	
Nsec	Nanoseconds	Nanoseconds	ULINT	0 to 999999999	Nanoseconds	

The following example is for when *In* is DT#1970-1-2-12:34:56.999999999.

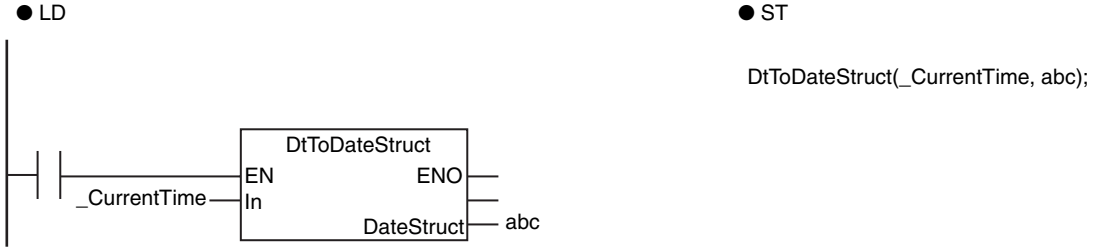


Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

- Use the DateStructToDt instruction (page 2-599) to join a year, month, day, hour, minutes, seconds, and nanoseconds into a date and time.
- The following example shows how to find the current time of day.



Precautions for Correct Use

Return value *Out* is not used when the instruction is used in ST.

DateStructToDt

The DateStructToDt instruction joins a year, month, day, hour, minutes, seconds, and nanoseconds into a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DateStructToDt	Join Time	FUN	(@)DateStructToDt EN ENO In Out	Out:=DateStructToDt(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	---	---	---
Out	Date and time	Output	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

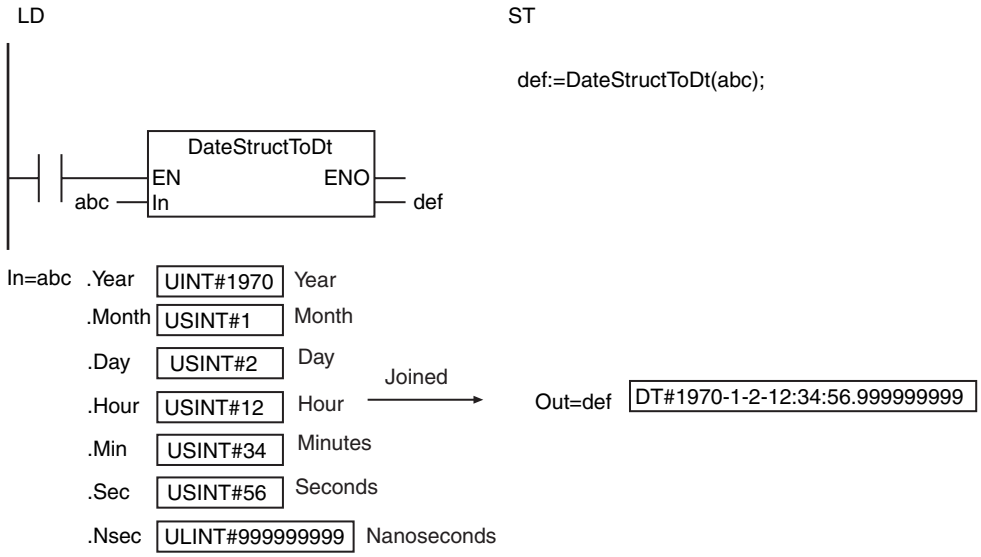
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	Refer to *Function* for details on the structure _sDT.																			
Out																			OK	

Function

The DateStructToDt instruction joins the year, month, day, hour, minutes, seconds, and nanoseconds in *In* into a date and time. The data type of *In* is structure _sDT. The meanings of the members are as follows:

Name		Meaning	Content	Data type	Valid range	Unit	Default
In		Date and time	Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	_sDT	---	---	---
	Year	Year	Year	UINT	1970 to 2554	Year	1970
	Month	Month	Month	USINT	1 to 12	Month	1
	Day	Day	Day	USINT	1 to 31	Day	
	Hour	Hour	Hour	USINT	0 to 23	Hour	0
	Min	Minutes	Minutes	USINT	0 to 59	Minutes	
	Sec	Seconds	Seconds	USINT	0 to 59	Seconds	
	Nsec	Nanoseconds	Nanoseconds	ULINT	0 to 999999999	Nanoseconds	

The following example is for the following values for the members of *In*: *Year* is UINT#1970, *Month* is USINT#1, *Day* is USINT#2, *Hour* is USINT#12, *Min* is USINT#34, *Sec* is USINT#56, and *Nsec* is ULINT#999999999.



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

Use the DtToDateStruct instruction (page 2-597) to break down a date and time into a year, month, day, hour, minutes, seconds, and nanoseconds.

Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of a member of *In* is outside of the valid range.
- The processing result exceeds the valid range of *Out*.

System Control Instructions

Instruction	Name	Page
TraceSamp	Data Trace Sampling	2-602
TraceTrig	Data Trace Trigger	2-605
GetTraceStatus	Read Data Trace Status	2-607
SetAlarm	Create User-defined Error	2-610
ResetAlarm	Reset User-defined Error	2-615
GetAlarm	Get User-defined Error Status	2-617
ResetPLCError	Reset PLC Controller Error	2-619
GetPLCError	Get PLC Controller Error Status	2-622
ResetCJBError	Reset CJ Bus Controller Error	2-624
GetCJBError	Get I/O Bus Error Status	2-626
GetEIPError	Get EtherNet/IP Error Status	2-628
ResetMCError	Reset Motion Control Error	2-630
GetMCError	Get Motion Control Error Status	2-634
ResetECError	Reset EtherCAT Controller Error	2-636
GetECError	Get EtherCAT Error Status	2-637
SetInfo	Create User-defined Information	2-639
ResetUnit	Restart Unit	2-641
GetNTPStatus	Read NTP Status	2-645

TraceSamp

The TraceSamp instruction performs sampling for a data trace.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TraceSamp	Data Trace Sampling	FUN	(@)TraceSamp EN ENO TraceNo Out Point	TraceSamp(TraceNo, Point);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TraceNo	Trace number	Input	Trace number	0 to 3	---	0
Point	Sampling point number		Sampling point number	Depends on data type.		
Out	Return value	Output	Always TRUE	TRUE only	---	---

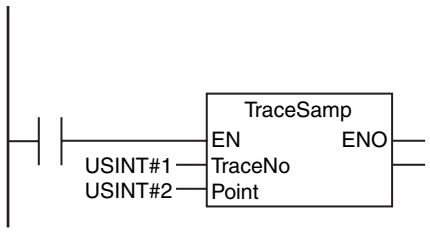
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						OK														
Point						OK														
Out	OK																			

Function

The TraceSamp instruction performs sampling for a data trace. The sampling settings are specified from the Sysmac Studio. The present values for all variables that are set to be sampled are read and stored with trace number *TraceNo* and sampling point number *Point* in trace memory. This instruction is executed only during execution of data tracing and only when the sampling timing is set to sampling instructions from the Sysmac Studio.

The following figure shows a programming example. Trace number 1 and sampling point number 2 are attached, and the present values of all variables to be sampled are stored in trace memory.

LD



ST

```
TraceSamp(USINT#1, USINT#2);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_TraceSta[0..3]	Trace Information	*	Trace information Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

* _sTRACE_STA[]

Additional Information

- Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details on data tracing.
- Tracing is used to sample the values of specified variables under specified conditions. The conditions are specified from the Sysmac Studio.
- This instruction can be located in more than one place in the user program. Programming can be written to sample according to specific conditions.
- *Point* can be suitably set so that you can see which sampled values on the Trace Window in the Sysmac Studio were returned by which TraceSamp instruction. *Point* will default to 0 if it is omitted.

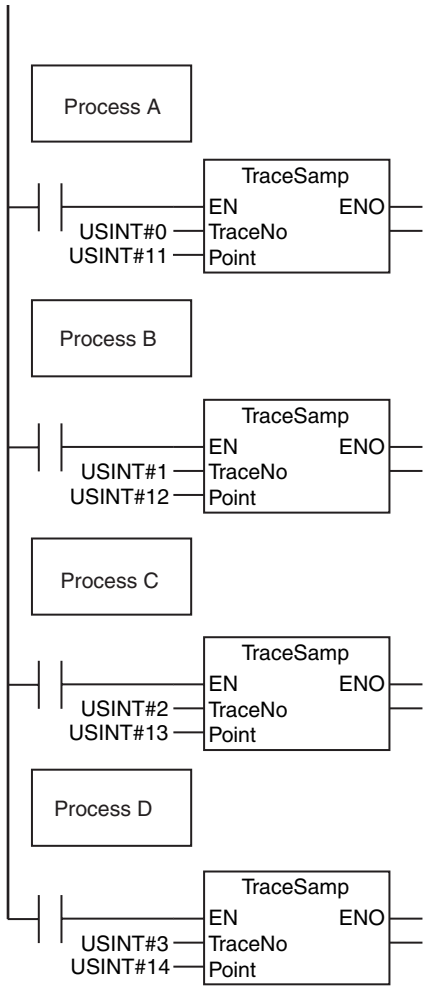
Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- In the following cases, nothing is done and the instruction ends normally.
  - Data tracing is stopped.
  - The sampling timing is not set to sampling instructions in the trace settings.
  - The value of *TraceNo* is not the trace number set from the Sysmac Studio.

Sample Programming

Here, sampling is performed at the end of each process A to D. The values of the variables are stored at each point.

LD

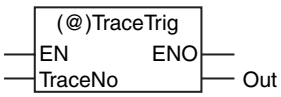


ST

```
Process A
TraceSamp(USINT#0, USINT#11);
Process B
TraceSamp(USINT#1, USINT#12);
Process C
TraceSamp(USINT#2, USINT#13);
Process D
TraceSamp(USINT#3, USINT#14);
```

TraceTrig

The TraceTrig instruction generates a trigger for data tracing.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TraceTrig	Data Trace Trigger	FUN		TraceTrig(TraceNo);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TraceNo	Trace number	Input	Trace number	0 to 3	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

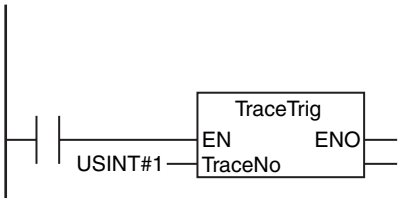
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						OK														
Out	OK																			

Function

The TraceTrig instruction generates a trigger for data tracing. It does not matter whether the trigger conditions that were set from the Sysmac Studio have been met. Sampling starts if data tracing is in progress for trace number *TraceNo* when the instruction is executed.

The following figure shows a programming example. Here, a data trace trigger is generated for trace number 1.

LD



ST

```
TraceTrig(USINT#1);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_TraceSta[0..3]	Trace Information	*	Trace information Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

* _sTRACE_STA[]

Additional Information

- Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details on data tracing.
- This instruction can be located in more than one place in the user program. Programming can be written to generate a trigger according to specific conditions.
- Programming can be written to generate triggers in ways that are not possible for normal trigger conditions settings, such as programming to generate a trigger based on a comparison of two variables.

Precautions for Correct Use

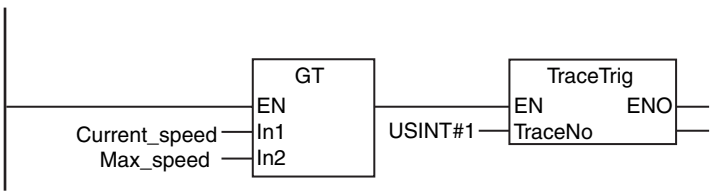
- Return value *Out* is not used when the instruction is used in ST.
- In the following cases, nothing is done and the instruction ends normally.
  - Data tracing is stopped.
  - The trigger condition has already been met.
  - The value of *TraceNo* is not the trace number set from the Sysmac Studio.
  - A continuous trace is specified as the trace type for the trace number that is specified with *TraceNo*.

Sample Programming

Here, a data trace trigger is generated to store the values of variables when the current speed exceeds the maximum speed. The TraceTrig instruction is executed when the value of *Current_speed* exceeds the value of *Max_speed*.

LD

Variable	Data type	Initial value	Comment
Current_speed	INT	0	Current speed
Max_speed	INT	20	Maximum speed



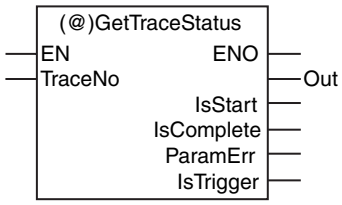
ST

Variable	Data type	Initial value	Comment
Current_speed	INT	0	Current speed
Max_speed	INT	20	Maximum speed

```
IF (Current_speed > Max_speed) THEN
    TraceTrig(USINT#1);
END_IF;
```

GetTraceStatus

The GetTraceStatus instruction reads the execution status of a data trace.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetTraceStatus	Read Data Trace Status	FUN		GetTraceStatus(TraceNo, IsStart, IsComplete, ParamErr, IsTrigger);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TraceNo	Trace number	Input	Trace number	0 to 3	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---
IsStart	Executing flag		TRUE: Data trace in progress. FALSE: Data trace not in progress.	Depends on data type.		
IsComplete	Completed flag		TRUE: Data trace was completed. FALSE: Data trace in progress or not executed.			
ParamErr	Parameter error flag		TRUE: Data trace setting error. FALSE: No data trace setting error.			
IsTrigger	Trigger flag		TRUE: Data trace trigger condition met. FALSE: Data trace trigger condition not met.			

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						OK														
Out	OK																			
IsStart	OK																			
IsComplete	OK																			
ParamErr	OK																			
IsTrigger	OK																			

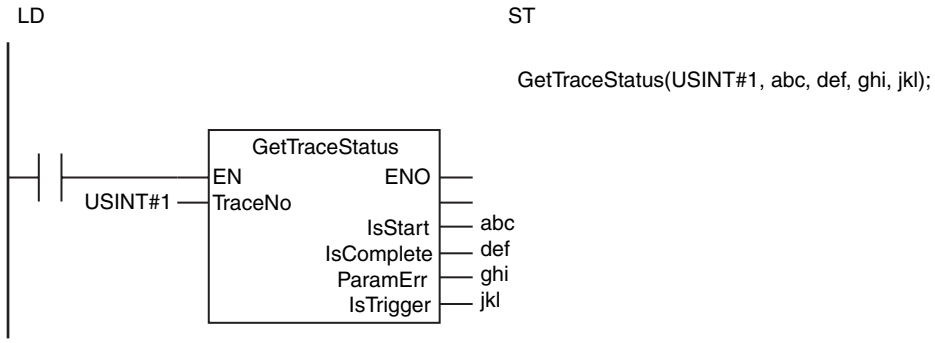
Function

The GetTraceStatus instruction reads the execution status of the data trace that is specified with trace number *TraceNo*. The status that is read is output to execution flag *IsStart*, completed flag *IsComplete*, parameter error flag *ParamErr*, and trigger flag *IsTrigger*.

The value of *ParamErr* changes to TRUE when one of the following errors is found in the trace settings.

- A variable that is specified in the trigger or sampling settings does not exist.
- Sampling is set to be performed on a specified task period, but the specified task does not exist.

The following figure shows a programming example. The GetTraceStatus instruction reads the execution status of the data trace with trace number 1.



Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_TraceSta[0..3]	Trace Information	*	Contains trace Information. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

* _sTRACE_STA[]

Additional Information

Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details on data tracing.

Precautions for Correct Use

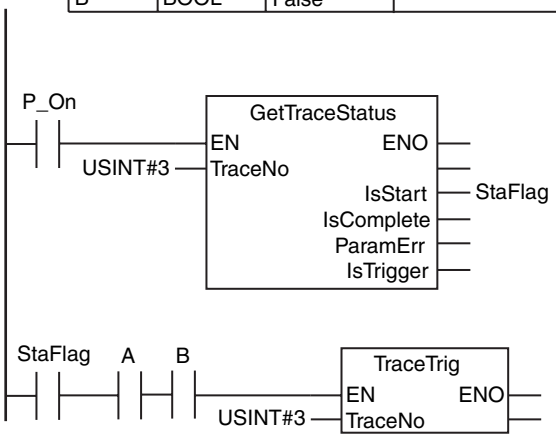
- Return value *Out* is not used when the instruction is used in ST.
- This instruction reads the contents of the *_PLC_TraceSta[]* system-defined variable. You cannot access this variable directly. Always use this instruction to read the contents of the variable.
- If *TraceNo* is not in the valid range, the values of *IsStart*, *IsComplete*, *ParamErr*, and *IsTrigger* are FALSE.

Sample Programming

In this sample, the GetTraceStatus instruction reads the execution status of the data trace with trace number 3. If the data trace is in progress, the TraceTrig instruction is executed to trigger data tracing.

LD

Variable	Data type	Initial value	Comment
StaFlag	BOOL	False	Trace execution status
A	BOOL	False	
B	BOOL	False	



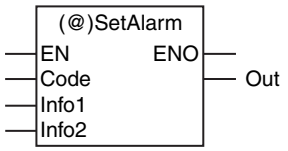
ST

Variable	Data type	Initial value	Comment
StaFlag	BOOL	False	Trace execution status
A	BOOL	False	
B	BOOL	False	

```
GetTraceStatus(TraceNo:=USINT#3, IsStart=>StaFlag);

IF ( (StaFlag=TRUE) AND (A=TRUE) AND (B=TRUE) ) THEN
    TraceTrig(TraceNo:=USINT#3);
END_IF;
```

SetAlarm

The SetAlarm instruction creates a user-defined error.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetAlarm	Create User-defined Error	FUN		SetAlarm(Code, Info1, Info2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Code	Event code	Input	Event code of user-defined error to generate	1 to 40000	---	1
Info1	Attached information 1		Values recorded in event log when the user-defined error is generated	Depends on data type.		*
Info2	Attached information 2					
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							OK													
Info1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Info2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Out	OK																			

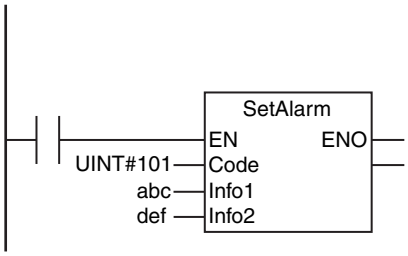
Function

The SetAlarm instruction generates the user-defined error that corresponds to event code *Code*. Event codes are defined in the event setting table on the Sysmac Studio. The time of occurrence, event name, event group, event code *Code*, event level, additional information *Info1*, additional information *Info2*, and detailed information are stored in the user event log area that corresponds to the level of the event code. The value for the time of occurrence is automatically obtained. The event name, event group, and detailed information that are set from the Sysmac Studio are recorded. The event level that corresponds to the event code is recorded. The event levels are given below. The smaller the event code is, the higher the event level is.

Event code	Classification: User fault level
1 to 5000	1
5001 to 10000	2
10001 to 15000	3
15001 to 20000	4
20001 to 25000	5
25001 to 30000	6
30001 to 35000	7
35001 to 40000	8

The following figure shows a programming example. A user-defined error with event code 101 is generated. The values of variables *abc* and *def* are stored as attached information.

LD



ST

```
SetAlarm(UINT#101, abc, def);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_AlarmFlag	Error Status of User-defined Errors	WORD	These flags indicate when user-defined errors are detected. Bit 0 to bit 7 indicate the status of user-defined error levels 1 to 8. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

Additional Information

You can specify either global variables or local variables for *Info1* and *Info2*.

Precautions for Correct Use

- Up to 32 user-defined errors can be generated in each of the eight event levels (for up to 256 user-defined errors total).

- If a user-defined error for the same event code already exists, the new error is not recorded in the event log.
- Always use variables for the input parameters that pass *Info1* and *Info2*. If you use a constant, a building error will occur.
- An error does not occur even if the value of *Code* is not set as a event code on the Sysmac Studio. If the event code is not registered, the event group and detailed information are not recorded in the user-defined event log. The value of *Code* is recorded for the event name.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE.
  - The value of *Code* is outside of the valid range.
  - An attempt was made to generate more than the maximum number of user-defined errors.

Sample Programming

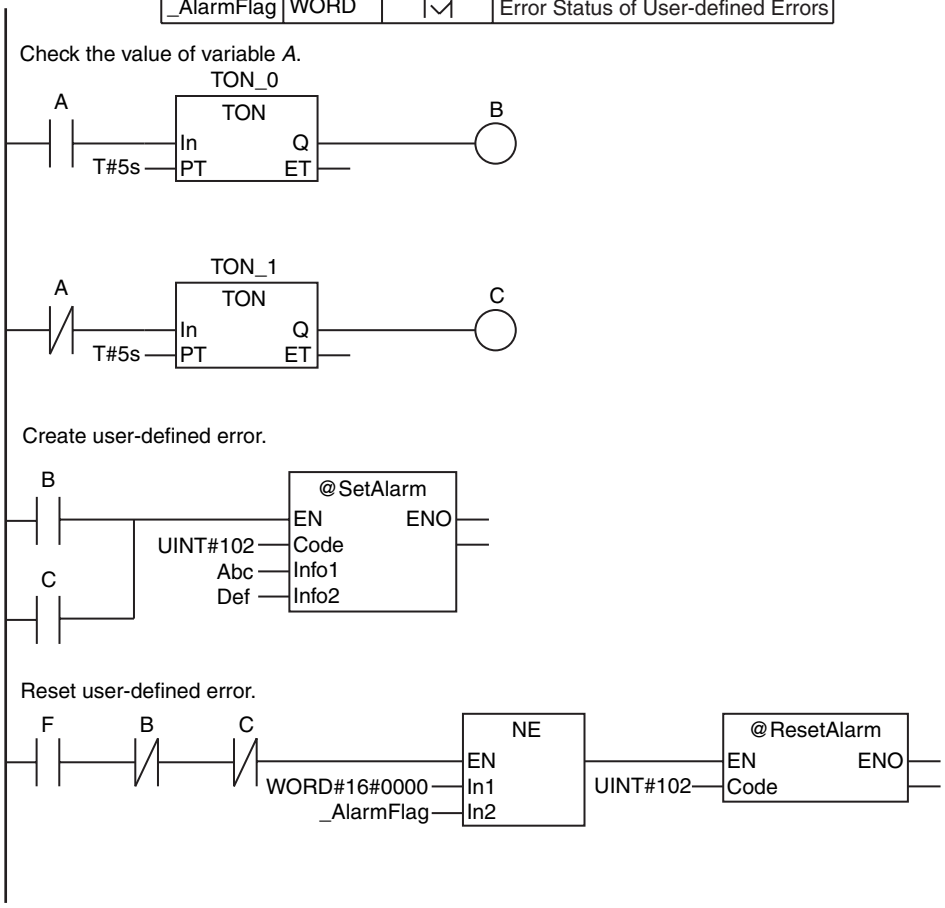
In this sample, the value of variable *A* changes between TRUE and FALSE every five seconds. The value of *A* is monitored. If it does not change for more than five seconds, a user-defined error with event code 102 is generated. UINT#123 and UINT#456 are given as the attached information.

When variable *F* changes to TRUE, the user-defined error is cleared.

LD

Internal Variables	Variable	Data type	Initial value
	A	BOOL	False
	B	BOOL	False
	C	BOOL	False
	F	BOOL	False
	Abc	UINT	123
	Def	UINT	456
	TON_instance0	TON	
	TON_instance1	TON	

External Variables	Variable	Data type	Constant	Comment
	_AlarmFlag	WORD	☑	Error Status of User-defined Errors



ST

Internal Variables	Variable	Data type	Initial value
	A	BOOL	False
	B	BOOL	False
	C	BOOL	False
	F	BOOL	False
	Abc	UINT	123
	Def	UINT	456
	TON_instance0	TON	
	TON_instance1	TON	

External Variables	Variable	Data type	Constant	Comment
	_AlarmFlag	WORD	☑	Error Status of User-defined Errors

```
// Check the value of variable A.
IF (A=TRUE) THEN
    TON_instance0(In:=TRUE, PT:=T#5s, Q=>B);
ELSE
    TON_instance0(In:=FALSE, Q=>B);
END_IF;

IF (A=FALSE) THEN
    TON_instance1(In:=TRUE, PT:=T#5s, Q=>C);
ELSE
    TON_instance1(In:=FALSE, Q=>C);
END_IF;

// Create user-defined error.
IF (B=TRUE) OR (C=TRUE) THEN
    SetAlarm(
        Code:=UINT102,
        Info1 :=Abc,
        info2 :=Def);
END_IF;

// Reset user-defined error.
IF (F=TRUE)  & (B=FALSE) & (C=FALSE) & (_AlarmFlag<>WORD#16#0000) THEN
    ResetAlarm(Code:=UINT#102);
END_IF;
```

ResetAlarm

The ResetAlarm instruction resets a user-defined error.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetAlarm	Reset User-defined Error	FUN	(@)ResetAlarm EN ENO Code Out	ResetAlarm(Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Code	Event code	Input	Event code of user-defined error to reset 16#0:Reset all application errors.	Depends on data type.	---	1
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							OK													
Out	OK																			

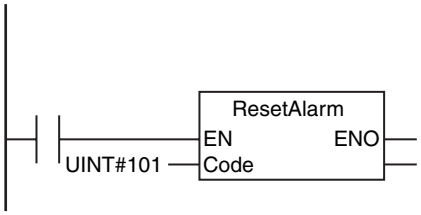
Function

The ResetAlarm instruction resets the user-defined error specified by event code *Code*. An event is then recorded in the user-defined event log to show that a specific user-defined error was reset. The event code for this event is 65533 and the level is user information.

If the value of *Code* is 0, all current user-defined errors are reset. An event is then recorded in the user-defined event log to show that all user-defined errors were reset. The event code for this event is 65534 and the level is user information.

The following figure shows a programming example. A user-defined error for event code 101 is reset.

LD



ST

```
ResetAlarm(UINT#101);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_AlarmFlag	Error Status of User-defined Errors	WORD	These flags indicate when user-defined errors are detected. Bit 0 to bit 7 indicate the status of user-defined error levels 1 to 8. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

Precautions for Correct Use

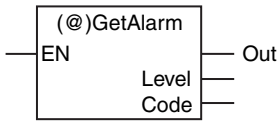
- An error does not occur if the user-defined error specified by *Code* has not occurred.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE.
  - The value of *Code* is outside of the valid range.

Sample Programming

Refer to the sample programming that is provided for the SetAlarm instruction (page 2-610).

GetAlarm

The GetAlarm instruction gets the highest event level (of user-defined error levels 1 to 8) and the highest level event code of the current user-defined errors.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetAlarm	Get User-defined Error Status	FUN		Out:=GetAlarm(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: User-defined error exists. FALSE: No user-defined error	Depends on data type.	---	---
Level	Highest event level		Highest event level of all current user-defined errors 0: No user-defined error 1 to 8: Event level	0 to 8		
Code	Highest level event code		Highest level event code of all current user-defined errors 0: No user-defined error 1 to 40000: Event level	0 to 40000		

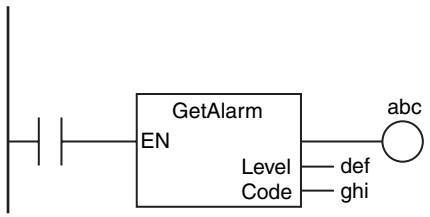
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code							OK													

Function

The GetAlarm instruction gets the highest event level and the highest level event code of the current user-defined errors and outputs them to *Level* and *Code*. If there are currently no user-defined errors, the value of error flag *Out* is FALSE. If there is more than one use-defined error at the highest event level, the value of *Code* is the event code for the user-defined error that occurred first.

The following figure shows a programming example.

LD



ST

```
abc:=GetAlarm(def, ghi);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_AlarmFlag	Error Status of User-defined Errors	WORD	These flags indicate when user-defined errors are detected. Bit 0 to bit 7 indicate the status of user-defined error levels 1 to 8. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

Precautions for Correct Use

If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

ResetPLCError

The ResetPLCError instruction resets errors in the PLC Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetPLCError	Reset PLC Controller Error	FB	ResetPLCError_instance ResetPLCError Execute Done Busy Error ErrorID	ResetPLCError(Execute, Done, Busy, Error, ErrorID);

Variables

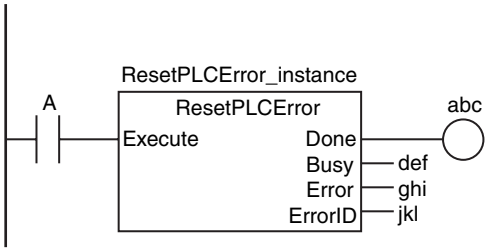
Only common variables are used.

Function

The ResetPLCError instruction resets errors in the PLC Function Module.

The following figure shows a programming example.

LD



ST

```
ResetPLCError_instance(A, abc, def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_ErrSta	Error Status of PLC Function Module	WORD	Contains the error status of the PLC Function Module, Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

Precautions for Correct Use

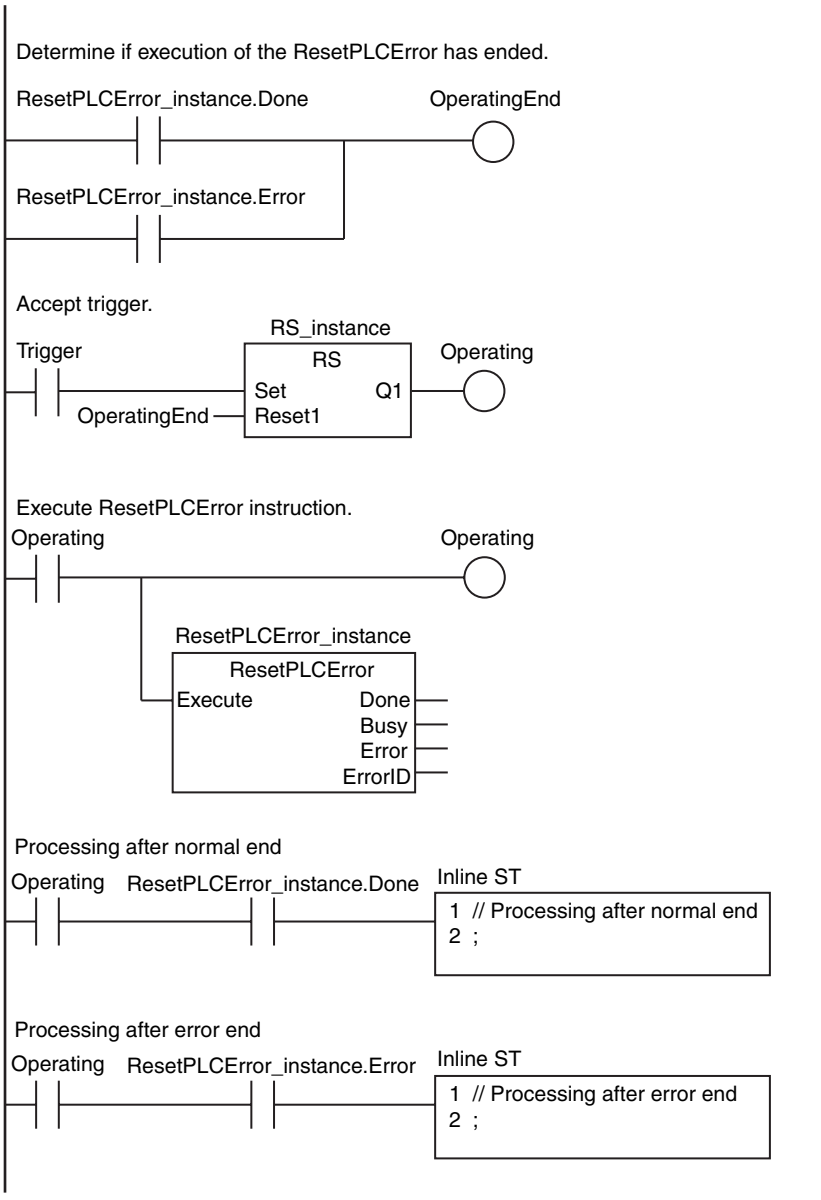
The error may not be reset immediately after you execute this instruction. Use the GetPLCError instruction to confirm that the errors were reset.

Sample Programming

The ResetPLCError instruction is executed when the value of *Trigger* changes to TRUE. Normal end processing is performed if execution of the ResetPLCError instruction ends normally (i.e., if the value of *Done* is TRUE). Error end processing is performed if execution ends in an error (i.e., if the value of *Error* is TRUE).

LD

Variable	Data type	Initial value	Comment
OperatingEnd	BOOL	False	Processing completed
Trigger	BOOL	False	Execution condition
Operating	BOOL	False	Processing
RS_instance	RS		
ResetPLCError_instance	ResetPLCError		



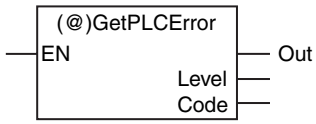
ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	False	Execution condition
LastTrigger	BOOL	False	Value of *Trigger* from previous task period
OperatingStart	BOOL	False	Processing started
Operating	BOOL	False	Processing
ResetPLCError_instance	ResetPLCError		

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize ResetPLCError_instance.
IF (OperatingStart=TRUE) THEN
    ResetPLCError_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

// Execute ResetPLCError instruction.
IF (Operating=TRUE) THEN
    ResetPLCError_instance(Execute:=TRUE);

    IF (ResetPLCError_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;

    IF (ResetPLCError_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;
```

GetPLCError

The GetPLCError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the PLC Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetPLCError	Get PLC Controller Error Status	FUN		Out:=GetPLCError(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Controller error exists. FALSE: No Controller error	Depends on data type.		
Level	Highest event level		Highest level status of all current Controller errors in the PLC Function Module 0: No Controller error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current Controller errors in the PLC Function Module 16#0000_0000: No Controller error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

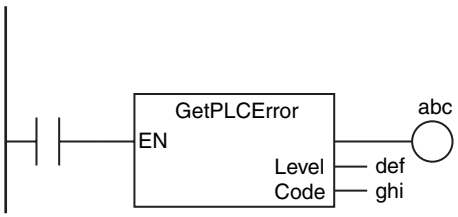
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetPLCError instruction gets the highest level status and the highest level event code of the current Controller errors in the PLC Function Module and outputs them to *Level* and *Code*.
If there are currently no Controller errors, the value of error flag *Out* is FALSE.
If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.

LD



ST

```
abc:=GetPLCError(def, ghi);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_ErrSta	Error Status of PLC Function Module	WORD	Contains the error status of the PLC Function Module. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

ResetCJBError

The ResetCJBError instruction resets a Controller error in the I/O bus.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetCJBError	Reset I/O Bus Error	FB	ResetCJBError_instance ResetCJBError EN UnitNo Done Busy Error ErrorID	ResetCJBError_instance(Execute, UnitNo, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitNo	Unit number	Input	Unit number for which to reset errors	_CBU_No00 to _CBU_No15, _SIO_No00 to _SIO_No95 _UNIT_ALL	---	_UNIT_ALL

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitNo	Refer to *Function* for the enumerators of the enumerated type _eUnitNo.																			

Function

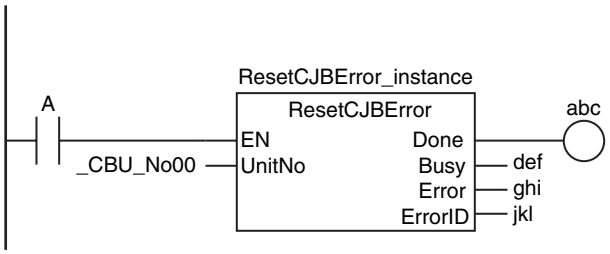
The ResetCJBError instruction resets a Controller error in the I/O bus. If the Unit specified by unit number *UnitNo* is a CJ-series Special Unit, the Unit is restarted.

The data type of *UnitNo* is enumerated type _eUnitNo. The meanings of the enumerators are as follows:

Enumerators	Meaning
_CBU_No00 to _CBU_No15	Unit number of CPU Bus Unit, 00 to 15
_SIO_No00 to _SIO_No95	Unit number of Special I/O Unit, 00 to 95
_UNIT_ALL	All Units

The following example is for when *UnitNo* is _CBU_No00. The Controller error on the I/O bus is reset and the CPU Bus Unit with unit number 0 is restarted.

LD



ST

```
ResetCJBError_instance(A, _CBU_No00,
                       abc, def, ghi, jkl);
```

Related System-defined Variables

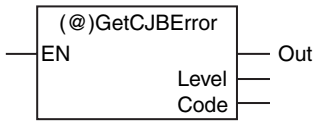
Name	Meaning	Data type	Description
_CJB_ErrSta	I/O Bus Error Status	WORD	Contains the error status of the I/O bus. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

Precautions for Correct Use

- The error may not be reset immediately after you execute this instruction. Use the GetCJBError instruction to confirm that the errors were reset.
- An error occurs in the following cases. *ENO* will be FALSE.
  - The value of *UnitNo* is outside of the valid range.
  - The Unit specified by *UnitNo* does not exist.

GetCJBError

The GetCJBError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the I/O bus.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetCJBError	Get I/O Bus Error Status	FUN		Out:=GetCJBError(Level, Code);

Variables

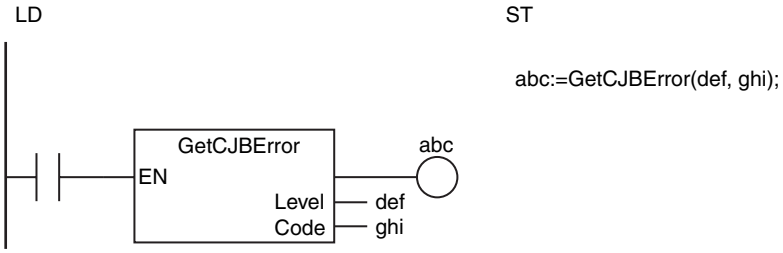
Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Controller error exists. FALSE: No Controller error	Depends on data type.	---	---
Level	Highest event level		Highest level status of all current Controller errors in the I/O bus 0: No Controller error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current Controller errors in the I/O bus 16#0000_0000: No Controller error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetCJBError instruction gets the highest level status and the highest level event code of the current Controller errors in the I/O bus and outputs them to *Level* and *Code*. If there are currently no Controller errors, the value of error flag *Out* is FALSE. If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.

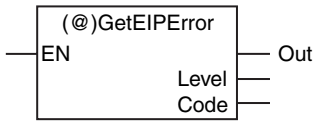


Related System-defined Variables

Name	Meaning	Data type	Description
_CJB_ErrSta	I/O Bus Error Status	WORD	Contains the error status of the I/O bus. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

GetEIPError

The GetEIPError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the EtherNet/IP Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetEIPError	Get EtherNet/IP Error Status	FUN		Out:=GetEIPError(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Controller error exists. FALSE: No Controller error	Depends on data type.	---	---
Level	Highest event level		Highest level status of all current Controller errors in the EtherNet/IP Function Module 0: No Controller error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current Controller errors in the EtherNet/IP Function Module 16#0000_0000: No Controller error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

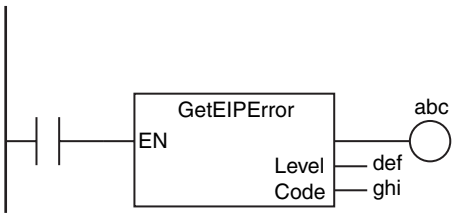
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetEIPError instruction gets the highest level status and the highest level event code of the current Controller errors in the EtherNet/IP Function Module and outputs them to *Level* and *Code*. If there are currently no Controller errors, the value of error flag *Out* is FALSE. If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.

LD



ST

```
abc:=GetEIPError(def, ghi);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_ErrSta	Error Status of EtherNet/IP Function Module	WORD	Contains the error status of the EtherNet/IP Function Module. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

ResetMCError

The ResetMCError instruction resets Controller errors in the Motion Control Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetMCError	Reset Motion Control Error	FB	ResetMCError_instance ResetMCError Execute Done Busy Failure Error ErrorID	ResetMCError_instance(Execute, Done, Busy, Failure Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Failure	Failure end	Output	TRUE: The errors were not reset. FALSE: The errors were reset normally.	Depends on data type.	---	---

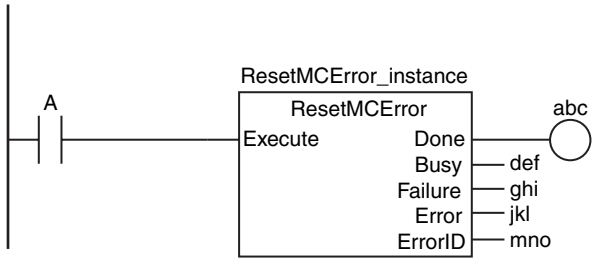
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Failure	OK																			

Function

The ResetMCError instruction resets a Controller error in the Motion Control Function Module. If the errors are not reset, the value of *Failure* changes to TRUE.

The following figure shows a programming example.

LD



ST

```
ResetMCError_instance(A, abc, def,
                      ghi, jkl, mno);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_MC_ErrSta	Motion Control Error Status	WORD	Contains the error status of the Motion Control Function Module. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

Precautions for Correct Use

- The error may not be reset immediately after you execute this instruction. Use the GetMCError instruction to confirm that the errors were reset.
- If you attempt to execute this instruction during an MC Test Run, the value of *BUSY* remains TRUE and the instruction is not executed.

Sample Programming

This sample detects Controller errors in the EtherCAT Master Function Module and Motion Control Function Module. If errors are detected, they are reset. The processing procedure is as follows:

1 The GetECError instruction is executed to detect any Controller errors in the EtherCAT Master Function Module.

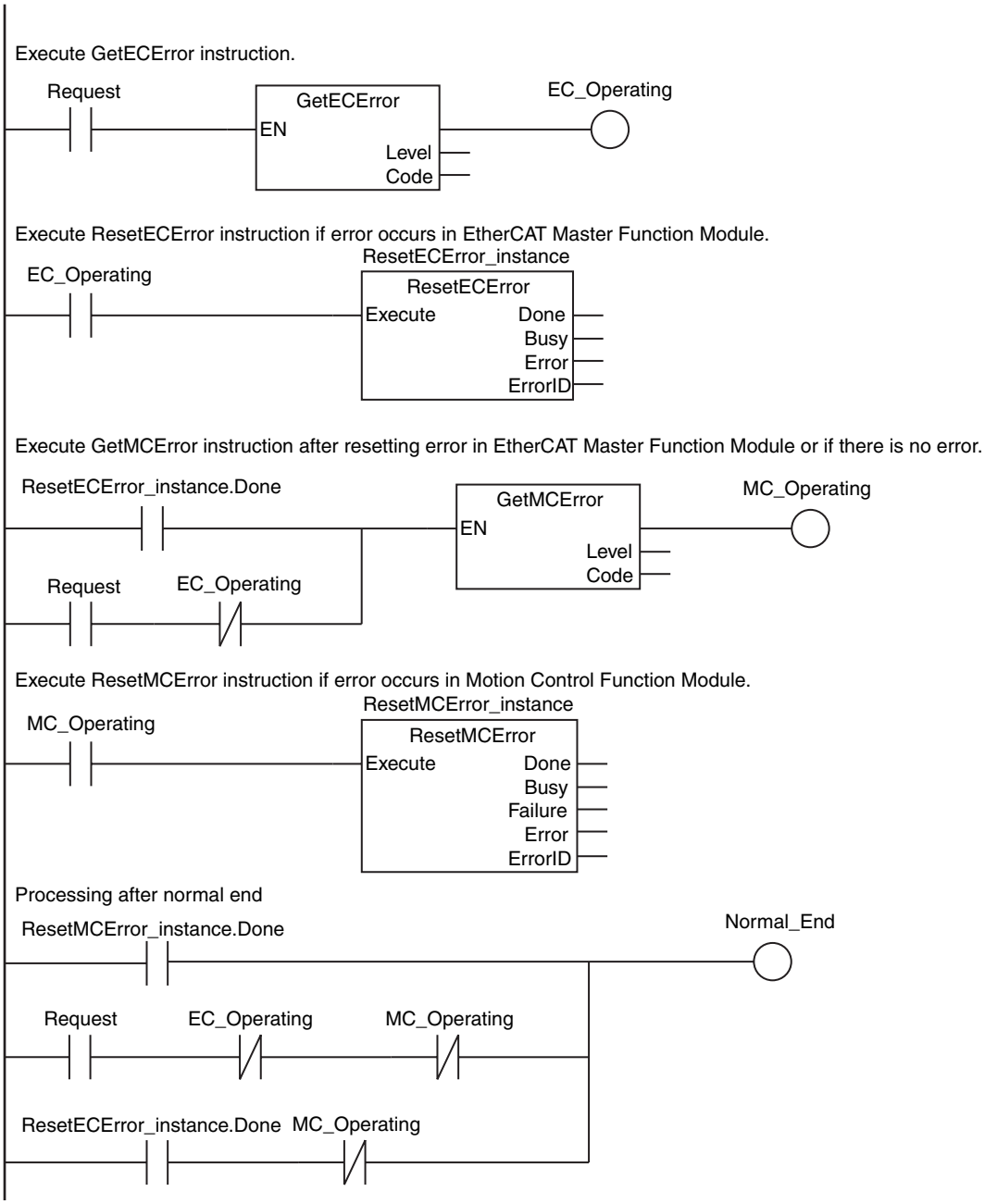
2 If errors are detected, they are reset with the ResetECError instruction.

3 The GetMCError instruction is executed to detect any Controller errors in the Motion Control Function Module.

4 If errors are detected, they are reset with the ResetMCError instruction.

LD

Variable	Data type	Initial value	Comment
Request	BOOL	False	Error detection reset request
EC_Operating	BOOL	False	Resetting error in EtherCAT Master Function Module
MC_Operating	BOOL	False	Resetting error in Motion Control Function Module
Normal_End	BOOL	False	Normal end
ResetECError_instance	ResetECError		
ResetMCError_instance	ResetMCError		



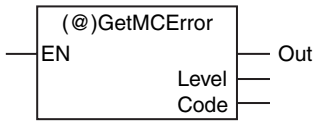
ST

Variable	Data type	Initial value	Comment
Request	BOOL	False	Error detection reset request
EC_Error	BOOL	False	Error in EtherCAT Master Function Module
EC_Stage	INT	0	Resetting error in EtherCAT Master Function Module
MC_Error	BOOL	False	Error in Motion Control Function Module
MC_Stage	INT	0	Error reset in Motion Control Function Module
ResetECError_instance	ResetECError		
ResetMCError_instance	ResetMCError		

```
IF (Request=TRUE) THEN     // Determine error resetting requests.
    EC_Error:=GetECError();  // Detect Controller errors in EtherCAT Master Function Module.
    MC_Error:=GetMCError(); // Detect Controller errors in Motion Control Function Module.

    IF (EC_Error=TRUE) THEN     // Controller error in EtherCAT Master Function Module.
        CASE EC_Stage OF
        0 :                 // Initialize
            ResetECError_instance(Execute:=FALSE);
            EC_Stage:=INT#1;
        1 :                 // Resetting Controller error in EtherCAT Master Function Module.
            ResetECError_instance(Execute:=TRUE);
            IF (ResetECError_instance.Done=TRUE) THEN
                EC_Stage:=INT#99;         // Normal end
            END_IF;
            IF (ResetECError_instance.Error=TRUE) THEN
                EC_Stage:=INT#98;         // Error end
            END_IF;
        99 :                // Processing after normal end
            EC_Stage:=INT#0;
        98 :                // Processing after error end.
            EC_Stage:=INT#0;
        END_CASE;
    END_IF;

    IF (MC_Error=TRUE) THEN    // Controller error in Motion Control Function Module.
        CASE MC_Stage OF
        0 :                 // Initialize
            ResetMCError_instance(Execute:=FALSE);
            MC_Stage:=INT#1;
        1 :                 // Resetting Controller error in Motion Control Function Module.
            IF (EC_Error=FALSE) THEN                        // Recover operation for all slaves.
                ResetMCError_instance(Execute:=TRUE);
                IF (ResetMCError_instance.Done=TRUE) THEN
                    MC_Stage:=INT#99;   // Normal end
                END_IF;
                IF ( (ResetMCError_instance.Error=TRUE) OR (ResetMCError_instance.Failure=TRUE) ) THEN
                    MC_Stage:=INT#98;   // Error end
                END_IF;
            END_IF;
        99 :                // Processing after normal end
            MC_Stage:=INT#0;
        98 :                // Processing after error end.
            MC_Stage:=INT#0;
        END_CASE;
    END_IF;
END_IF;
```

GetMCError

The GetMCError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the Motion Control Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetMCError	Get Motion Control Error Status	FUN		Out:=GetMCError(Level, Code);

Variables

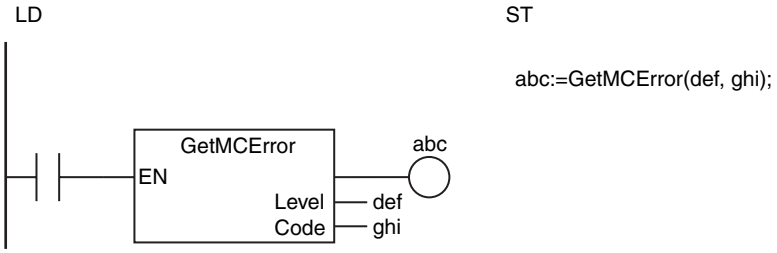
Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Controller error exists. FALSE: No Controller error	Depends on data type.	---	---
Level	Highest event level		Highest level status of all current Controller errors in the Motion Control Function Module 0: No Controller error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current Controller errors in the Motion Control Function Module 16#0000_0000: No Controller error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetMCError instruction gets the highest level status and the highest level event code of the current Controller errors in the Motion Control Function Module and outputs them to *Level* and *Code*. If there are currently no Controller errors, the value of error flag *Out* is FALSE. If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.



Related System-defined Variables

Name	Meaning	Data type	Description
_MC_ErrSta	Error Status of Motion Control Function Module	WORD	Contains the error status of the Motion Control Function Module. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

Sample Programming

Refer to the sample programming that is provided for the ResetMCError instruction (page 2-630).

ResetECError

The ResetECError instruction resets a Controller error in the EtherCAT Master Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetECError	Reset EtherCAT Controller Error	FB	ResetECError_instance ResetECError Execute Done Busy Error ErrorID	ResetECError_instance(Execute, Done, Busy, Error, ErrorID);

Variables

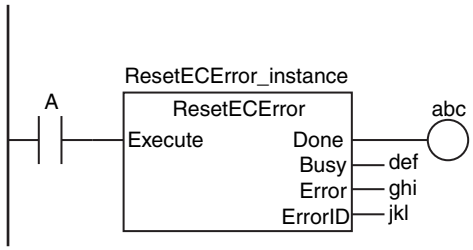
Only common variables are used.

Function

The ResetECError instruction resets Controller errors in the EtherCAT Master Function Module.

The following figure shows a programming example.

LD



ST

```
ResetECError_instance(A, abc, def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_ErrSta	Built-in EtherCAT Error	WORD	Contains a summary of the errors in the EtherCAT Master Function Module. Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details.

Precautions for Correct Use

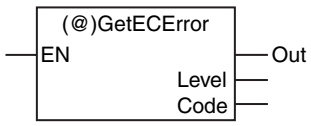
- The error may not be reset immediately after you execute this instruction. Use the GetECError instruction to confirm that the errors were reset.
- An error occurs in the following case. *Error* will change to TRUE.
  - This instruction is executed again while processing to clear a Controller error from the EtherCAT Master Function Module is in progress.

Sample Programming

Refer to the sample programming that is provided for the ResetMCError instruction (page 2-630).

GetECError

The GetECError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current communications port errors or master errors in the EtherCAT Master Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetECError	Get EtherCAT Error Status	FUN		Out:=GetECError(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Communications port error or master error exists. FALSE: No communications port error or master error.	Depends on data type.	---	---
Level	Highest event level		Highest level status of all current communications port errors and master errors in the EtherCAT Function Module 0: No error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current communications port errors and master errors in the EtherCAT Function Module 16#0000_0000: No error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

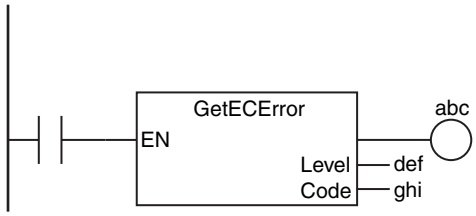
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetECError instruction gets the highest level status and highest level event code of the current communications port errors or master errors in the EtherCAT Master Function Module and outputs them to *Level* and *Code*. If there are currently no communications port errors or master errors, the value of error flag *Out* is FALSE. If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.

LD



ST

```
abc:=GetECError(def, ghi);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_ErrSta	Built-in EtherCAT Error	WORD	Contains a summary of the errors in the EtherCAT Master Function Module.*2
_EC_PortErr*1	Communications Port Error	WORD	Contains a summary of the EtherCAT master communications port errors.*2
_EC_MstrErr*1	Master Error	WORD	Contains a summary of the EtherCAT master errors and the slave errors detected by the EtherCAT master.*2
_EC_SlavErr	Slave Error	WORD	Contains a summary of the overall EtherCAT slave error status.*2

*1 The GetECError instruction gets the errors that are shown by _EC_PortErr (Communications Port Error) and _EC_MstrErr (Master Error).

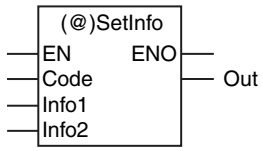
*2 Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details.

Sample Programming

Refer to the sample programming that is provided for the ResetMCError instruction (page 2-630).

SetInfo

The SetInfo instruction creates user-defined information.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetInfo	Create User-defined Information	FUN		SetInfo(Code, Info1, Info2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Code	Event code	Input	Event code of user-defined information to generate	40001 to 60000	---	40001
Info1	Attached information 1		Values recorded in event log when the user-defined information is generated	Depends on data type.		*
Info2	Attached information 2					
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

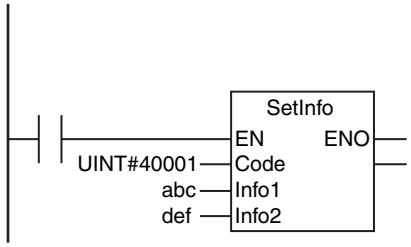
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							OK													
Info1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Info2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Out	OK																			

Function

The SetInfo instruction generates the user-defined information specified by event code *Code*. The time of occurrence, event code *Code*, event level, attached information *Info1*, and attached information *Info2* are stored in the user event log area that corresponds to the level of the event code.

The following figure shows a programming example. User-defined information for event code 40001 is generated. The values of variables *abc* and *def* are stored as attached information.

LD



ST

```
SetInfo(UINT#40001, abc, def);
```

Precautions for Correct Use

- Always use variables for the input parameters that are passed to *Info1* and *Info2*. If the attached information is not used, specify a dummy variable.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE.
  - The value of *Code* is outside of the valid range.

ResetUnit

The ResetUnit instruction restarts a CPU Bus Unit or Special I/O Unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetUnit	Restart Unit	FB	ResetUnit_instance ResetUnit Execute / Done UnitNo / Busy Error ErrorID	ResetUnit_instance(Execute, UnitNo, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitNo	Unit number	Input	Unit number of Unit to restart	_CBU_No00 to _CBU_No15, _SIO_No00 to _SIO_No95	---	_CBU_No00

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitNo	Refer to *Function* for the enumerators of enumeration type _eUnitNo.																			

Function

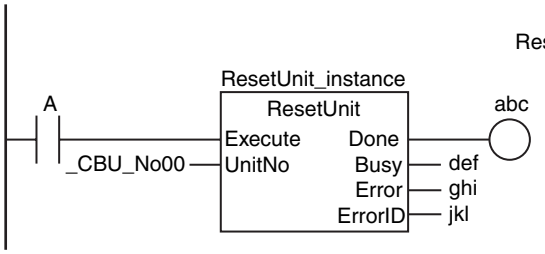
The ResetUnit instruction restarts the CPU Bus Unit or Special I/O Unit with unit number *UnitNo*.

The data type of *UnitNo* is enumerated type _eUnitNo. The meanings of the enumerators are as follows:

Enumerators	Meaning
_CBU_No00 to _CBU_No15	Unit number of CPU Bus Unit, 00 to 15
_SIO_No00 to _SIO_No95	Unit number of Special I/O Unit, 00 to 95

The following example is for when *UnitNo* is _CBU_No00. CPU Bus Unit with unit number 0 is restarted.

LD



ST

```
ResetUnit_instance(A, _CBU_No00, abc, def, ghi, jkl);
```

Precautions for Correct Use

- This instruction will not end in an error even if restart processing is in progress for the Unit specified by *UnitNo.* The value of *Busy* remains at TRUE and the value of *Done* changes to TRUE when restart processing is finished. Restart requests are not queued.
- The Unit is restarted if the value of *Execute* is TRUE when operation starts.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *UnitNo* is outside of the valid range.
  - The Unit specified with *UnitNo* does not exist.
  - Restart processing failed.

Sample Programming

When the value of *Trigger* changes to TRUE, the baud rate of serial port 1 on the Serial Communications Unit with a unit number of 0 is set to 38,400 bps and the Unit is restarted.

Definitions of Global Variables

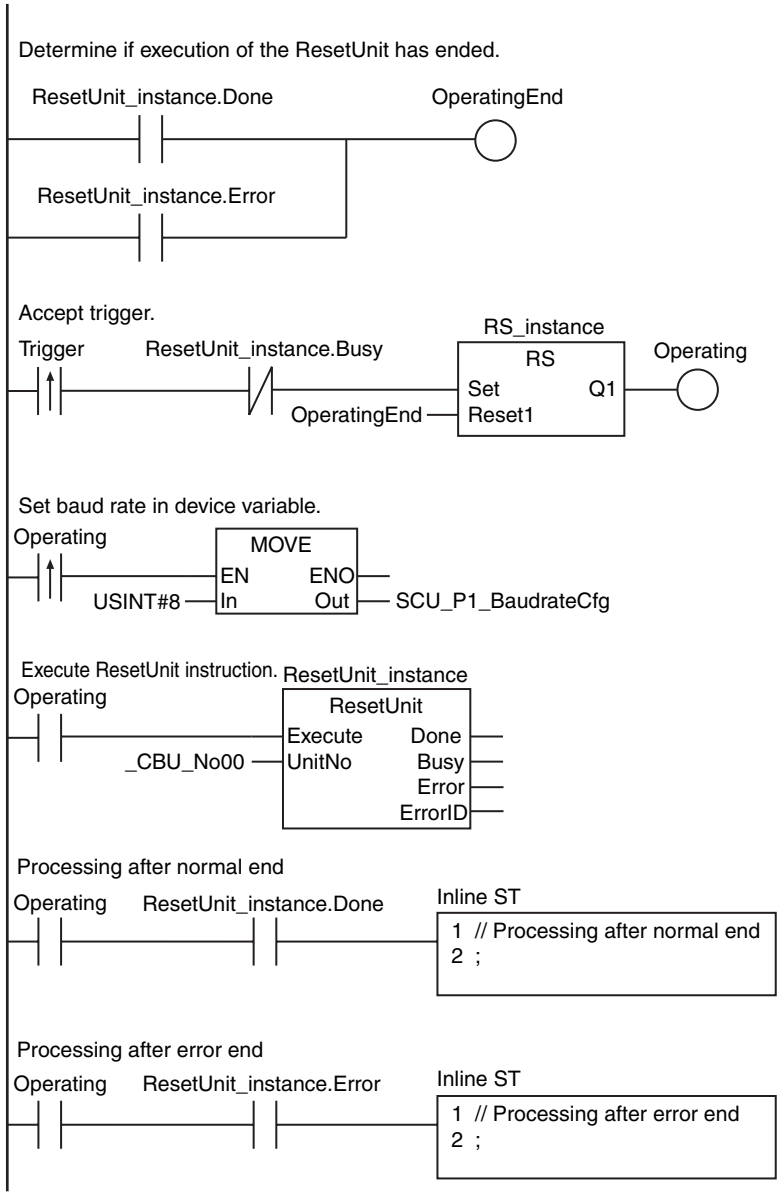
Global Variables

Name	Data type	Initial value	AT specification	Retain	Comment
SCU_P1_BaudrateCfg	USINT	0	IOBus://rack#0/slot#0 /P1_BaudrateCfg	☑	Baud rate

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	RS_instance	RS		
	ResetUnit_instance	ResetUnit		

External Variables	Variable	Data type	Comment
	SCU_P1_BaudrateCfg	USINT	Baud rate



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LaseTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started
	Operating	BOOL	False	Processing
	ResetUnit_instance	ResetUnit		

External Variables	Variable	Data type	Comment
	SCU_P1_BaudrateCfg	USINT	Baud rate

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (ResetUnit_instance.Busy=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize ResetUnit_instance and set baud rate in device variable.
IF (OperatingStart=TRUE) THEN
    ResetUnit_instance(Execute:=FALSE);
    SCU_P1_BaudrateCfg:=USINT#8;
    OperatingStart:=FALSE;
END_IF;

// Execute ResetUnit instruction.
IF (Operating=TRUE) THEN
    ResetUnit_instance(
        Execute:=TRUE,          // Execution condition
        UnitNo  :=_CBU_No00);  // Unit number

    IF (ResetUnit_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;

    IF (ResetUnit_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;
```

GetNTPStatus

The GetNTPStatus instruction reads the NTP status.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetNTPStatus	Read NTP Status	FUN	(@)GetNTPStatus EN ENO — Out ExecTime ExecNormal	GetNTPStatus(ExecTime, ExecNormal);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Return value	Output	Always TRUE	TRUE only	---	---
ExecTime	NTP last normal operation time		NTP last normal operation time	Depends on data type.	Year, month, day, hour, minutes, seconds	---
Exec Normal	NTP normal end flag		TRUE: Normal end FALSE: Error end		---	

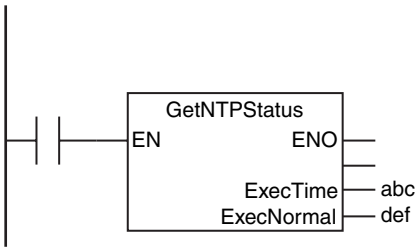
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
ExecTime																			OK	
Exec Normal	OK																			

Function

The GetNTPStatus instruction reads the NTP status. The following information is read: NTP last normal operation time *ExecTime* and normal end flag *ExecNormal*.

The following figure shows a programming example.

LD



ST

```
GetNTPStatus(abc, def);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_NTPResult	NTP Status	*	Contains the NTP status. Refer to the *NJ-series CPU Unit Software User's Manual* (Cat. No. W501) for details.

* _sNTP_RESULT

Precautions for Correct Use

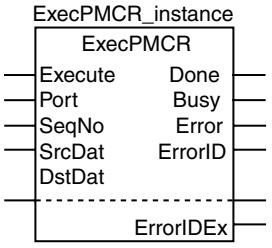
- Return value *Out* is not used when the instruction is used in ST.
- This instruction reads the contents of the *_EIP_NTPResult* system-defined variable. You cannot access this variable directly. Always use this instruction to read the contents of the variable.

Communications Instructions

Instruction	Name	Page	Instruction	Name	Page
ExecPMCR	Protocol Macro	2-648	EC_StopMon	Stop EtherCAT Packet Monitor	2-740
SerialSend	SCU Send Serial	2-658	EC_SaveMon	Save EtherCAT Packets	2-742
SerialRcv	SCU Receive Serial	2-665	EC_CopyMon	Transfer EtherCAT Packets	2-744
SendCmd	Send Command	2-674	EC_DisconnectSlave	Disconnect EtherCAT Slave	2-746
CIPOpen	Open CIP Class 3 Connection	2-684	EC_ConnectSlave	Connect EtherCAT Slave	2-752
CIPRead	Read Variable Class 3 Explicit	2-692	SktUDPCreate	Create UDP Socket	2-754
CIPWrite	Write Variable Class 3 Explicit	2-696	SktUDPRcv	UDP Socket Receive	2-761
CIPSend	Send Explicit Message Class 3	2-701	SktUDPSend	UDP Socket Send	2-764
CIPClose	Close CIP Class 3 Connection	2-704	SktTCPAccept	Accept TCP Socket	2-767
CIPUCMMRead	Read Variable UCMM Explicit	2-706	SktTCPConnect	Connect TCP Socket	2-770
CIPUCMMWrite	Write Variable UCMM Explicit	2-710	SktTCPRcv	TCP Socket Receive	2-777
CIPUCMMSend	Send Explicit Message UCMM	2-716	SktTCPSend	TCP Socket Send	2-780
EC_CoESDOWrite	Write EtherCAT CoE SDO	2-726	SktGetTCPStatus	Read TCP Socket Status	2-783
EC_CoESDORead	Read EtherCAT CoE SDO	2-729	SktClose	Close TCP/UDP Socket	2-786
EC_StartMon	Start EtherCAT Packet Monitor	2-734	SktClearBuf	Clear TCP/UDP Socket Receive Buffer	2-789

ExecPMCR

The ExecPMCR instruction requests execution of a communications sequence (protocol data) registered in a Serial Communications Unit (unit version 2.2 or later).

Instruction	Name	FB/ FUN	Graphic expression	ST expression
ExecPMCR	Protocol Macro	FB		ExecPMCR_instance(Execute, Port, SeqNo, SrcDat, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Port	Destination port	Input	Destination port	---	---	---
SeqNo	Communications sequence number		Communications sequence number	0 to 999		0
SrcDat[] (array)	Send data array		Send data array	Depends on data type.		*
DstDat[] (array)	Receive data array	In-out	Receive data array	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Port	Refer to *Function* for details on the structure _sPORT.																			
SeqNo							OK													
SrcDat[] (array)			OK																	
DstDat[] (array)			OK																	

Function

The ExecPMCR instruction requests execution of the sequence that is specified with communications sequence number *SeqNo* from the specified destination port *Port*.

If data is sent, it is sent from the second element (*SrcDat[1])* of send data array *SrcDat[]*. The number of array elements to send is specified in *SrcDat[0].*

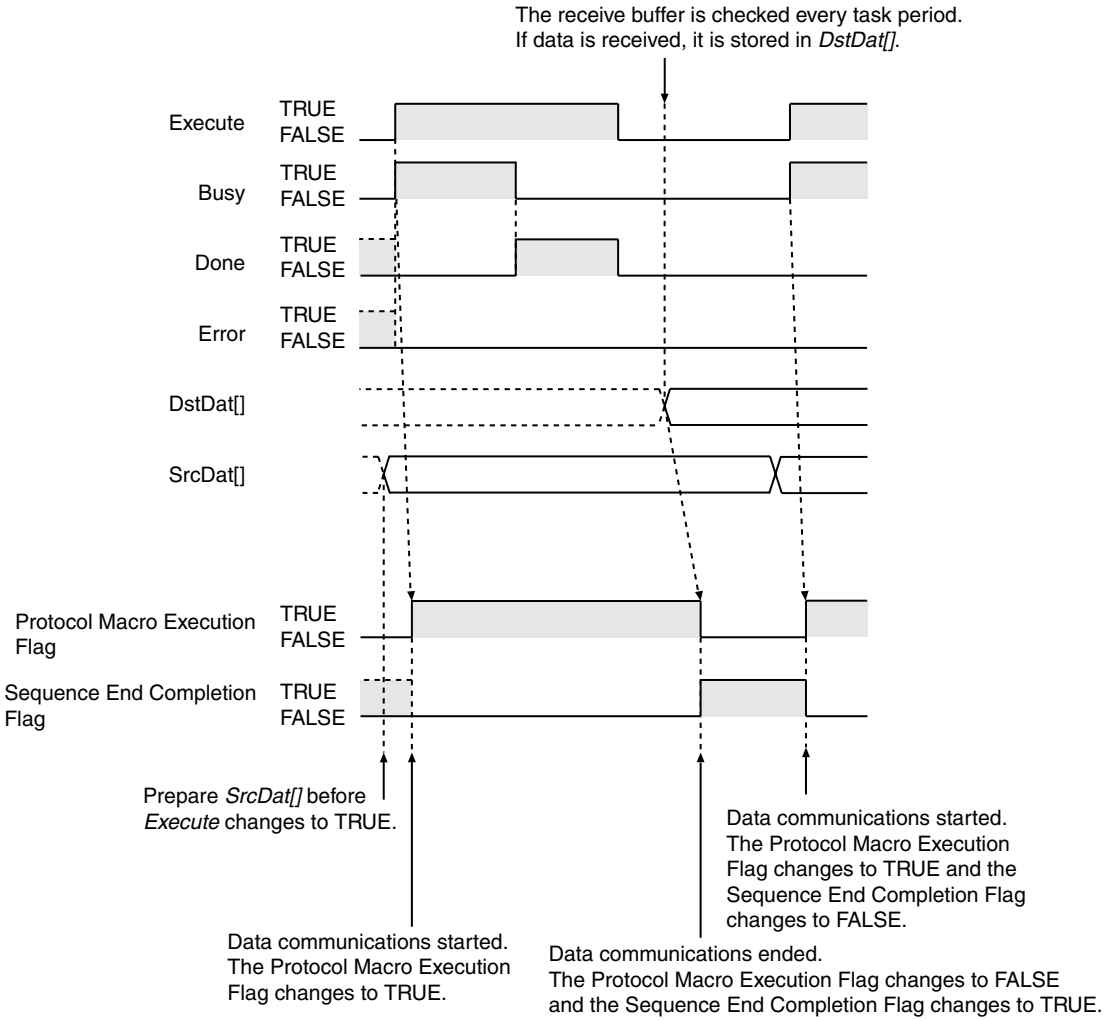
If data is received successfully, the receive data is stored from the second element (*DstDat[1]*) of receive data array *DstDat[]*. The number of receive data elements is stored in *DstDat[0].*

If data is not received successfully, the contents of *DstDat[]* from before instruction execution is retained for the number of elements specified in *DstDat[0]*.

The data type of destination port *Port* is the structure _sPORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Port	Destination port	Destination port	_sPORT	---	---	---
UnitNo	Unit number	Unit number of Serial Communications Unit	_eUnitNo	_CBU_No00 to _CBU_No15		_CBU_No00
PhysicPortNo	Serial port number	Serial port number on Serial Communications Unit	USINT	1 or 2	---	1

The following figure shows a timing chart. Communications is performed to the end after the value of *Done* changes to TRUE.



Related System-defined Variables

Name	Meaning	Data type	Description
_Port_numUsingPort	Number of Used Ports	USINT	This is the number of ports that are currently used.
_Port_isAvailable	Network Communications Instruction Enabled Flag	BOOL	TRUE: A port is available. FALSE: A port is not available.
_CJB_SCU##P1ChgSta, _CJB_SCU##P2ChgSta*	Serial Communications Unit ## Port 1/2 Settings Changing Flag	BOOL	TRUE: The serial port settings are currently being changed. FALSE: The serial port settings are currently not being changed.

* "##" denotes the unit number on the Serial Communications Unit.

Related Semi-user-defined Variables

Name	Meaning	Data type	Description
P#_PmrExecSta*	Protocol Macro Execution Flag	BOOL	TRUE: Protocol macro execution is in progress. FALSE: Protocol macro execution is not in progress or failed.
P#_PmrSeqEndSta*	Sequence End Completion Flag	BOOL	TRUE: The sequence was completed with an End. FALSE: The sequence was not completed with an End.
P#_PmrSeqAbtSta*	Sequence Abort Completion Flag	BOOL	TRUE: The sequence was completed with an Abort. FALSE: The sequence was not completed with an Abort.

* "#" denotes the port number on the Serial Communications Unit.

Additional Information

Refer to the *SYSMAC CX-Protocol Operation Manual* (Cat. No. W344) for details on protocol macros.

Precautions for Correct Use

- The ExecPMCR instruction starts execution of a protocol macro. Use the *P#PmrExecSta* (Protocol Macro Execution Flag) system-defined variable to check the status of protocol macro execution.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- An address in memory for CJ-series Units must be specified in the AT Specification attribute of *DstDat[]*.
- Set the value of *SrcDat[0]* and *DstDat[0]* to 0 to use a direct specification and link word specification. An error occurs if you set any other constant or variable, and the instruction is not executed.
- If the value of *DstDat[0]* is 0 or 1 and reception fails, all elements in *DstDat[]* change to 0.
- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *_Port_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- The instruction is not executed while *Busy* is TRUE. Therefore, use *Busy* in an N.C. execution condition for the instruction.

- The Protocol Macro Execution Flag (semi-user-defined variable *P#_PmrExecSta*) changes to TRUE when instruction execution is started. It changes to FALSE after the communications sequence is completed and the receive data is stored in *DstDat[]*. You cannot execute this instruction for the same serial port while *P#_PmrExecSta* is TRUE. Therefore, use *P#_PmrExecSta* in an N.C. execution condition for the instruction.
- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing is sometimes not possible.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The serial communications mode is not set to Protocol Macro Mode when the instruction is executed.
  - The value of *_Port_isAvailable* is FALSE.
  - The value of *SeqNo* is outside of the valid range.
  - The value of *SeqNo* is not registered to a Serial Communications Unit.
  - The value of *Port.UnitNo* or *Port.PhysicPortNo* is outside of the valid range.
  - There is no CJ-series Serial Communications Unit with the specified unit number.
  - The value of *SrcDat[0]* exceeds the size of *SrcDat[]*.
  - The value of *DstDat[0]* exceeds the size of *DstDat[]*.
  - The value of *SrcDat[0]* or *DstDat[0]* exceeds 250 words.
  - Communications fail.
  - An address in memory for CJ-series Units is not specified in the AT Specification attribute of *DstDat[].*
- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

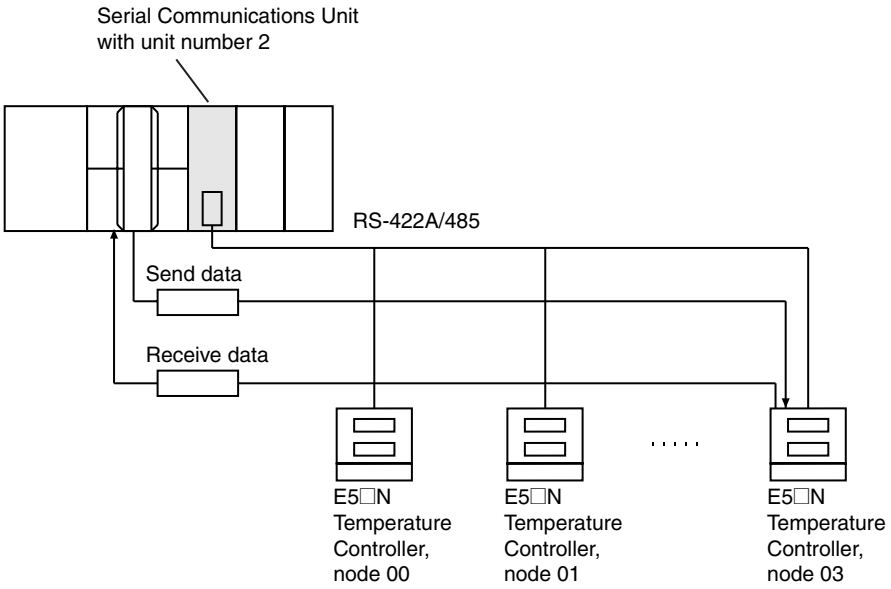
Value	Error	Correction
16#00001106	The value of *SeqNo* is not a registered communications sequence number.	• Correct the value of *SeqNo*. • Add the sequence with the CX-Protocol.
16#00002201	Instruction execution is already in progress. The values of *Busy* and *P#_PmrExecSta* are TRUE.	Use *P#_PmrExecSta* in an N.C. input as the execution condition for the instruction.
16#00002202	The protocol is being switched, so execution is not possible.	Use *_CJB_SCU##P1ChgSta* or *_CJB_SCU##P2ChgSta* Serial Communications Unit, Port 1/2 Settings Changing Flag in an N.C. input as the execution condition for the instruction.
16#00002401	A checksum error occurred in the protocol macro data or the data transfer is not yet completed.	Transfer the protocol macro data from the CX-Protocol.

Sample Programming

In this sample, a CJ-series Serial Communications Unit is used for data communications with an OMRON Temperature Controller. The present value of the Temperature Controller is read with a protocol macro. CompoWay/F master sequence 610 (Read Variable Area) is used. The contents of send data array *SendData[]* is sent from the Controller. The data received from the Temperature Controller is stored in receive data array *RecvData[]*.

The following communications specifications are used.

Item	Description
Unit used	Serial Communications Unit
Unit number	2
Port number	1 (RS-422/485)
Communications sequence number	610 (Read Variable Area)
Remote node number	3
Data to read	Present value



The communications data for sequence 610 (Read Variable Area) is allocated as shown below.

Send Data: WORD Array

SendData[0]	Number of send data words	
SendData[1]	Not used.	Node No.
SendData[2]	Variable type	
SendData[3]	Read start address	
SendData[4]	Number of elements	

Receive Data: WORD Array

RecvData[0]	Number of receive data words
RecvData[1]	Response code
RecvData[2]	Receive data
RecvData[3]	

The contents of send data *SendData[]* and receive data *RecvData[]* are as follows:

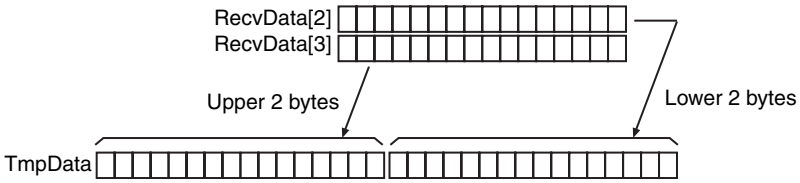
Send Data: WORD Array

Variable	Item	Contents	Value
SendData[0]	Number of send data words	Five words from *SendData[0]* to *SendData[4]* are sent.	WORD#16#0005
SendData[1]	Node number	Node 3 is used.	WORD#16#0003
SendData[2]	Variable type + Upper byte of read start address	To read the present value, the variable type is BYTE#16#C0 and the read start address is WORD#16#00.	WORD#16#C000
SendData[3]	Lower byte of read start address + BYTE#16#00 (fixed value)		WORD#16#0000
SendData[4]	Number of elements	One element is read.	WORD#16#0001

Receive Data: WORD Array

Variable	Item	Contents	Value
RecvData[0]	Number of receive data words	Four words from *RecvData[0]* to *RecvData[3]* are received.	WORD#16#0004
RecvData[1]	Response code	WORD#16#0000 is returned for a normal end.	
RecvData[2]	Receive data	The lower two bytes of the present value of the Temperature Controller are returned.	
RecvData[3]		The upper two bytes of the present value of the Temperature Controller are returned.	

If the data is received successfully, the lower two bytes (*RecvData[2]*) and the upper two bytes (*RecvData[3]*) of the present value of the Temperature Controller are assigned to *TmpData*.



Definitions of Global Variables

Global Variables

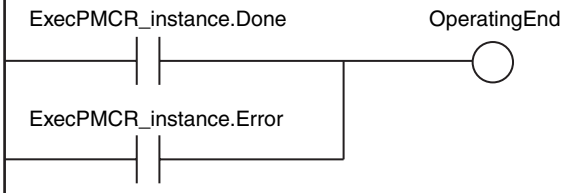
Name	Data type	AT specification	Comment
SCU_P1_PmrSeqEndSta	BOOL	IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrSeqEndSta	Sequence End Completion Flag
SCU_P1_PmrSeqAbtSta	BOOL	IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrSeqAbtSta	Sequence Abort Completion Flag
SCU_P1_PmrExecSta	BOOL	IOBus://rack#0/slot#0/P1_PmrSta/P1_PmrExecSta	Protocol Macro Execution Flag

LD

Internal Variables	Variable	Data type	Initial value	AT specification	Retain	Comment
	OperatingEnd	BOOL	False		☐	Processing completed
	Trigger	BOOL	False		☐	Execution condition
	Operating	BOOL	False		☐	Processing
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)		☐	Port settings
	SendData	ARRAY[0..4] OF WORD	[5(16#0)]		☐	Send data
	RecvData	ARRAY[0..3] OF WORD	[4(16#0)]	%D200	☑	Receive data
	TmpData	DINT	0		☐	Present value
	RS_instance	RS			☐	
	ExecPMCR_instance	ExecPMCR			☐	

External Variables	Variable	Data type	Comment
	SCU_P1_PmrSeqEndSta	BOOL	Sequence End Completion Flag
	SCU_P1_PmrSeqAbtSta	BOOL	Sequence Abort Completion Flag
	SCU_P1_PmrExecSta	BOOL	Protocol Macro Execution Flag
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag

Determine if execution of the ExecPMCR instruction is completed.



Accept trigger.

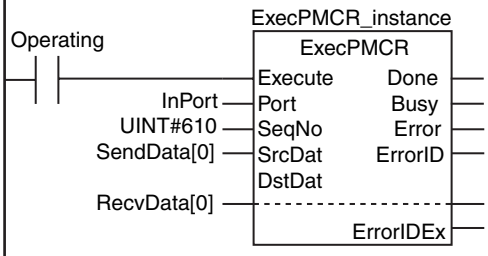
Set communications parameters.

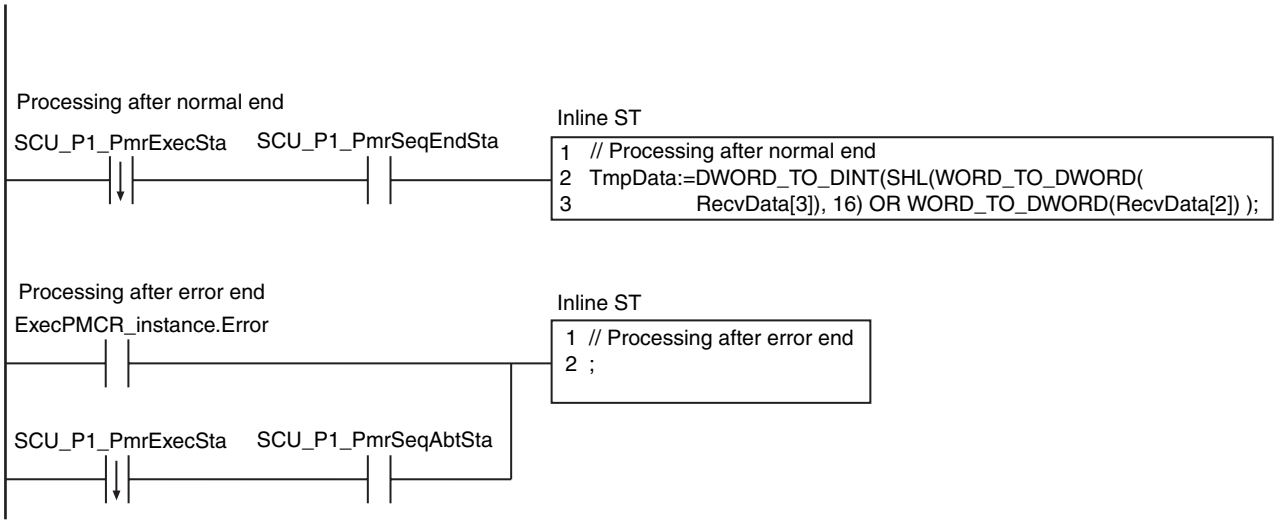
Operating

Inline ST

```
1 InPort.UnitNo         :=_CBU_No02;   // Serial Communications Unit with unit number 2
2 InPort.PhysicPortNo:=USINT#1;        // Port number 1
3 SendData[0]           :=WORD#16#0005;
4 SendData[1]           :=WORD#16#0003;
5 SendData[2]           :=WORD#16#C000;
6 SendData[3]           :=WORD#16#0000;
7 SendData[4]           :=WORD#16#0001;
8 RecvData[0]           :=WORD#16#0004;
```

Execute ExecPMCR instruction.




Processing after normal end
SCU_P1_PmrExecSta
SCU_P1_PmrSeqEndSta
Inline ST
1 // Processing after normal end
2 TmpData:=DWORD_TO_DINT(SHL(WORD_TO_DWORD(
3 RecvData[3]), 16) OR WORD_TO_DWORD(RecvData[2]));
Processing after error end
ExecPMCR_instance.Error
Inline ST
1 // Processing after error end
2 ;
SCU_P1_PmrExecSta
SCU_P1_PmrSeqAbtSta

ST

Internal Variables	Variable	Data type	Initial value	AT specification	Retain	Comment
	State	INT	0		☐	Current state
	Trigger	BOOL	False		☐	Execution condition
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)		☐	Port settings
	SendData	ARRAY[0..4] OF WORD	[5(16#0)]		☐	Send data
	RecvData	ARRAY[0..3] OF WORD	[4(16#0)]	%D200	☑	Receive data
	End_ExecPMCR	BOOL	False		☐	Completion of ExecPMCR instruction execution
	TmpData	DINT	0		☐	Present value
	RS_instance	RS			☐	
	ExecPMCR_instance	ExecPMCR			☐	
	F_TRIG_instance	F_TRIG			☐	

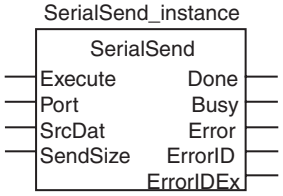
External Variables	Variable	Data type	Comment
	SCU_P1_PmrSeqEndSta	BOOL	Sequence End Completion Flag
	SCU_P1_PmrSeqAbtSta	BOOL	Sequence Abort Completion Flag
	SCU_P1_PmrExecSta	BOOL	Protocol Macro Execution Flag
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag

```
// Accept trigger.
IF (State=INT#0) THEN
    IF ( (Trigger=TRUE) AND (_Port_isAvailable=TRUE) AND (SCU_P1_PmrExecSta<>TRUE)
         AND (ExecPMCR_instance.Busy<>TRUE) ) THEN
        State:=INT#1;
    END_IF;
END_IF;

// Set communications parameters and initialize ExecPMCR instruction.
IF (State=INT#1) THEN
    InPort.UnitNo          :=_CBU_No02;        // Serial Communications Unit with unit number 2
    InPort.PhysicPortNo:=USINT#1;              // Port number 1
    SendData[0]            :=WORD#16#0005;
    SendData[1]            :=WORD#16#0003;
    SendData[2]            :=WORD#16#C000;
    SendData[3]            :=WORD#16#0000;
    SendData[4]            :=WORD#16#0001;
    RecvData[0]            :=WORD#16#0004;
    ExecPMCR_instance(
        Execute:=FALSE,            // Initialize ExecPMCR instruction.
        SrcDat  :=SendData[0],     // Dummy
        DstDat  :=RecvData[0]);
    State:=INT#2;
END_IF;

// Execute ExecPMCR instruction.
IF (State=INT#2) THEN
    ExecPMCR_instance(
        Execute:=TRUE,
        Port      :=InPort,
        SeqNo  :=UINT#610,
        SrcDat  :=SendData[0],
        DstDat  :=RecvData[0]);

    F_TRIG_instance(SCU_P1_PmrExecSta, End_ExecPMCR);

            IF (End_ExecPMCR=TRUE) THEN
                End_ExecPMCR:=FALSE;
                State:=INT#3;
            END_IF;

            IF (ExecPMCR_instance.Error=TRUE) THEN
                State:=INT#5;
            END_IF;

END_IF;
```

```
// Confirm completion of ExecPMCR instruction execution.
IF (State=INT#3) THEN
    IF (SCU_P1_PmrSeqEndSta=TRUE) THEN
        State:=INT#4;
    END_IF;
    IF (SCU_P1_PmrSeqAbtSta=TRUE) THEN
        State:=INT#5;
    END_IF;
END_IF;

IF (State=INT#4) THEN
    // Processing after normal end.
    TmpData:=DWORD_TO_DINT(SHL(WORD_TO_DWORD(RecvData[3]), 16)
                OR WORD_TO_DWORD(RecvData[2]));
    State:=INT#0;
END_IF;

IF (State=INT#5) THEN
    // Processing after error end
    State:=INT#0;
END_IF;
```

SerialSend

The SerialSend instruction sends data in No-protocol Mode from a serial port on a Serial Communications Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SerialSend	SCU Send Serial	FB		SerialSend_instance(Execute, Port, SrcDat, SendSize, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Port	Destination port	Input	Destination port	---	---	---
SrcDat[] (array)	Send data array		Send data array	Depends on data type.		*
SendSize	Number of send data elements		Number of elements to send from *SrcDat[]*	0 to 256	Bytes	1

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Port	Refer to *Function* for details on the structure _sPORT.																			
SrcDat[] (array)		OK																		
SendSize							OK													

Function

The SerialSend instruction sends data in No-protocol Mode from the port and the Serial Communications Unit specified with *Port*. The data that is sent is contained in send data array *SrcDat[]*. The number of array elements to send is specified in number of send data elements *SendSize*.

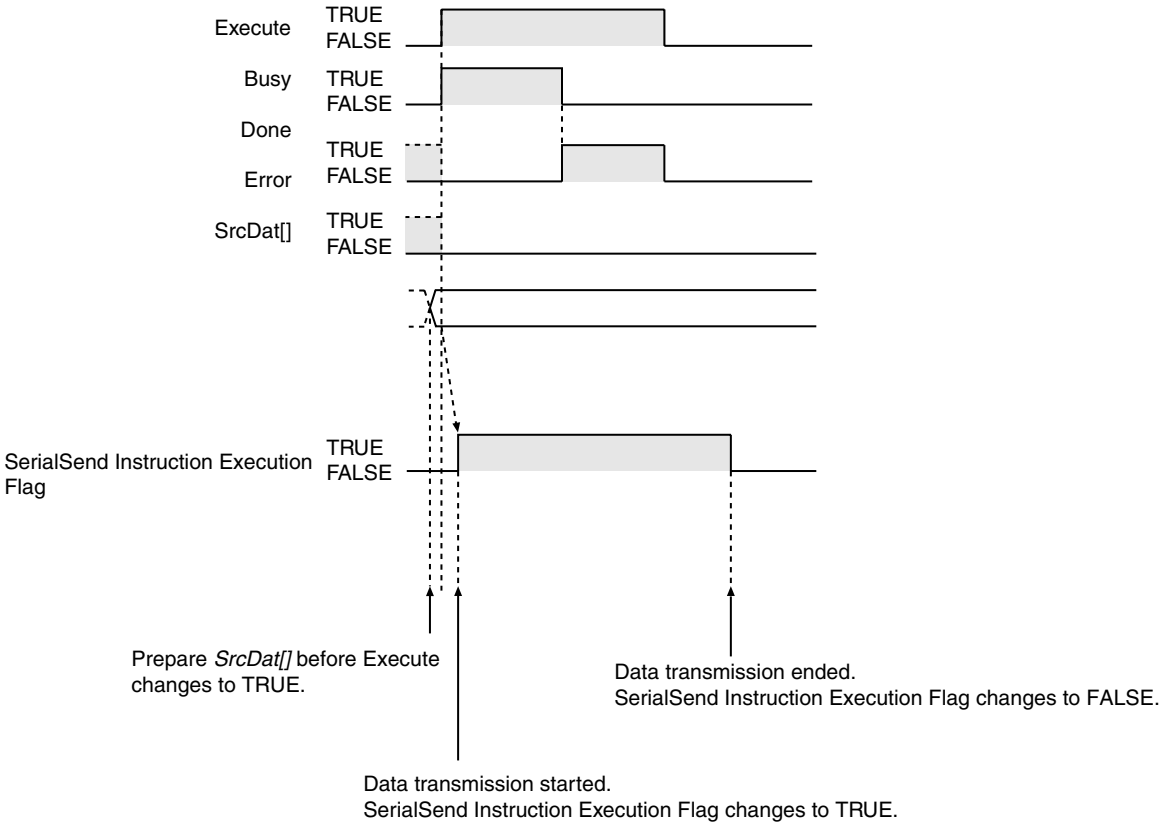
To attach start and end codes to the send data, set them in the DM Area words that are assigned to the Special Unit.

If you add start and end codes, the maximum number of bytes to send is 259 (1-byte start code, 2-byte end code (for CR+LF specification), and 256 bytes of send data).

The data type of destination port *Port* is the structure _sPORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Port	Destination port	Destination port	_sPORT	---	---	---
UnitNo	Unit number	Unit number of Serial Communications Unit	_eUnitNo	_CBU_No00 to _CBU_No15	---	_CBU_No00
PhysicPortNo	Serial port number	Serial port number on Serial Communications Unit	USINT	1 or 2		1

The following figure shows a timing chart. Communications is performed to the end after the value of *Done* changes to TRUE.



Related System-defined Variables

Name	Meaning	Data type	Description
_Port_numUsingPort	Number of Used Ports	USINT	This is the number of ports that are currently used.
_Port_isAvailable	Network Communications Instruction Enabled Flag	BOOL	TRUE: A port is available. FALSE: A port is not available.

Related Semi-user-defined Variables

Name	Meaning	Data type	Description
P#_NopSerialSendExecSta*	SerialSend Instruction Execution Flag	BOOL	TRUE: Execution of the SerialSend instruction is in progress. FALSE: Execution of the SerialSend instruction is not in progress.
P#_NopStartCodeYNCfg*	No-protocol Start Code Enable	BOOL	TRUE: Start code FALSE: No start code
P#_NopEndCodeYNCfg*	No-protocol End Code Enable	BOOL	TRUE: End code FALSE: No end code
P#_NopCRLFCfg*	No-protocol CR LF Specification	BOOL	TRUE: CR+LF FALSE: No CR+LF
P#_NopStartCodeCfg*	No-protocol Start Code	USINT	16#00 to 16#FF
P#_NopEndCodeCfg*	No-protocol End Code	USINT	16#00 to 16#FF

* "#" denotes the port number on the Serial Communications Unit.

Additional Information

Refer to the following manual for details on no-protocol communications.

- *CJ-series Serial Communications Units Operation Manual for NJ-series CPU Unit* (Cat. No. W494)

Precautions for Correct Use

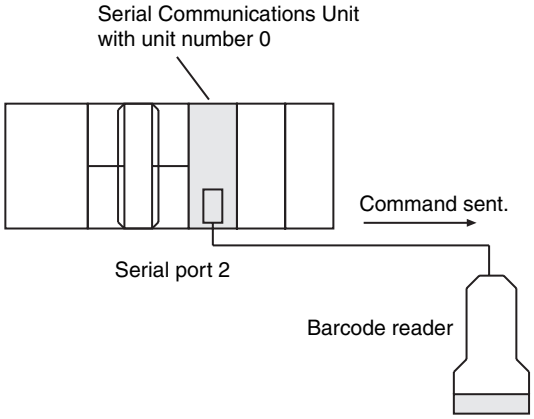
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You can use this instruction only for a Serial Communications Unit's serial port that is set to No-protocol Mode.
- Nothing is sent if the value of *SendSize* is 0. When the instruction is executed, the value of *Done* changes to TRUE.
- Even when attaching a start or end code, do not include it in the value of *SendSize*.
- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *_Port_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- The instruction is not executed while *Busy* is TRUE. Therefore, use *Busy* in an N.C. execution condition for the instruction.
- You cannot execute this instruction while the SerialSend Instruction Executing Flag (semi-user-defined variable *P#NopSerialSendExecSta*) is TRUE. Use *P#NopSerialSendExecSta* in an N.C. execution condition for the instruction.
- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing is sometimes not possible.

- You cannot use this instruction in the primary periodic task.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The serial communications mode is not set to No-protocol Mode when the instruction is executed.
  - The value of *_Port_isAvailable* is FALSE.
  - The value of *Port.UnitNo* or *Port.PhysicPortNo* is outside of the valid range.
  - There is no CJ-series Serial Communications Unit with the specified unit number.
  - The value of *SendSize* is outside of the valid range.
  - The value of *SendSize* exceeds the size of *SrcDat[]*.
  - Communications fail.
  - The instruction is executed during a Unit restart.
- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

Value	Meaning
16#00000401	The serial communications mode is set to Protocol Macro, NT Link, Echoback Test, or Serial Gateway Mode.
16#00000205	The serial communications mode is set to Host Link Mode.
16#00001001	The command is too long.
16#00001002	The command is too short.
16#00001003	The value of *SendSize* does not match the number of send bytes.
16#00001004	The command format is incorrect.
16#0000110C	This is another parameter error.
16#00002201	The SerialSend or SerialRcv instruction is already in execution.
16#00002202	The protocol is being switched, so execution is not possible.

Sample Programming

In this sample, a no-protocol command is sent to the barcode reader that is connected to serial port 2 of a CJ-series Serial Communications Unit (unit number 0, device name 'Barcode'). The Read Scene Number command (@READ) is sent. The send data is the contents of the array variable *SendDat[]*. There is no start code and the end code is 16#OD (CR).

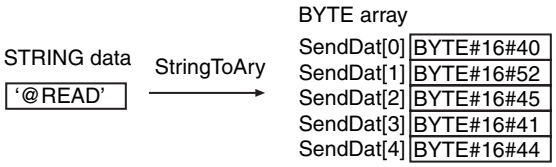


The settings of Serial Communications Unit are given in the following table.

Item	Set value
Port 2: User-specified Setting Inclusion	User settings
Port 2: Serial Communications Mode	No-protocol
Port 2: Data Length	8 bits
Port 2: Stop Bits	1 bit
Port 2: Parity	No
Port 2: Baud Rate	38,400 bps

Item	Set value
Port 2: No-Protocol End Code	D
Port 2: No-Protocol Start Code Inclusion Setting	No
Port 2: No-Protocol End Code Inclusion Setting	Yes (Specify a desired end code.)

The text string '@READ' is separated into individual characters and the character codes are stored in the array elements of *SendDat[]*. Therefore, BYTE#16#40 (@) is stored in *SendDat[0]*, BYTE#16#52(R) is stored in *SendData[1]*, etc. The StringToAry instruction is used to store the character codes.



Definitions of Global Variables

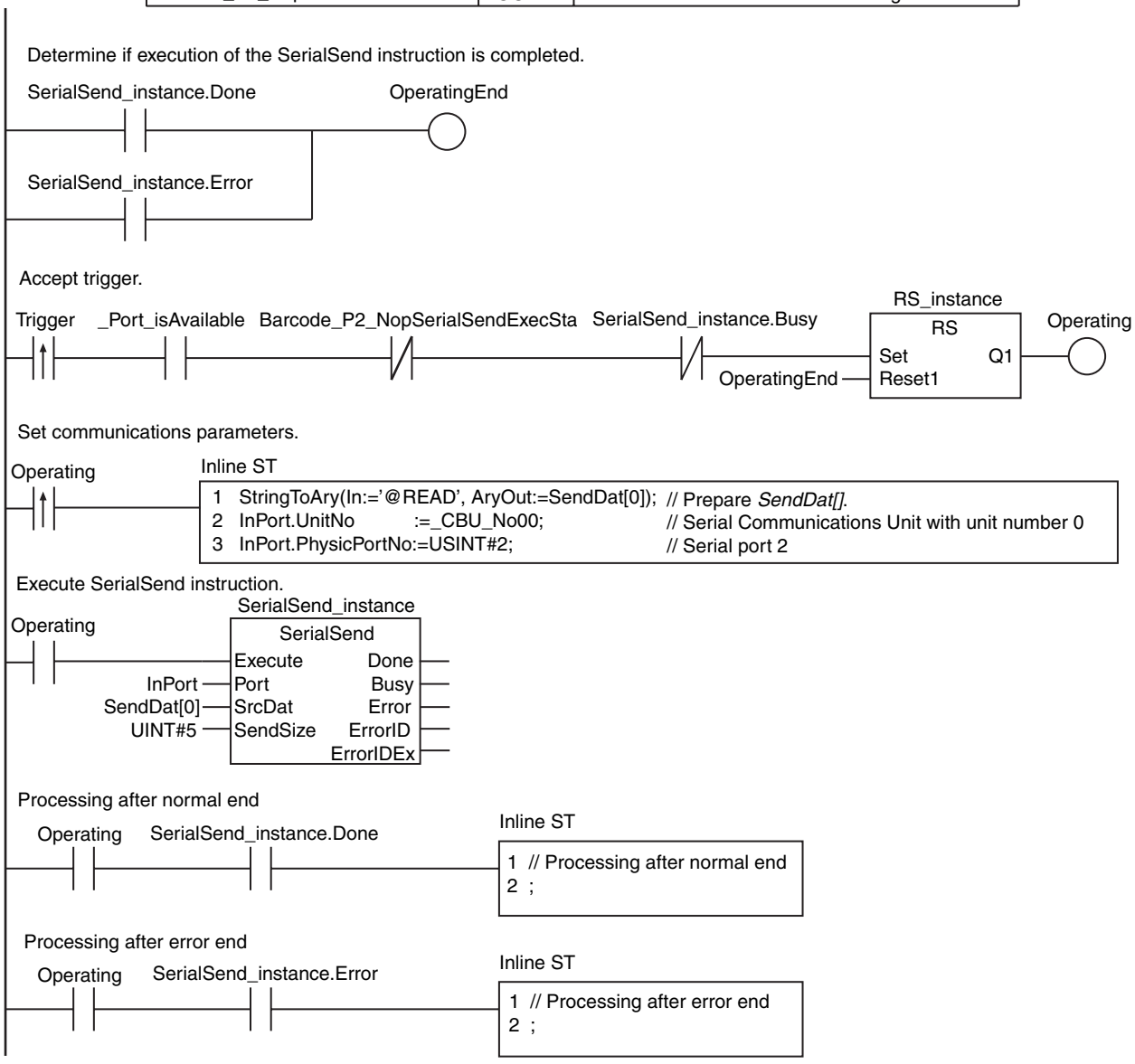
Global Variables

Name	Data type	AT specification	Comment
Barcode_P2_NopSerialSendExecSta	BOOL	IOBus://rack#0/slot#0/P2_NopSta/P2_NopSerialSendExecSta	SerialSend Instruction Execution Flag

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)	Port settings
	SendDat	ARRAY[0..4] OF BYTE	[5(16#0)]	Send data
	RS_instance	RS		
	SerialSend_instance	SerialSend		

External Variables	Variable	Data type	Comment
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag
	Barcode_P2_NopSerialSendExecSta	BOOL	SerialSend Instruction Execution Flag



ST

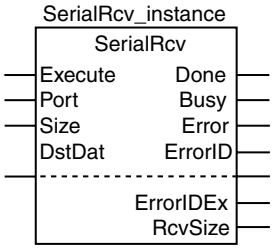
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started
	Operating	BOOL	False	Processing
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)	Port settings
	SendDat	ARRAY[0..4] OF BYTE	[5(16#0)]	Send data
	SerialSend_instance	SerialSend		

External Variables	Variable	Data type	Comment
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag
	Barcode_P2_NopSerialSendExecSta	BOOL	SerialSend Instruction Execution Flag

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
        AND (Barcode_P2_NopSerialSendExecSta=FALSE) AND (SerialSend_instance.Busy=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize SerialSend instruction.
IF (OperatingStart=TRUE) THEN
    SerialSend_instance(
        Execute:=FALSE,
        SrcDat  :=SendDat[0]);
    StringToAry(In:='@READ', AryOut:=SendDat[0]);
    InPort.UnitNo          :=_CBU_No00;                    // Serial Communications Unit with unit number 0
    InPort.PhysicPortNo:=USINT#2;                          // Serial port 2
    OperatingStart         :=FALSE;
END_IF;

// Execute SerialSend instruction.
IF (Operating=TRUE) THEN
    SerialSend_instance(
        Execute  :=TRUE,
        Port     :=InPort,          // Port settings
        SrcDat   :=SendDat[0],      // Send data
        SendSize:=UINT#5);          // Send data size

    IF (SerialSend_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;

    IF (SerialSend_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;
```

SerialRcv

The SerialRcv instruction receives data in No-protocol Mode from a serial port on a Serial Communications Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SerialRcv	SCU Receive Serial	FB		SerialRcv_instance(Execute, Port, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Port	Destination port	Input	Destination port	---	---	---
Size	Receive data size		Size of receive data stored in *DstDat[]*	0 to 256	Bytes	1
DstDat[] (array)	Receive data array	In-out	Receive data array	Depends on data type.	---	---
RcvSize	Number of receive data array elements	Output	Number of receive data array elements actually stored in *DstDat[]*	0 to 256	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Port	Refer to *Function* for details on the structure _sPORT.																			
Size							OK													
DstDat[] (array)		OK																		
RcvSize							OK													

Function

First, data is received in No-protocol Mode from the serial port specified with *Port* and stored in the receive buffer in the Serial Communications Unit. This instruction transfers the number of bytes specified with receive data size *Size* from the receive buffer to receive data array *DstDat[]*.

After the data is transferred, the number of array elements that was actually stored in *DstDat[]* is assigned to the number of receive data array elements *RcvSize*. If the amount of data in the receive buffer is smaller than *Size*, all of the data in the receive buffer is transferred to *DstDat[]*. The number of array elements that was actually stored in *DstDat[]* is assigned to *RcvSize*. The receive buffer is cleared after the data is transferred.

Device variables are used in the user program to recognize the start code and end code in the receive data. The start and end codes are deleted from the receive data before it is stored in *DstDat[].*

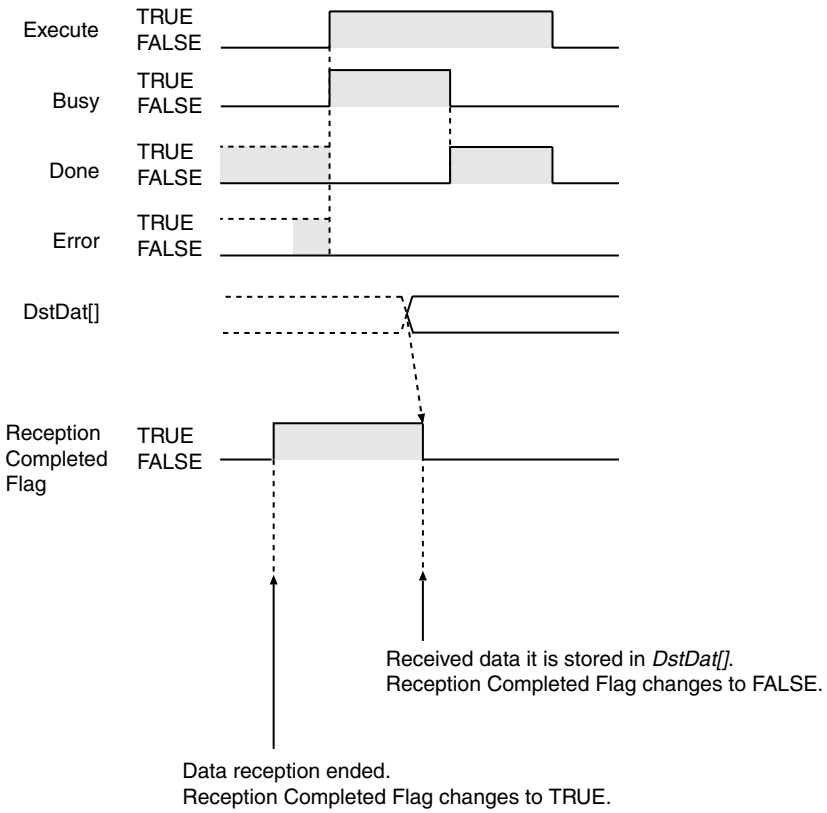
Code to attach	Device variable for port 1	Value
Specified start code	P1_NopStartCodeYNCfg	TRUE
	P1_NopStartCodeCfg	Start code (16#00 to 16#FF)
Specified end code	P1_NopEndCodeYNCfg	TRUE
	P1_NopCRLFCfg	FALSE
	P1_NopEndCodeCfg	End code (16#00 to 16#FF)
CR+LF as end code	P1_NopEndCodeYNCfg	TRUE
	P1_NopCRLFCfg	TRUE

If you add start and end codes, the maximum number of bytes to receive is 259 (1-byte start code, 2-byte end code (for CR+LF specification), and 256 bytes of send data).

The data type of destination port *Port* is the structure _sPORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Port	Destination port	Destination port	_sPORT	---	---	---
UnitNo	Unit number	Unit number of Serial Communications Unit	_eUnitNo	_CBU_No00 to _CBU_No15	---	_CBU_No00
PhysicPortNo	Serial port number	Serial port number on Serial Communications Unit	USINT	1 or 2		1

The following figure shows a timing chart.



Related System-defined Variables

Name	Meaning	Data type	Description
_Port_numUsingPort	Number of Used Ports	USINT	This is the number of ports that are currently used.
_Port_isAvailable	Network Communications Instruction Enabled Flag	BOOL	TRUE: A port is available. FALSE: A port is not available.

Related Semi-user-defined Variables

Name	Meaning	Data type	Description
P#_NopRcvOvfSta*	Reception Overflow Flag	BOOL	TRUE: The Unit received more than the specified amount of data (i.e., data was received after the Reception Completed Flag changed to TRUE). FALSE: The Unit did no receive more than the specified number of bytes.
P#_NopRcvCompleteSta*	Reception Completed Flag	BOOL	TRUE: Reception was completed. FALSE: No data received or currently receiving data.
P#_NopRcvCntSta*	Reception Counter	UINT	16#0000 to 16#0100: Number of bytes of receive data
P#_NopStartCodeYNCfg*	No-protocol Start Code Enable	BOOL	TRUE: Start code FALSE: None
P#_NopEndCodeYNCfg*	No-protocol End Code Enable	BOOL	TRUE: End code FALSE: None
P#_NopCRLFCfg*	No-protocol CR LF Specification	BOOL	TRUE: CR+LF FALSE: No CR+LF

Name	Meaning	Data type	Description
P#_NopRcvDatSzCfg*	Number of No-protocol Receive Data Bytes	USINT	16#01 to 16#FF: 1 to 255 bytes 16#00: 256 bytes
P#_NopStartCodeCfg*	No-protocol Start Code	USINT	16#00 to 16#FF
P#_NopEndCodeCfg*	No-protocol End Code	USINT	16#00 to 16#FF
P#_TransErr*	Transmission Error	BOOL	TRUE: Error occurred. FALSE: No error occurred.
P#_OverRunErr*	Overrun Error	BOOL	TRUE: Error occurred. FALSE: No error occurred.

* "#" denotes the port number on the Serial Communications Unit.

Additional Information

- The Reception Completed Flag (*P#_NopRcvCompleteSta*) changes to TRUE at the following times.
  - The amount of data set in Number of No-protocol Receive Data Bytes (*P#_NopRcvDatSzCfg*) is received.
  - The specified end code is received.
  - A total of 256 bytes of data is received.
- The Reception Overflow Flag (*P#_NopRcvOvfSta*) changes to TRUE at the following times.
  - Data is received when this instruction is not executed and the Reception Completed Flag (*P#_NopRcvCompleteSta*) is TRUE.
  - More than the amount of data set in Number of No-protocol Receive Data Bytes (*P#_NopRcvDatSzCfg*) is received.
- Refer to the following manual for details on no-protocol communications.
  - *CJ-series Serial Communications Units Operation Manual for NJ-series CPU Unit* (Cat. No. W494)

Precautions for Correct Use

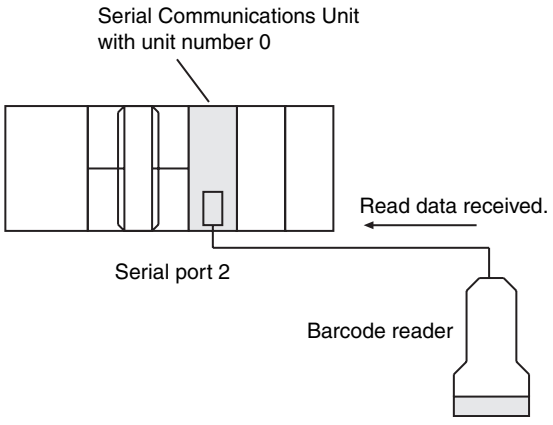
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Execute the instruction only when the Reception Completed Flag (*P#_NopRcvCompleteSta*) is TRUE.
- When data is received, always execute this instruction to transfer the data in the receive buffer to *DstDat[].* No more data is received until the previous data is transferred.
- The receive buffer in the Serial Communications Unit is cleared when this instruction is executed. Therefore, you cannot separate the data in the receive buffer to transfer it to *DstDat[].*
- Reception stops automatically after 259 bytes of data is received. If this instruction is not executed after that and more data is received, Overrun Error (*P#_OverRunErr*) changes to TRUE.
- Any receive data that exceeds the size specified with *Size* is discarded the next time the instruction is executed.
- Even when a start or end code is attached, do not include it in the value of *Size*.
- You can use this instruction only for a Serial Communications Unit's serial port that is set to No-protocol Mode.
- If the value of *Size* is 0, the data in the receive buffer is not transferred to *DstDat[].* If that occurs, the Reception Completed Flag (*P#_NopRcvCompleteSta*) and Reception Overflow Flag (*P#_NopRcvOvfSta*) will change to FALSE. Also, the Reception Counter (*P#_NopRcvCntSta*) will be 0.

- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *_Port_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- Execute the instruction only when the Reception Completed Flag (*P#_NopRcvCompleteSta*) is TRUE.
- The instruction is not executed while *Busy* is TRUE. Therefore, use *Busy* in an N.C. execution condition for the instruction.
- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing is sometimes not possible.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The serial communications mode is not set to No-protocol Mode when the instruction is executed.
  - The value of *_Port_isAvailable* is FALSE.
  - The value of *Port.UnitNo* or *Port.PhysicPortNo* is outside of the valid range.
  - There is no CJ-series Serial Communications Unit with the specified unit number.
  - The value of *Size* is outside of the valid range.
  - The value of *Size* exceeds the size of *DstDat[]*.
  - Communications fail.
  - The instruction is executed during a Unit restart.
- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

Value	Meaning
16#00000401	The serial communications mode is set to Protocol Macro, NT Link, Echoback Test, or Serial Gateway Mode.
16#00000205	The serial communications mode is set to Host Link Mode.
16#00001001	The command is too long.
16#00001002	The command is too short.
16#00001004	The command format is incorrect.
16#0000110C	This is another parameter error.
16#00002201	The SerialSend or SerialRcv instruction is already in execution.
16#00002202	The protocol is being switched, so execution is not possible.

Sample Programming

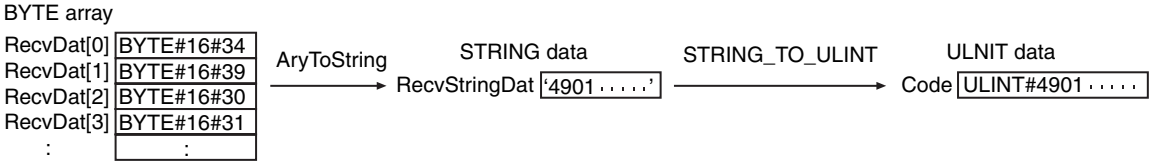
In this sample, data that was read by the barcode reader that is connected to serial port 2 of a CJ-series Serial Communications Unit (unit number 0, device name 'Barcode') is received. The receive data is stored in array variable *RecvDat[]*. There is no start code and the end code is 16#OD (CR).



The settings of Serial Communications Unit are given in the following table.

Item	Set value
Port 2: User-specified Setting Inclusion	User settings
Port 2: Serial Communications Mode	No-protocol
Port 2: Data Length	8 bits
Port 2: Stop Bits	1 bit
Port 2: Parity	No
Port 2: Baud Rate	38,400 bps
Port 2: No-Protocol End Code	D
Port 2: No-Protocol Start Code Inclusion Setting	No
Port 2: No-Protocol End Code Inclusion Setting	Yes (Specify a desired end code.)

The number from the barcode reader is separated into individual characters and bit strings for the character codes are stored in *RecvDat[]*. One element of the *RecvDat[]* array corresponds to one character from the barcode. First, the AryToString instruction is used to convert the data to a text string (*RecvStringDat*). Then, the STRING_TO_ULINT instruction is used to convert the data to an ULINT integer (*Code*).



Definitions of Global Variables

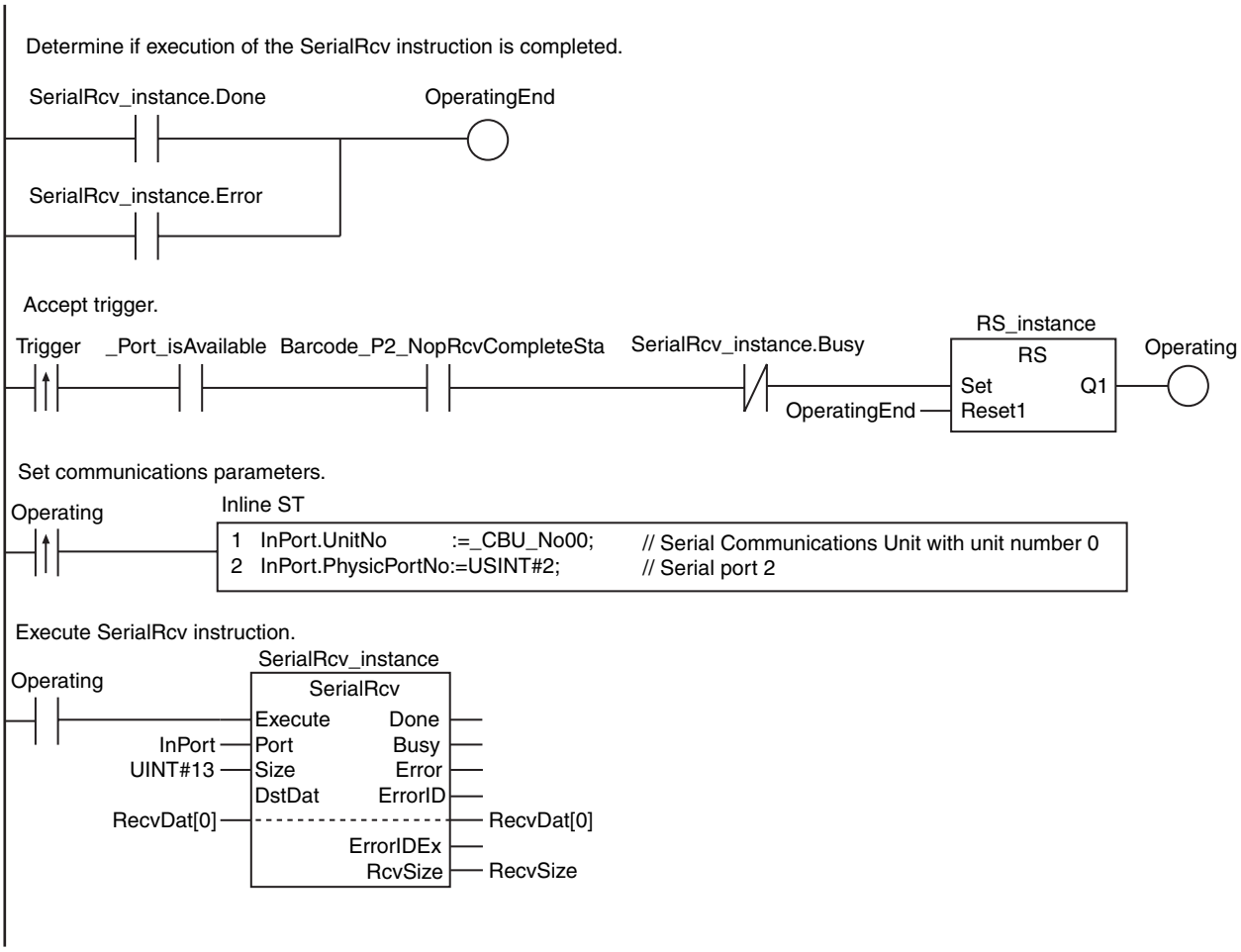
Global Variables

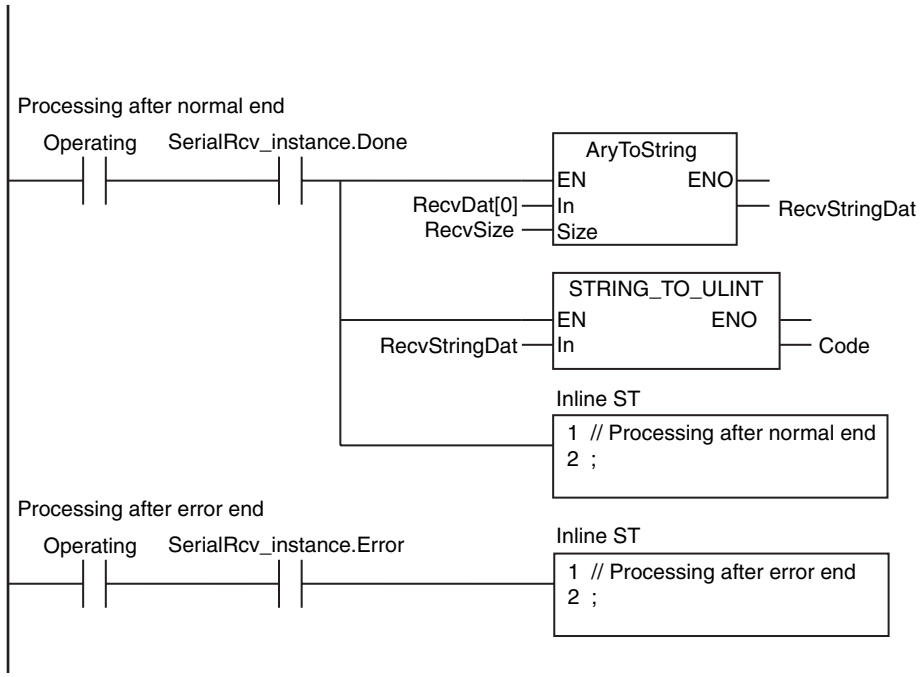
Name	Data type	AT specification	Comment
Barcode_P2_NopRcvCompleteSta	BOOL	IOBus://rack#0/slot#0/P2_NopSta/P2_NopRcvCompleteSta	Reception Completed Flag

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	InPort	_sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)	Port settings
	RecvDat	ARRAY[0..12] OF BYTE	[13(16#0)]	Receive data
	RecvSize	UINT	0	Receive data size
	RecvStringDat	STRING[255]	''	Barcode text string
	Code	ULINT	0	Barcode integer
	RS_instance	RS		
	SerialRcv_instance	SerialRcv		

External Variables	Variable	Data type	Comment
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag
	Barcode_P2_NopRcvCompleteSta	BOOL	Reception Completed Flag




Processing after normal end
Operating
SerialRcv_instance.Done
AryToString
EN
ENO
RecvDat[0]
In
RecvSize
Size
RecvStringDat
STRING_TO_ULINT
EN
ENO
RecvStringDat
In
Code
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
SerialRcv_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

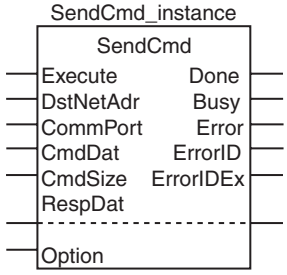
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started
	Operating	BOOL	False	Processing
	InPort	sPORT	(UnitNo:=_CBU_No00, PhysicPortNo:=0)	Port settings
	RecvDat	ARRAY[0..12] OF BYTE	[13(16#0)]	Receive data
	RecvSize	UINT	0	Receive data size
	RecvStringDat	STRING[255]	''	Barcode text string
	Code	ULINT	0	Barcode integer
	SerialRcv_instance	SerialRcv		

External Variables	Variable	Data type	Comment
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag
	Barcode_P2_NopRcvCompleteSta	BOOL	Reception Completed Flag

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
        AND (Barcode_P2_NopRcvCompleteSta=TRUE) AND (SerialRcv_instance.Busy=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize SerialRcv instance.
IF (OperatingStart=TRUE) THEN
    SerialRcv_instance(
        Execute:=FALSE,        // Initialize instance.
        Port    :=InPort,      // Port settings
        Size    :=UINT#13,     // Receive data size
        DstDat  :=RecvDat[0],  // Receive data
        RcvSize=>RecvSize);    // Data size that was actually received
    InPort.UnitNo         :=_CBU_No00;  // Serial Communications Unit with unit number 0
    InPort.PhysicPortNo:=USINT#2;       // Serial port 2
    OperatingStart        :=FALSE;
END_IF;

// Execute SerialRcv instance.
IF (Operating=TRUE) THEN
    SerialRcv_instance(
        Execute:=TRUE,
        Port    :=InPort,
        Size    :=UINT#13,
        DstDat  :=RecvDat[0],
        RcvSize=>RecvSize);

    IF (SerialRcv_instance.Done=TRUE) THEN
        // Processing after normal end
        RecvStringDat:=AryToString(In:=RecvDat[0], Size:=RecvSize);  // Convert character codes to a text string.
        Code          :=STRING_TO_ULINT(RecvStringDat);              // Convert text string to an integer.
        Operating     :=FALSE;
    END_IF;

    IF (SerialRcv_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;
```

SendCmd

The SendCmd instruction uses a serial gateway and sends a command to a Serial Communications Unit. Or, it sends an explicit command to a DeviceNet Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SendCmd	Send Command	FB		SendCmd_instance(Execute, DstNetAdr, CommPort, CmdDat, CmdSize, RespDat, Option, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DstNetAdr	Destination network address	Input	Destination network address	---	---	---
CommPort	Destination serial port		Destination serial port	Only _NONE		_NONE
CmdDat[] (array)	Command array		Command to send	Depends on data type.		*
CmdSize	Command data size		Command data size	2 to maximum data length (depends on network type)	Bytes	2
Option	Response		Response monitoring and retry specifications	---	---	---
RespDat[] (array)	Response storage array	In-out	Array to store response	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DstNetAdr	Refer to *Function* for details on the structure _sDNET_ADR.																			
CommPort	Refer to *Function* for the enumerators of the enumerated type _ePORT.																			
CmdDat[] (array)		OK																		
CmdSize							OK													
Option	Refer to *Function* for details on the structure _sRESPONSE.																			
RespDat[] (array)		OK																		

Function

The SendCmd instruction sends the contents of command array *CmdDat[]* to the destination specified with destination network address *DstNetAdr* and destination serial port *CommPort*. The command data size *CmdSize* specifies how many elements of *CmdDat[]* contain the command. The response that is returned is stored in response storage array *RespDat[]*.

The data type of *DstNetAdr* is structure _sDNET_ADR. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DstNetAdr	Destination network address	Destination network address	_sDNET_ADR	---	---	---
NetNo	Network address	Network address	USINT	0	---	0
NodeNo	Node address	Node address	USINT			
UnitNo	Unit address	Unit address	BYTE	Depends on data type.		16#00

The destination node is found with routing tables. If the value of *DstNetAdr.NetNo* is 0, the data is routed through the built-in EtherNet/IP port. If the value of *DstNetAdr.NodeNo* is 255, the data is broadcast to all nodes with the specified network address.

The data type of *CommPort* is enumerated type _ePORT. The meanings of the enumerators of enumerated type _ePORT are as follows:

Enumerators	Meaning
_NONE	The destination is not a serial port in Host Link Mode.

The data type of *Option* is structure _sRESPONSE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Option	Response	Response monitoring and retry specifications	_sRESPONSE	---	---	---
isNonResp	No response	TRUE:Response is not required. FALSE:Response is required.	BOOL	Depends on data type.	---	FALSE
TimeOut	Timeout time	Timeout time 0: 2.0 s	UINT		0.1 s	20 (2.0 s)
Retry	Retry count	Retry count	USINT	0 to 15	Times	0

If the value of the Response Not Necessary Flag (*Option.isNonResp*) is FALSE and the response does not return within the value set for the Timeout Time (*Option.TimeOut*), the command is resent until there is a response. *Option.Retry* specifies the number of retries. The timeout time is *Option.TimeOut* multiplied by 0.1 s. However, if the value of *Option.TimeOut* is 0, the timeout time is 2.0 s. The default value of *Option.TimeOut* is 2.0 s. No responses are received for broadcast data. Also, the command is not resent.

Related System-defined Variables

Name	Meaning	Description
_Port_numUsingPort	Number of Used Ports	This is the number of ports that are currently used.
_Port_isAvailable	Network Communications Instruction Enabled Flag	TRUE: A port is available. FALSE: A port is not available.

Additional Information

- The command or response is sometimes lost during communications due to noise or other factors. You can increase reliability by setting *Option.Retry* to a value higher than 0 to perform retry processing when a response is not returned.
- To specify a serial port with the serial gateway function, specify the unit address of the serial port for *DstNetAdr.NetNo*. The unit addresses of the ports on Serial Communications Units are as follows:
  - Port 1
    Unit address = BYTE#16#80 + BYTE#16#04 × unit number (hex)

    Example for Unit Number 1
    BYTE#16#80+BYTE#16#04 × 1 = BYTE#16#84
  - Port 2
    Unit address = BYTE#16#81 + BYTE#16#04 × unit number (hex)

    Example for Unit Number 2
    BYTE#16#81+BYTE#16#04 × 2 = BYTE#16#89

Precautions for Correct Use

- The instruction is executed only when there is an available port. Therefore, use the system-defined variable *_Port_isAvailable* (Network Communications Instruction Enabled Flag) in an N.O. execution condition for the instruction.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- If the instruction is used in ST, make sure that the instruction is processed each task period as long as instruction execution continues. Otherwise, normal processing is sometimes not possible.
- The command is not sent if the value of *CmdSize* is 0. When the instruction is executed, the value of *Done* changes to TRUE.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *CommPort* is outside of the valid range.
  - The value of a member of *DstNetAdr* is outside of the valid range.
  - The value of *CmdSize* is outside of the valid range.
  - The value of a member of *Option* is outside of the valid range.
  - The value of *CmdSize* exceeds the size of *CmdDat[]*.
  - The response size exceeds the size of *RespDat[]*.
  - The value of *_Port_isAvailable* is FALSE.
  - Communications fail.

- For this instruction, expansion error code *ErrorIDEx* gives the communications response code. The values and meanings are listed in the following table. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0800.

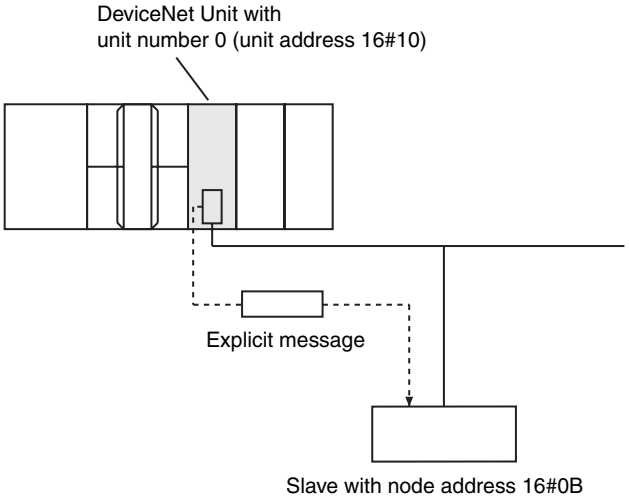
Value	Error	Correction
16#00000101	The local node is not part of the network.	Make the local node part of the network.
16#00000105	The IP address of the local node is out of range.	Set the rotary switches on the Serial Communications Unit correctly.
16#00000106	The IP address of the local node is also used by another node in the network.	Change one of the node addresses that are duplicated.
16#00000202	A Unit with the specified unit address does not exist at the destination.	Correctly set the unit address for the destination network address.
16#00000301	A Communications Controller Error occurred.	Refer to the operation manual for the relevant Unit and make suitable corrections.
16#00000304	The unit number setting is not correct.	Set the rotary switches on the Serial Communications Unit correctly.
16#00000401	The command that was sent is not supported.	Set the command string correctly.
16#00000402	The Unit model or version is not supported.	Check the Unit model and version.
16#00001001	The command is too long.	Set the command string correctly.
16#00001002	The command is too short.	Set the command string correctly.
16#00001003	The number of write elements that is specified in the command does not agree with the number of write data.	Specify the same number of write elements and write data.
16#00001004	The command format is incorrect.	Set the command string correctly.
16#0000110B	The response is too long.	Set the number of elements in the command string correctly.
16#0000110C	This is another parameter error.	Set the command string correctly.
16#00002202	The operating mode is wrong.	Check the operating mode.
16#00002502	There is an error in the part of memory for processing.	Transfer the correct data to memory.
16#00002503	The registered I/O Unit configuration does not agree with the physical Unit configuration.	Check the I/O Unit configuration.
16#00002504	There are too many local or remote I/O points.	Set the number of local and remote I/O points correctly.
16#00002505	An error occurred in a data transmission between the CPU Unit and a CPU Bus Unit.	Check the Unit and the Connecting Cable. After removing the error, execute a command to reset the error.
16#00002506	The same rack number, unit number, or I/O address is set more than once.	Correct the settings so that each number is unique.
16#00002507	An error occurred in a data transmission between the CPU Unit and an I/O Unit.	Check the Unit and the Connecting Cable. After removing the error, execute a command to reset the error.
16#00002509	There is an error in SYSMAC BUS/2 data transmission.	Check the Unit and the Connecting Cable. After removing the error, execute a command to reset the error.
16#0000250A	An error occurred in a CPU Bus Unit data transmission.	Check the Unit and the Connecting Cable. After removing the error, execute a command to reset the error.
16#0000250D	The same word setting is used more than once.	Set the I/O words correctly.
16#00002510	The end station setting is wrong.	Set the end station correctly.

Sample Programming

In this sample, the SendCmd instruction sends an explicit message via a DeviceNet Unit. This sample reads the vendor ID from the slave with node address 16#0B through the DeviceNet Unit with unit address 16#10.

The following communications specifications are used.

Item	Description
Unit address of DeviceNet Unit	16#10
Slave node address	16#0B
Service code	16#0E
Class ID	1
Instance ID	1
Attribute ID	1
Timeout time	2.0 s
Retry count	2



The contents of command array *SendDat[]* and response storage array *RecvDat[]* are as follows:

Command Array: BYTE array

Array element	Item	Content	Value
SendDat[0]	Command code	The command code to send an explicit message is 16#2801.	BYTE#16#28
SendDat[1]			BYTE#16#01
SendDat[2]	Slave node address	The node address is 16#0B.	BYTE#16#0B
SendDat[3]	Service code	The service code to read the value of a specified attribute (Get Attribute Single) is 16#0E.	BYTE#16#0E
SendDat[4]	Class ID	The class ID of the Identity object is 16#0001.	BYTE#16#00
SendDat[5]			BYTE#16#01
SendDat[6]	Instance ID	---	BYTE#16#00
SendDat[7]			BYTE#16#01
SendDat[8]	Attribute ID	The attribute ID of the vendor ID (Vendor ID) is 16#01.	BYTE#16#01

Response Storage Array: BYTE Array

Array element	Item	Content
RecvDat[0]	Command code	The command code to send an explicit message is 16#2801.
RecvDat[1]		
RecvDat[2]	Completion code	The completion code is 16#0000 for a normal end.
RecvDat[3]		
RecvDat[4]	Number of bytes received after the slave node address	4 bytes
RecvDat[5]		
RecvDat[6]	Slave node address	The node address is 16#0B for a normal end.
RecvDat[7]	Service code	The service code for a normal end is 16#8E.
RecvDat[8]	Vendor ID	Slave vendor ID.
RecvDat[9]		

Definitions of Global Variables

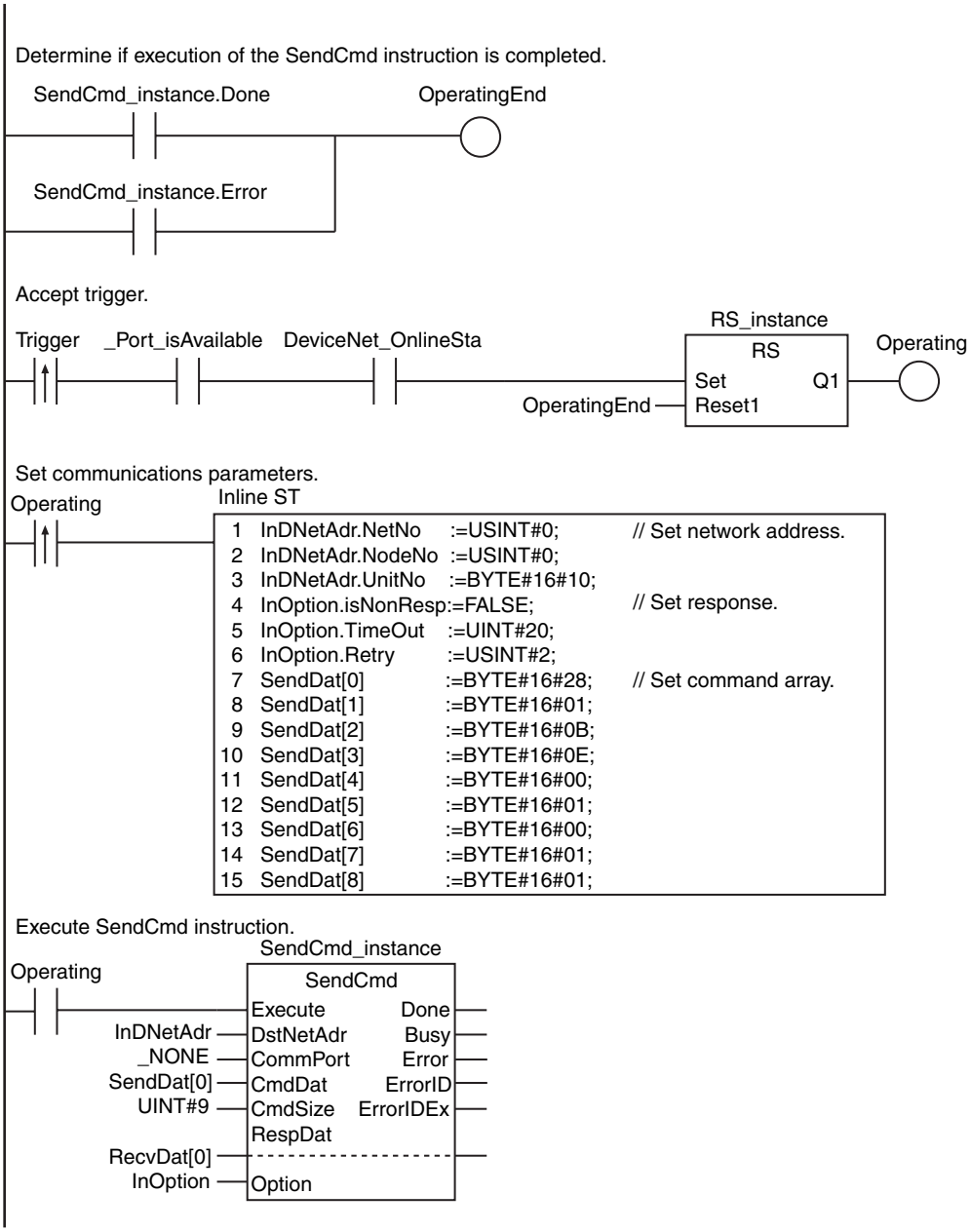
Global Variables

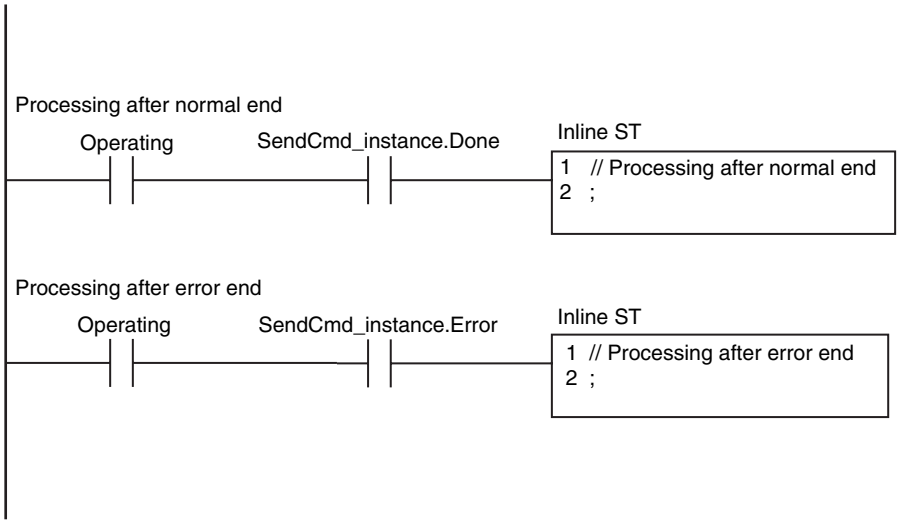
Name	Data type	AT specification	Comment
DeviceNet_OnlineSta	BOOL	IOBus://rack#0/slot#0/Unit2Sta/OnlineSta	Online

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	InDNetAdr	_sDNET_ADR	(NetNo:=0, NodeNo:=0, UnitNo:=16#0)	Destination network address
	InOption	_sRESPONSE	(isNonResp:=False, TimeOut:=0, Retry:=0)	Response
	SendDat	ARRAY[0..8] OF BYTE	[9(16#0)]	Send data
	RecvDat	ARRAY[0..9] OF BYTE	[10(16#0)]	Receive data
	RS_instance	RS		
	SendCmd_instance	SendCmd		

External Variables	Variable	Data type	Comment
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag




Processing after normal end
Operating
SendCmd_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
SendCmd_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started
	Operating	BOOL	False	Processing
	InDNetAdr	_sDNET_ADR	(NetNo:=0, NodeNo:=0, UnitNo:=16#0)	Destination network address
	InOption	_sRESPONSE	(isNonResp:=False, TimeOut:=0, Retry:=0)	Response
	SendDat	ARRAY[0..8] OF BYTE	[9(16#0)]	Send data
	RecvDat	ARRAY[0..9] OF BYTE	[10(16#0)]	Receive data
	SendCmd_instance	SendCmd		

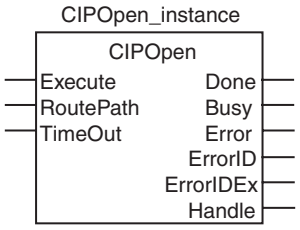
External Variables	Variable	Data type	Comment
	DeviceNet_OnlineSta	BOOL	Online
	_Port_isAvailable	BOOL	Network Communications Instruction Enabled Flag

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Port_isAvailable=TRUE)
        AND (DeviceNet_OnlineSta=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize SendCmd instruction.
IF (OperatingStart=TRUE) THEN
    SendCmd_instance(
        Execute:=FALSE,
        DstNetAdr:=InDNetAdr,
        CommPort:=_NONE,
        CmdDat :=SendDat[0],
        CmdSize :=UINT#9,
        RespDat:=RecvDat[0],
        Option :=InOption);
    InDNetAdr.NetNo    :=USINT#0;          // Set network address.
    InDNetAdr.NodeNo   :=USINT#0;
    InDNetAdr.UnitNo   :=BYTE#16#10;
    InOption.isNonResp:=FALSE;             // Set response.
    InOption.TimeOut   :=UINT#20;
    InOption.Retry     :=USINT#2;
    SendDat[0]         :=BYTE#16#28;       // Set command array.
    SendDat[1]         :=BYTE#16#01;
    SendDat[2]         :=BYTE#16#0B;
    SendDat[3]         :=BYTE#16#0E;
    SendDat[4]         :=BYTE#16#00;
    SendDat[5]         :=BYTE#16#01;
    SendDat[6]         :=BYTE#16#00;
    SendDat[7]         :=BYTE#16#01;
    SendDat[8]         :=BYTE#16#01;
    OperatingStart     :=FALSE;
END_IF;
```

```
 // Execute SendCmd instruction.
IF (Operating=TRUE) THEN
    SendCmd_instance(
        Execute  :=TRUE,
        DstNetAdr:=InDNetAdr,
        CommPort:=_NONE,
        CmdDat :=SendDat[0],
        CmdSize :=UINT#9,
        RespDat:=RecvDat[0],
        Option :=InOption);

    IF (SendCmd_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;

    IF (SendCmd_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;
```

CIPOpen

The CIPOpen instruction opens a CIP class 3 connection with the specified remote node.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPOpen	Open CIP Class 3 Connection	FB		CIPOpen_instance(Execute, RoutePath, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, Handle);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2 s)
Handle	Handle	Output	Handle	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				OK
TimeOut							OK													
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			

Function

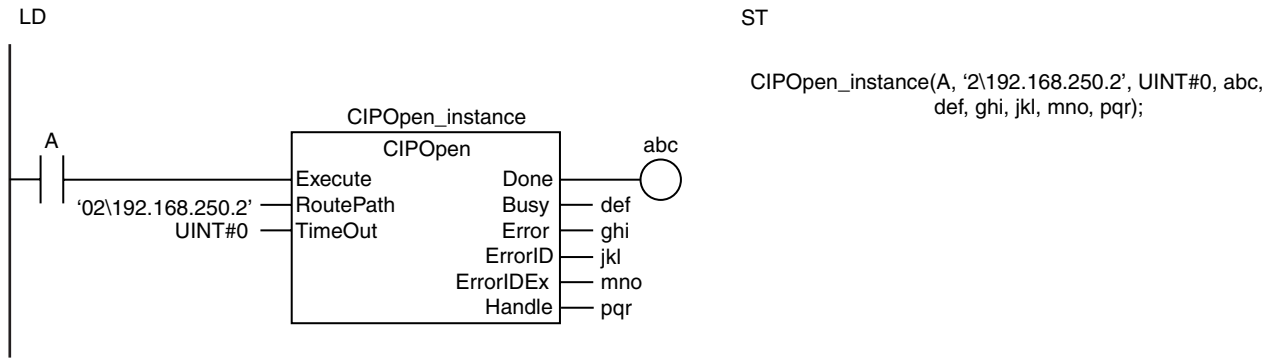
The CIPOpen instruction opens a CIP class 3 connection with another Controller on a CIP network. The other Controller is specified with route path *RoutePath*. The handle *Handle* is output when the connection is open.

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed. The timeout time is reset when the CIPRead, CIPWrite, or CIPSend instruction is executed.

The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_ HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

The following figure shows a programming example.



Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must execute this instruction before you execute CIPRead, CIPWrite, or CIPSend.
- For this instruction, the first timeout time after a connection is established is 10 s even if the value of *TimeOut* is set to less than 100 (10 s).
- Use the CIPClose instruction to close connections that were opened with the CIPOpen instruction.
- Even if the connection times out, the handle created by this instruction will remain. Always use the CIPClose instruction to close the connection.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- You can create a maximum of 32 handles at the same time.
- You can use this instruction only through a built-in EtherNet/IP port on an NJ-series CPU Unit or a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- This instruction does not use *ErrorIDEx*.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The text string in *RoutePath* does not end in a NULL character.
  - The value of *TimeOut* is outside of the valid range.

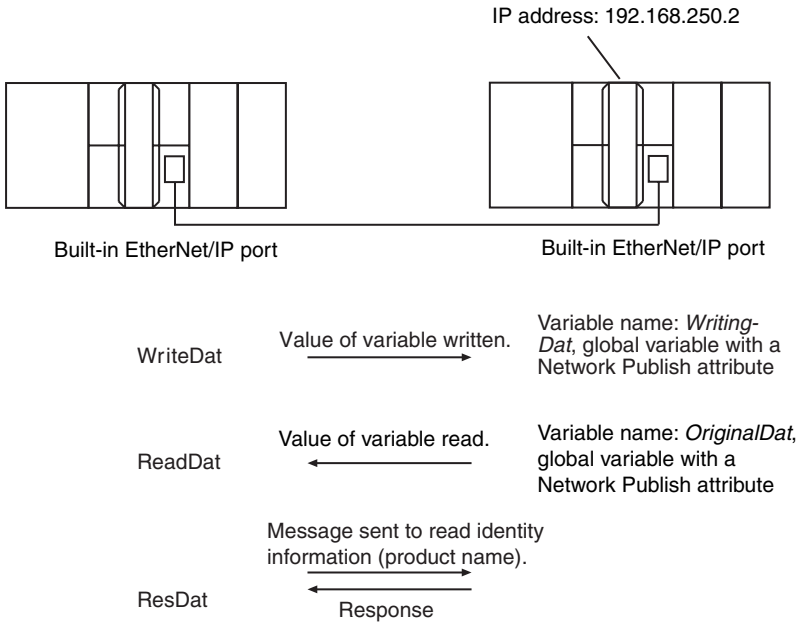
Sample Programming

This sample uses CIP class 3 messages to write a variable, read a variable, and send a message. The Controllers are connected to an EtherNet/IP network. The IP address of the remote node is 192.168.250.2.
The following procedure is used.

1. The CIPOpen instruction is used to open a class 3 connection. The timeout time is 2 s.
2. The CIPWrite instruction is used to write the value of a variable at a remote node. The variable name at the remote node is *WritingDat* and the contents of the *WriteDat* is written to it. *WritingDat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.
3. The CIPRead instruction is used to read the value of a variable at a remote node. The value of the variable *OriginalDat* at the other node is read and the read value is stored in the *ReadDat* variable. *OriginalDat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.
4. The CIPSend instruction is used to send an explicit message to a remote node. The contents of the message is to read identity information (product name). The class ID, instance ID, attribute ID, and service code are as follows: The response data is stored in the *ResDat* variable.

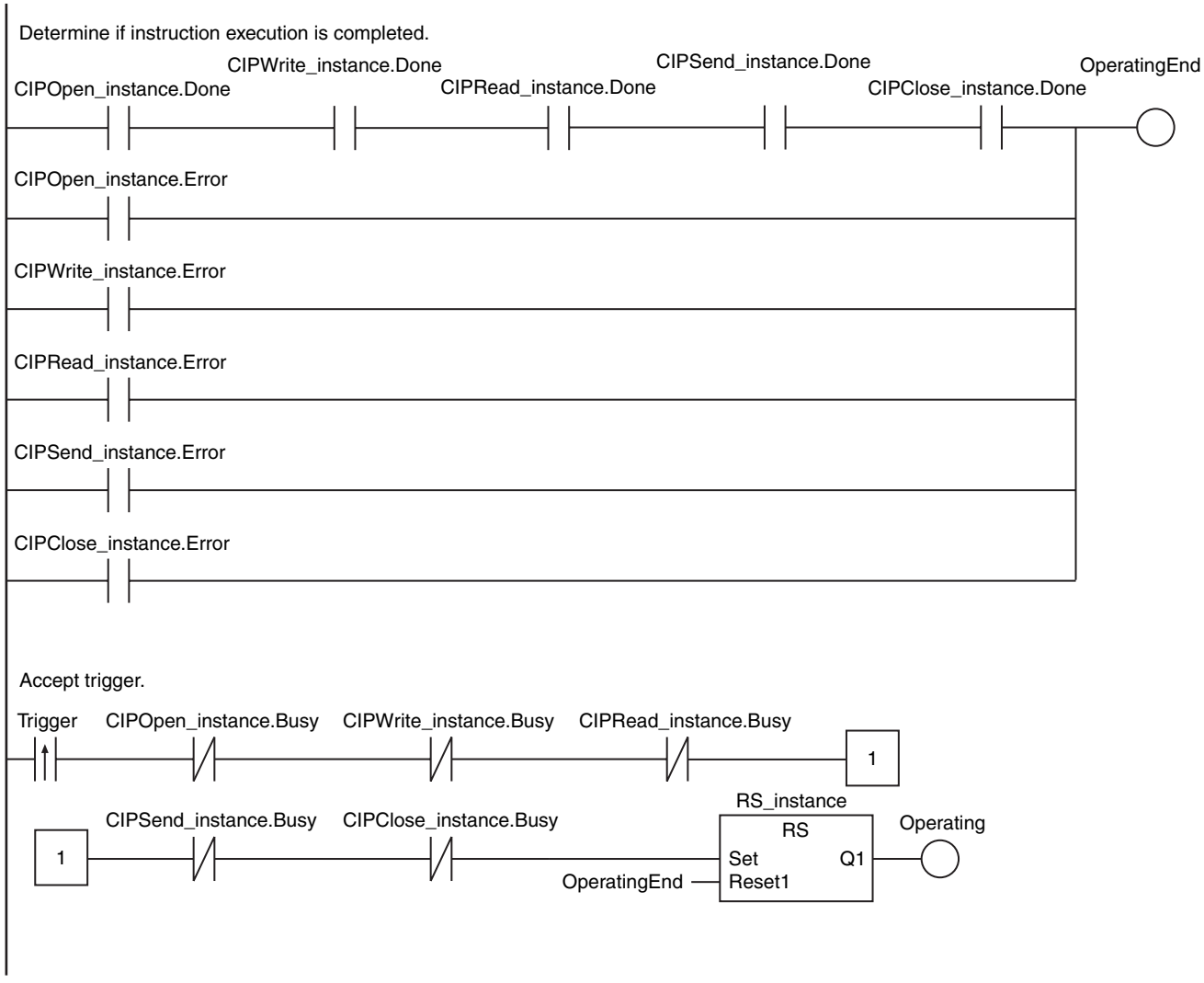
Item	Value
Class ID	1
Instance ID	1
Attribute ID	7
Service code	16#0E

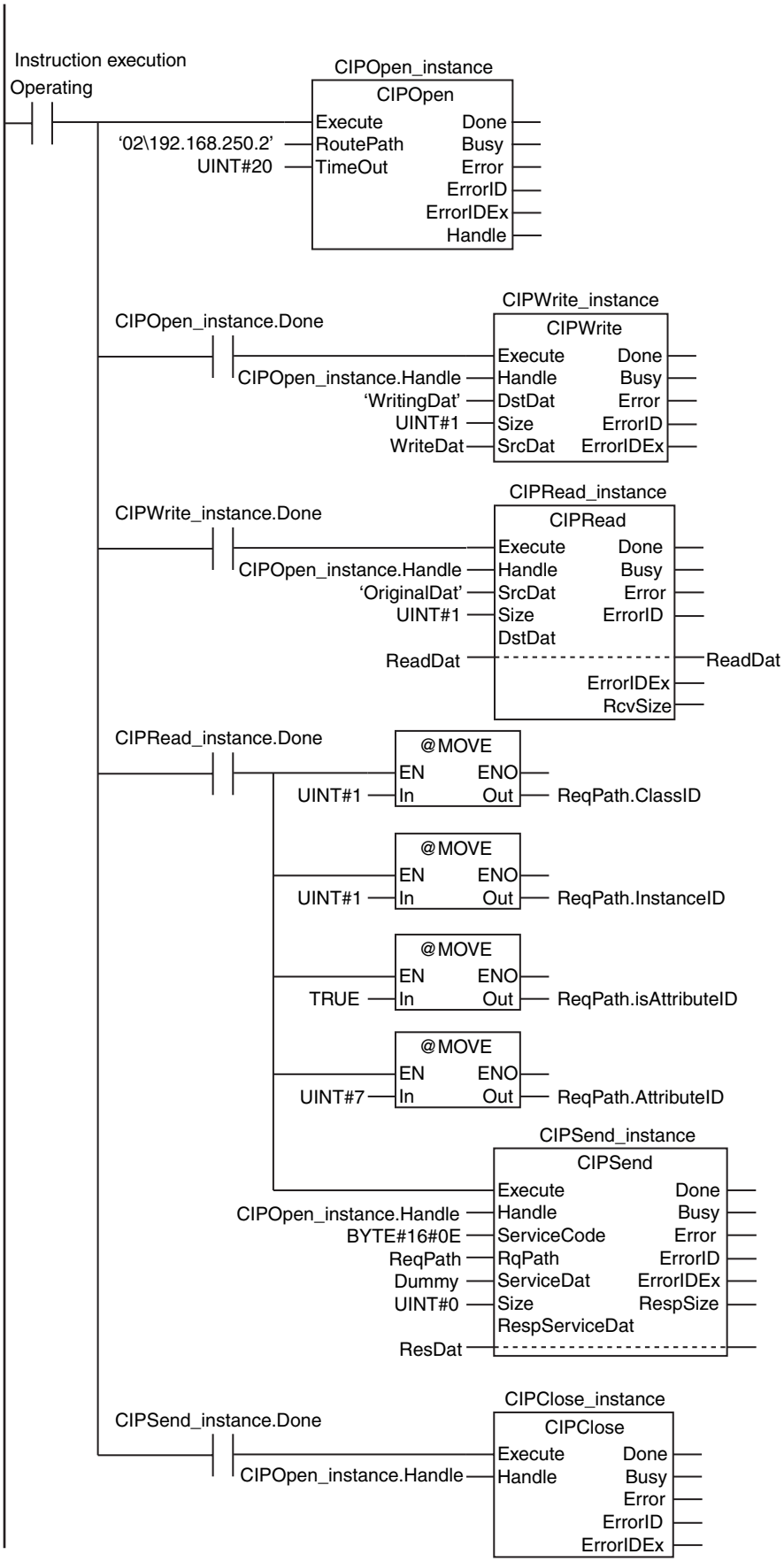
5. The CIPClose instruction is used to close the class 3 connection.

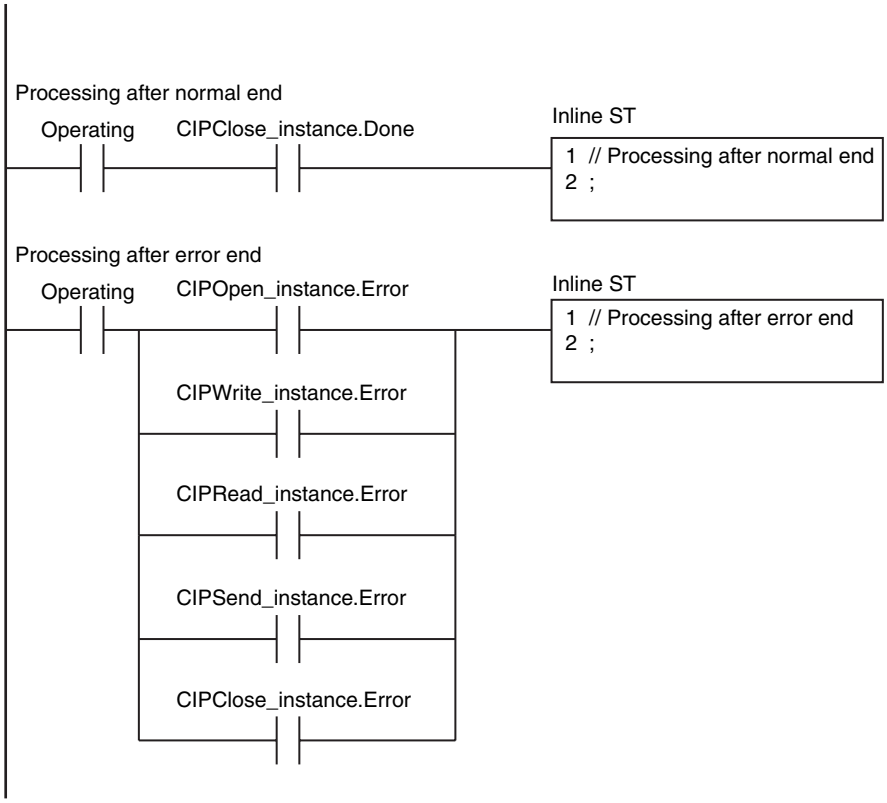


LD

Variable	Data type	Initial value	Comment
OperatingEnd	BOOL	False	Processing completed
Trigger	BOOL	False	Execution condition
Operating	BOOL	False	Processing
WriteDat	INT	1234	Write data
ReadDat	INT	0	Read data
ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=False, AttributeID:=0)	Request path
ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
Dummy	BYTE	16#0	Dummy
RS_instance	RS		
CIPOpen_instance	CIPOpen		
CIPWrite_instance	CIPWrite		
CIPRead_instance	CIPRead		
CIPSend_instance	CIPSend		
CIPClose_instance	CIPClose		




Instruction execution
Operating
CIPOpen_instance
CIPOpen
Execute
RoutePath
TimeOut
Done
Busy
Error
ErrorID
ErrorIDEx
Handle
'02\192.168.250.2'
UINT#20
CIPOpen_instance.Done
CIPWrite_instance
CIPWrite
Execute
Handle
DstDat
Size
SrcDat
Done
Busy
Error
ErrorID
ErrorIDEx
CIPOpen_instance.Handle
'WritingDat'
UINT#1
WriteDat
CIPWrite_instance.Done
CIPRead_instance
CIPRead
Execute
Handle
SrcDat
Size
DstDat
Done
Busy
Error
ErrorID
ErrorIDEx
RcvSize
CIPOpen_instance.Handle
'OriginalDat'
UINT#1
ReadDat
ReadDat
CIPRead_instance.Done
@MOVE
EN
In
ENO
Out
UINT#1
ReqPath.ClassID
ReqPath.InstanceID
TRUE
ReqPath.isAttributeID
UINT#7
ReqPath.AttributeID
CIPSend_instance
CIPSend
Execute
Handle
ServiceCode
RqPath
ServiceDat
Size
RespServiceDat
Done
Busy
Error
ErrorID
ErrorIDEx
RespSize
CIPOpen_instance.Handle
BYTE#16#0E
ReqPath
Dummy
UINT#0
ResDat
CIPClose_instance
CIPClose
Execute
Handle
Done
Busy
Error
ErrorID
ErrorIDEx
CIPSend_instance.Done
CIPOpen_instance.Handle


Processing after normal end
Operating
CIPClose_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
CIPOpen_instance.Error
Inline ST
1 // Processing after error end
2 ;
CIPWrite_instance.Error
CIPRead_instance.Error
CIPSend_instance.Error
CIPClose_instance.Error

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	DoCIPTrigger	BOOL	False	Processing
	Stage	INT	0	Stage change
	WriteDat	INT	0	Write data
	ReadDat	INT	0	Read data
	ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=False, AttributeID:=0)	Request path
	ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
	Dummy	BYTE	16#0	Dummy
	CIPOpen_instance	CIPOpen		
	CIPWrite_instance	CIPWrite		
	CIPRead_instance	CIPRead		
	CIPSend_instance	CIPSend		
	CIPClose_instance	CIPClose		

External Variables	Variable	Constant	Data type	Comment
	_EIP_EtnOnlineSta	☑	BOOL	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoCIPTrigger=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoCIPTrigger:=TRUE;
    Stage      :=INT#1;
    CIPOpen_instance(Execute:=FALSE);      // Initialize instance.
    CIPWrite_instance(
        Execute          :=FALSE,          // Initialize instance.
        SrcDat           :=WriteDat);      // Dummy
    CIPRead_instance(                      // Initialize instance.
        Execute          :=FALSE,          // Dummy
        DstDat           :=ReadDat);       // Dummy
    CIPSend_instance(
        Execute          :=FALSE,          // Initialize instance.
        ServiceDat       := Dummy,         // Dummy
        RespServiceDat:=ResDat);           // Dummy
    CIPClose_instance(Execute:=FALSE);     // Initialize instance.
END_IF;


IF (DoCIPTrigger=TRUE) THEN
    CASE Stage OF
    1 :                                    // Open CIP class 3 connection.
        CIPOpen_instance(
            Execute    :=TRUE,
            TimeOut    :=UINT#20,          // Timeout time: 2.0 s
            RoutePath:='02\192.168.250.2');// Route path

        IF (CIPOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;                  // Normal end
        ELSIF (CIPOpen_instance.Error=TRUE) THEN
            Stage:=INT#10;                 // Error end
        END_IF;

    2 :                                    // Request writing value of variable.
        CIPWrite_instance(
            Execute:=TRUE,
            Handle  :=CIPOpen_instance.Handle,  // Handle
            DstDat  :='WritingDat',             // Source variable name
            Size    :=UINT#1,                   // Number of elements to write
            SrcDat  :=WriteDat);                // Write data

        IF (CIPWrite_instance.Done=TRUE) THEN
            Stage:=INT#3;                  // Normal end
        ELSIF (CIPWrite_instance.Error=TRUE) THEN
            Stage:=INT#20;                 // Error end
        END_IF;
```

```
    3 :                                    // Request reading value of variable.
        CIPRead_instance(
            Execute:=TRUE,
            Handle  :=CIPOpen_instance.Handle,     // Handle
            SrcDat  :='OriginalDat',               // Source variable name
            Size    :=UINT#1,                      // Number of elements to read
            DstDat  :=ReadDat);                    // Read data

        IF (CIPRead_instance.Done=TRUE) THEN
            Stage:=INT#4;                          // Normal end
        ELSIF (CIPRead_instance.Error=TRUE) THEN
            Stage:=INT#30;                         // Error end
        END_IF;

    4 :                                    // Send message
        ReqPath.ClassID      :=UINT#01;
        ReqPath.InstanceID   :=UINT#01;
        ReqPath.isAttributeID:=TRUE;
        ReqPath.AttributeID  :=UINT#07;
        CIPSend_instance(
            Execute          :=TRUE,
            Handle           :=CIPOpen_instance.Handle,  // Handle
            ServiceCode      :=BYTE#16#0E,               // Service code
            RqPath           :=ReqPath,                  // Request path
            ServiceDat       :=Dummy,                    // Service data
            Size             :=UINT#0,                   // Number of elements
            RespServiceDat:=ResDat);                     // Response data

        IF (CIPSend_instance.Done=TRUE) THEN
            Stage:=INT#5;                          // Normal end
        ELSIF (CIPSend_instance.Error=TRUE) THEN
            Stage:=INT#40;                         // Error end
        END_IF;

    5 :                                    // Request closing CIP class 3 connection.
       CIPClose_instance(
            Execute :=TRUE,
            Handle  :=CIPOpen_instance.Handle);    // Handle

        IF (CIPClose_instance.Done=TRUE) THEN
            Stage:=INT#0;
        ELSIF (CIPClose_instance.Error=TRUE) THEN
            Stage:=INT#50;
        END_IF;

    0:                                  // Processing after normal end
        DoCIPTrigger:=FALSE;
        Trigger      :=FALSE;

    ELSE                                // Processing after error end
        DoCIPTrigger:=FALSE;
        Trigger      :=FALSE;
    END_CASE;
END_IF;
```

CIPRead

The CIPRead instruction uses a class 3 explicit message to read the value of a variable in another Controller on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPRead	Read Variable Class 3 Explicit	FB	CIPRead_instance CIPRead Execute Done Handle Busy SrcDat Error Size ErrorID DstDat ErrorIDEx RcvSize	CIPRead_instance(Execute, Handle, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen instruction	---	---	---
SrcDat	Source variable name		Name of variable to read in other Controller	Depends on data type.		"
Size	Number of elements to read		Number of elements to read	0 to 1988		1
DstDat	Read data	In-out	Read data value	Depends on data type.	---	---
RcvSize	Read data size	Output	Read data size	0 to 1988	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			
SrcDat																				OK
Size							OK													
DstDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, structure, structure member, or union member can also be specified.*																			
RcvSize							OK													

Function

The CIPRead instruction reads the value of the network variable specified with source variable name *SrcDat* from another Controller on a CIP network. The other Controller is specified with *Handle*.
The read data value is stored in *DstDat.*
Size specifies the number of elements to read. If *SrcDat* is an array, specify the number of elements to read with *Size*. If *SrcDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is read regardless of whether *SrcDat* is an array or not.
When the read operation is completed, the number of bytes of the data that was read is assigned to read data size *RcvSize*. You can read a maximum of 1,988 bytes of data.

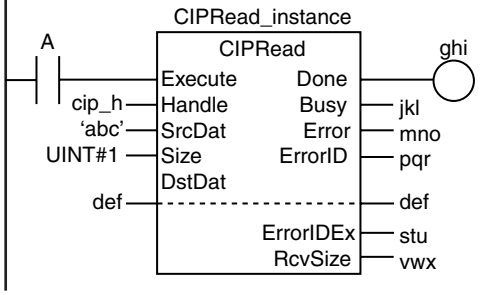
The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_ HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

The following example reads the value of variable *abc*. The read data is stored in variable *def* and the value of variable *vwx* changes to UINT#1.

LD



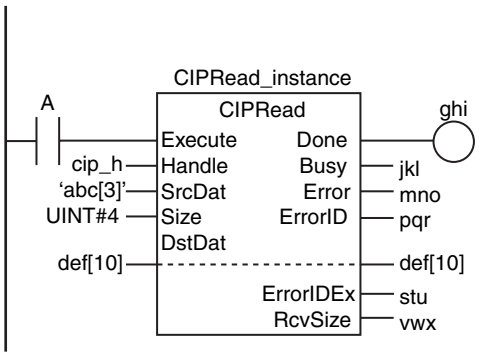
ST

```
CIPRead_instance(A, cip_h, 'abc', UINT#1,
                 def, ghi, jkl, mno, pqr,
                 stu, vwx);
```

Reading Arrays

To read array data, pass a subscripted array element to *ScrDat* as the parameter. Also pass a subscripted array element to *DstDat* as the parameter. The following example reads the four array variable elements *abc[3]* to *abc[6]* and stores the results in array variable elements *def[10]* to *def[13]*. The value of variable *vwx* will be UINT#4.

LD



ST

```
CIPRead_instance(A, cip_h, 'abc[3]', UINT#4,
                 def[10], ghi, jkl, mno, pqr,
                 stu, vwx);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Execute the CIPOpen instruction to obtain the value for *Handle* before you execute this instruction.
- This instruction can be used only through the built-in EtherNet/IP ports on NJ-series CPU Units.
- If a variable is read from an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to read data. To read specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot specify an address in local memory for CJ-series Units directly to store data. To store data in specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *DstDat*.
- The characters that can be used in *SrcDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and '_' (underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with "P_"

- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *Handle.Handle* is outside of the valid range.
  - The value of *Size* is outside of the valid range.
  - The value of *DstDat* is outside of the valid range.

- The value of *RcvSize* is outside of the valid range.
- For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

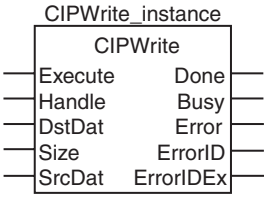
Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#05000000	The specified source variable does not exits on the other Controller.
16#0C008010	The specified source variable is being downloaded.
16#0C008011	
16#11000000	The value of Size is exceeds the data size that can currently be read.
16#20008017	The specified source variable is not an array and the number of elements to read is not 1.
16#20008018	The specified source variable is an array and the number of elements to read exceeds the number of elements in the array.
16#26000000	The specified destination variable contains only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-684).

CIPWrite

The CIPWrite instruction uses a class 3 explicit message to write the value of a variable in another Controller on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPWrite	Write Variable Class 3 Explicit	FB		CIPWrite_instance(Execute, Handle, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen instruction	---	---	---
DstDat	Destination variable name		Name of variable to write in another Controller	Depends on data type.	---	"
Size	Number of elements to write		Number of elements to write	0 to 1980		1
SrcDat	Source data		Data value to write	Depends on data type.		*

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			
DstDat																				OK
Size							OK													
SrcDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
Handle	An enumeration, array, structure, structure member, or union member can also be specified.*																			

* You cannot specify a STRING array.

Function

The CIPWrite instruction writes the value of the network variable specified with destination variable name *DstDat* at another Controller on a CIP network. The other Controller is specified with *Handle*. The content of source data *ScrDat* is written.

Size specifies the number of elements to write. If *DstDat* is an array, specify the number of elements to write with *Size*. If *DstDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is written regardless of whether *DstDat* is an array or not.

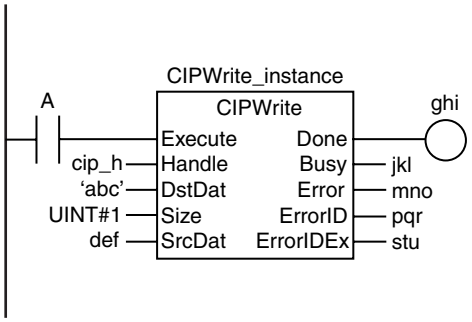
The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_ HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

The following example writes variable *abc*. The contents of variable *def* is written to variable *abc*.

LD



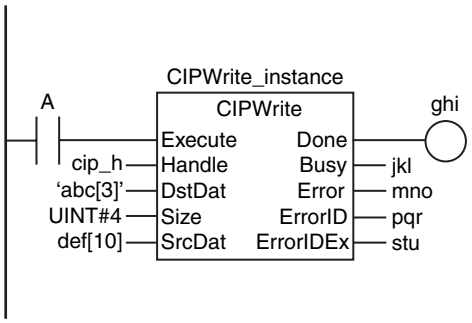
ST

```
CIPWrite_instance(A, cip_h, 'abc', UINT#1, def,
                  ghi, jkl, mno, pqr, stu);
```

Writing Arrays

To write array data, pass a subscripted array element to *DstDat* as the parameter. Also pass a subscripted array element to *SrcDat* as the parameter. The following example stores the contents of array variable elements *def[10]* to *def[13]* in the four array variable elements *abc[3]* to *abc[6]*.

LD



ST

```
CIPWrite_instance(A, cip_h, 'abc[3]', UINT#4, def,
                  ghi[10], jkl, mno, pqr, stu);
```

Maximum Write Data Size

The maximum size of the data that you can write depends on the data type and variable name that are specified for *DstDat*, as given in the following table.

Maximum write data size [bytes] = Base size − Size of variable name of *DstDat*

Item in above formula	Meaning
Base size	• Data type of variable specified for *DstDat* is a structure or STRING: 1,986 bytes • Other data types: 1,988 bytes

Item in above formula	Meaning
Size of variable name of *DstDat*	• The size of the variable name is calculated as the total bytes for the ASCII characters in all structure levels plus two times the number of levels. • If the number of bytes of ASCII characters in a level is an odd number, add 1. • If a level in the structure is an array, add four times the number of dimensions in the array. • Periods and commas in the structure and arrays are not included in the variable name size. Example 1: When the Variable Name of *DstDat* Is *aaa.bbbbb[1,2,3].cc* • The text string “aaa” in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • The text string “bbbbb[1,2,3]” in the second level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • Also *bbbbb[1,2,3]* is a three-dimensional array, so 3 times 4, or 12, is added to make 18 bytes. • The text string “cc” in the third level is 2 bytes. It is an even number, so 2 bytes is used in the calculation. • If we add the number of levels 3 times 2, or 6, to 4 bytes for the first level, 18 bytes for the second level, and 2 bytes for the third level, the size of the variable name come to 30 bytes. Example 2: When the Variable Name of *DstDat* Is *val* • The text string “val” in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 6 bytes. Example 3: When the Variable Name of *DstDat* Is *array[8]*. • The text string “array” in the first level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • It is a one-dimensional array. Therefore, 1 times 4, or 4, is added. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 12 bytes.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Execute the CIPOpen instruction to obtain the value for *Handle* before you execute this instruction.

- You can use this instruction only through a built-in EtherNet/IP port on an NJ-series CPU Unit or a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to write data. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot directly specify an address in local memory for CJ-series Units. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *SrcDat*.
- The characters that can be used in *DstDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and '_'(underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with "P_"

- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *Handle.Handle* is outside of the valid range.
  - The value of *Size* is outside of the valid range.
  - The value of *SrcDat* is outside of the valid range.
  - For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

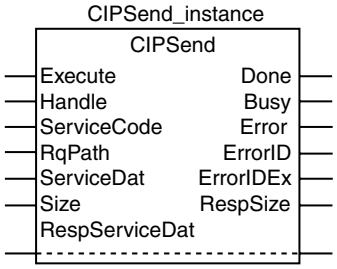
Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#05000000	The specified source variable does not exits on the other Controller.
16#0C008010	The specified source variable is being downloaded.
16#0C008011	
16#1F000102	• The specified destination variable has a Constant attribute, so it cannot be written. • The write data does not agree with the number of write elements.
16#20008017	The specified destination variable is not an array and the number of elements to write is not 1.
16#20008018	The specified destination variable is an array and the number of elements to write exceeds the number of elements in the array.
16#20008028	• The specified destination variable is an enumeration and the write data is not the value of an enumerator. • The specified destination variable has a Range Specification attribute and the write data is out of range.
16#26000000	The specified destination variable contains only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-684).

CIPSend

The CIPSend instruction sends a class 3 CIP message to a specified device on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPSend	Send Explicit Message Class 3	FB		CIPSend_instance(Execute, Handle, ServiceCode, RqPath, ServiceDat, Size, RespService-Dat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen instruction	---	---	---
ServiceCode	Service code		Service code	Depends on data type.		---
RqPath	Request path		Request path (class ID, instance ID, attribute ID)	---		
ServiceDat	Service data		Service data to send	Depends on data type.		
Size	Number of elements to send		Number of elements to send			*
RespService-Dat	Response data	In-out	Response data	Depends on data type.	---	---
RespSize	Response size	Output	Response data size	Depends on data type.	Bytes	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			
ServiceCode		OK																		
RqPath	Refer to *Function* for details on the structure _sREQUEST_PATH.																			
ServiceDat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
Size							OK													
RespService-Dat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
RespSize							OK													

Function

The CIPSend instruction sends service data *ServiceDat* for the service specified with service code *ServiceCode* as a class 3 explicit message.
The destination is specified with handle *Handle*.
RqPath specifies the request path.

Size specifies the number of elements to send. If *ServiceDat* is an array, specify the number of elements to send with *Size*. If *ServiceDat* is not an array, always specify 1 for *Size*. If no service data is required, set *Size* to 0.
The response data received later is stored in *RespServiceDat*. The number of bytes of the response data is stored in *RespSize*.

The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

The data type of *RqPath* is structure _sREQUEST_PATH. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
RqPath	Request path	Request path (class ID, instance ID, attribute ID)	_sREQUEST_PATH	---	---	---
ClassID	Class ID	Class ID	UINT	Depends on data type.	---	0
InstanceID	Instance ID	Instance ID	UINT			0
isAttributeID	Attribute usage	TRUE:Attribute ID used. FALSE:Attribute ID not used.	BOOL			FALSE
AttributeID	Attribute ID	Attribute ID	UINT			0

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*. The meaning and values of *ErrorIDEx* depend on the remote node. Refer to the manual for the remote node.

Sending and Receiving Arrays

If *ServiceDat* or *RespServiceDat* is an array, pass a subscripted array element to it as the parameter.

Maximum Read/Write Data Size

You can read a maximum of 1,990 bytes of data. The maximum size of the data that you can write depends on whether there is a request path attribute, as given below.

Maximum write data size [bytes] = Base size − Attribute usage

Item in above formula	Meaning
Base size	1,992 bytes
Attribute usage	Attribute ID used: 12 bytes Attribute ID not used: 8 bytes

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Execute the CIPOpen instruction to obtain the value for *Handle* before you execute this instruction.
- You can use this instruction only through a built-in EtherNet/IP port on an NJ-series CPU Unit or a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *Handle.Handle* is outside of the valid range.
  - The value of *ServiceCode* is outside of the valid range.
  - The value of a member of *RqPath* is outside of the valid range.
  - The value of *Size* is outside of the valid range.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-684).

CIPClose

The CIPClose instruction closes the CIP class 3 connection to the specified handle.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPClose	Close CIP Class 3 Connection	FB	CIPClose_instance CIPClose Execute / Handle Done / Busy / Error / ErrorID / ErrorIDEx	CIPClose_instance(Execute, Handle, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen instruction	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			

Function

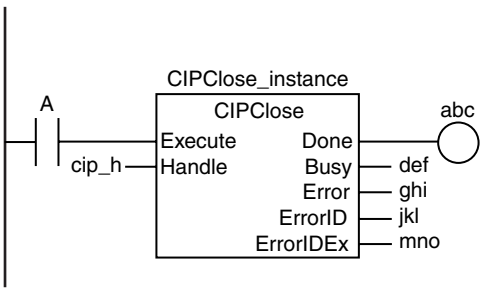
The CIPClose instruction closes the CIP class 3 connection specified with the handle *Handle*.

The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

The following figure shows a programming example.

LD



ST

```
CIPClose_instance(A, cip_h, abc, def,
                  ghi, jkl, mno);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Specify the handle that was obtained with the CIPOpen instruction for *Handle*.
- You can use this instruction only through a built-in EtherNet/IP port on an NJ-series CPU Unit or a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- This instruction does not use *ErrorIDEx*.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *Handle.Handle* is outside of the valid range.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-684).

CIPUCMMRead

The CIPUCMMRead instruction uses a UCMM explicit message to read the value of a variable in another Controller on the specified CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPUCMM Read	Read Variable UCMM Explicit	FB	CIPUCMMRead_instance CIPUCMMRead Execute / Done RoutePath / Busy TimeOut / Error SrcDat / ErrorID Size / ErrorIDEx DstDat RcvSize	CIPUCMMRead_instance(Execute, RoutePath, TimeOut, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	----	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2 s)
SrcDat	Source variable name		Name of variable to read in other Controller	Depends on data type.	---	"
Size	Number of elements to read		Number of elements to read	0 to 496		1
DstDat	Read data	In-out	Read data value	Depends on data type.	---	---
RcvSize	Read data size	Output	Read data size	0 to 496	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath								OK												OK
TimeOut							OK													
SrcDat																				OK
Size							OK													
DstDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, structure, structure member, or union member can also be specified.*																			
RcvSize									OK											

* You cannot specify a STRING array.

Function

The CIPUCMMRead instruction reads the value of the network variable specified with source variable name *SrcDat* from another Controller on a CIP network. The other Controller is specified with route path *RoutePath*.
The read data value is stored in *DstDat.*

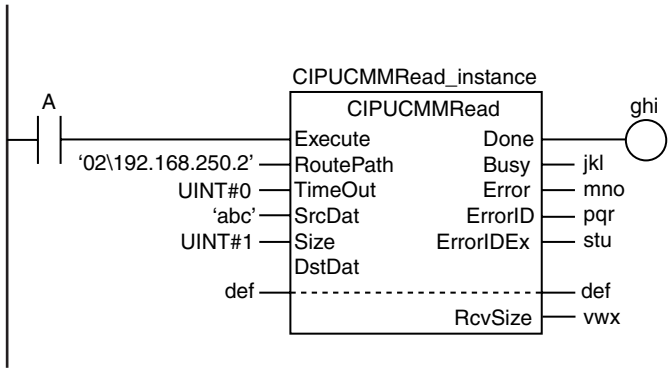
Size specifies the number of elements to read. If *SrcDat* is an array, specify the number of elements to read with *Size*. If *SrcDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is read regardless of whether *SrcDat* is an array or not.
When the read operation is completed, the number of bytes of the data that was read is assigned to read data size *RcvSize*. You can read a maximum of 496 bytes of data.

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed.

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

The following example reads the value of variable *abc*. The read data is stored in variable *def* and the value of variable *vwx* changes to UINT#1.

LD



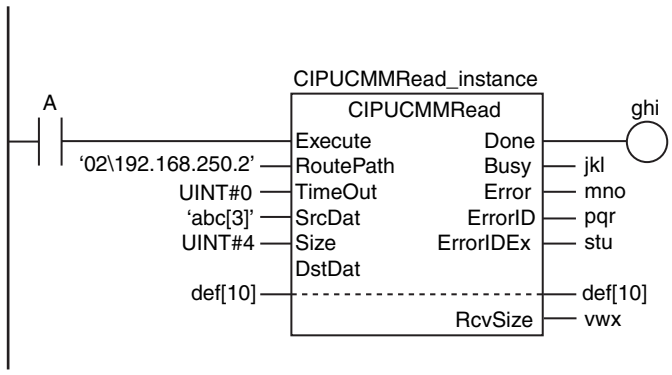
ST

```
CIPUCMMRead_instance(A, '2\192.168.250.2',
                     UINT#0, 'abc', UINT#1,
                     def, ghi, jkl, mno, pqr,
                     stu, vwx);
```

Reading Arrays

To read array data, pass a subscripted array element to *ScrDat* as the parameter. Also pass a subscripted array element to *DstDat* as the parameter. The following example reads the four array variable elements *abc[3]* to *abc[6]* and stores the results in array variable elements *def[10]* to *def[13]*. The value of variable *vwx* will be UINT#4.

LD



ST

```
CIPUCMMRead_instance(A, '2\192.168.250.2',
                     UINT#0, 'abc[3]', UINT#4,
                     def[10], ghi, jkl, mno, pqr,
                     stu, vwx);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only through the built-in EtherNet/IP ports on NJ-series CPU Units.
- If a variable is read from an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to read data. To read specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot specify an address in local memory for CJ-series Units directly to store data. To store data in specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *DstDat*.
- If the variable that is read is a user-defined structure, you can read a maximum of 492 bytes.
- The characters that can be used in *SrcDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and '_' (underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with "P_"

- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The text string in *RoutePath* does not end in a NULL character.
  - The value of *TimeOut* is outside of the valid range.

- The value of *Size* is outside of the valid range.
- The value of *DstDat* is outside of the valid range.
- The value of *RcvSize* is outside of the valid range.

- For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#05000000	The specified source variable does not exist on the other Controller.
16#0C008010	The specified source variable is being downloaded.
16#0C008011	
16#11000000	The value of Size is exceeds the data size that can currently be read.
16#20008017	The specified source variable is not an array and the number of elements to read is not 1.
16#20008018	The specified source variable is an array and the number of elements to read exceeds the number of elements in the array.
16#26000000	The specified destination variable contains only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPUCMMSend instruction (page 2-716).

CIPUCMMWrite

The CIPUCMMWrite instruction uses a UCMM explicit message to write the value of a variable in another Controller on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPUCMM Write	Write Variable UCMM Explicit	FB	CIPUCMMWrite_instance CIPUCMMWrite Execute / Done RoutePath / Busy TimeOut / Error DstDat / ErrorID Size / ErrorIDEx SrcDat	CIPUCMMWrite_instance(Execute, RoutePath, TimeOut, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2 s)
DstDat	Destination variable name		Name of variable to write in another Controller	Depends on data type.	---	"
Size	Number of elements to write		Number of elements to write	0 to 488		1
SrcDat	Source data		Data value to write	Depends on data type.		*

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				OK
TimeOut							OK													
DstDat																				OK
Size							OK													
SrcDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, structure, structure member, or union member can also be specified.*																			

* You cannot specify a STRING array.

Function

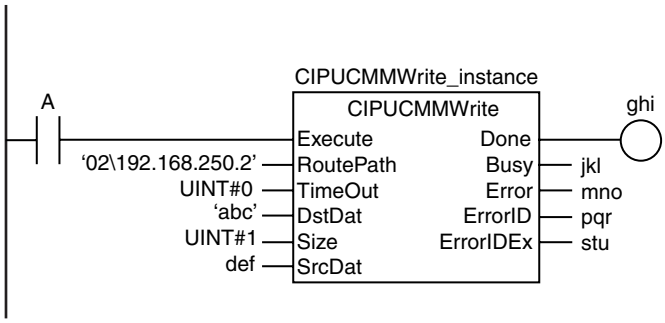
The CIPUCMMWrite instruction writes the value of the network variable specified with destination variable name *DstDat* at another Controller on a CIP network. The other Controller is specified with route path *RoutePath*.
The content of source data *ScrDat* is written.
Size specifies the number of elements to write. If *DstDat* is an array, specify the number of elements to write with *Size*. If *DstDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is written regardless of whether *DstDat* is an array or not.

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed.

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

The following example writes variable *abc*. The contents of variable *def* is written to variable *abc*.

LD



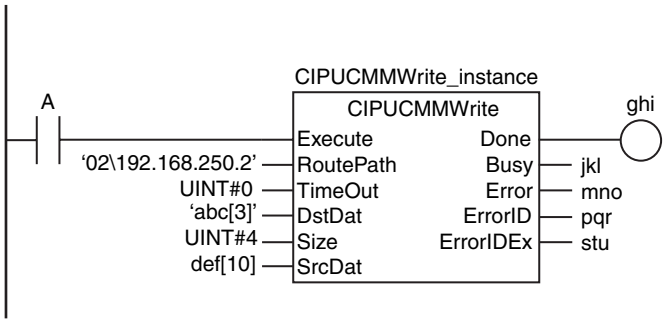
ST

```
CIPUCMMWrite_instance(A, '2\192.168.250.2',
                      UINT#0, 'abc', UINT#1,
                      def, ghi, jkl, mno, pqr,
                      stu);
```

Writing Arrays

To write array data, pass a subscripted array element to *DstDat* as the parameter. Also pass a subscripted array element to *SrcDat* as the parameter. The following example stores the contents of array variable elements *def[10]* to *def[13]* in the four array variable elements *abc[3]* to *abc[6]*.

LD



ST

```
CIPUCMMWrite_instance(A, '2\192.168.250.2',
                      UINT#0, 'abc[3]', UINT#4,
                      def[10], ghi, jkl, mno, pqr,
                      stu);
```

Maximum Write Data Size

The maximum size of the data that you can write depends on the data type and variable name that are specified for *DstDat* and the route path, as given in the following table.

Maximum write data size [bytes] = Base size − Size of variable name of *DstDat* − Path information size

Item in above formula	Meaning
Base size	• Data type of variable specified for *DstDat* is a structure or STRING: 494 bytes • Other data types: 496 bytes

Item in above formula	Meaning
Size of variable name of *DstDat*	• The size of the variable name is calculated as the total bytes for the ASCII characters in all structure levels plus two times the number of levels. • If the number of bytes of ASCII characters in a level is an odd number, add 1. • If a level in the structure is an array, add four times the number of dimensions in the array. • Periods and commas in the structure and arrays are not included in the variable name size. Example 1: When the Variable Name of *DstDat* Is *aaa.bbbbb[1,2,3].cc* • The text string “aaa” in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • The text string “bbbbb[1,2,3]” in the second level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • Also *bbbbb[1,2,3]* is a three-dimensional array, so 3 times 4, or 12, is added to make 18 bytes. • The text string “cc” in the third level is 2 bytes. It is an even number, so 2 bytes is used in the calculation. • If we add the number of levels 3 times 2, or 6, to 4 bytes for the first level, 18 bytes for the second level, and 2 bytes for the third level, the size of the variable name come to 30 bytes. Example 2: When the Variable Name of *DstDat* Is val • The text string “val” in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 6 bytes. Example 3: When the Variable Name of *DstDat* Is *array[8]*. • The text string “array” in the first level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • It is a one-dimensional array. Therefore, 1 times 4, or 4, is added. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 12 bytes.

Item in above formula	Meaning

Path information size	• If there are no hops, the path information size is 0 bytes.*
	• If there are hops, the path information size is the route path size plus 12 bytes.
	• The route path size is the bytes size of the ASCII characters in the route path.
	• However, the following precautions apply.
	  • If the address portion starts with "#", calculate the network and address portions as a total of 2 bytes.
	  • If the address portion does not start with "#", calculate the network portion as 2 bytes.
	  • If the address portion does not start with "#" and the number of bytes in the ASCII characters for the address portion is an odd number, add 1 byte.
	  • Do not include the level separator, "\", between levels of the route path in the route path size.
	  • Do not include the first hop in the route path size.
	Example 1: When the Route Path Is 01\#11\02\192.168.250.2\01\#01
	  • The first hop in the route path size is not included, so ignore '01\#11' at the start of the path.
	  • The network type is '02', so use 2 bytes in the calculation.
	  • The address portion is '192.168.250.2', so use 13 bytes in the calculation. It is an odd number, so 1 is added to make 14 bytes.
	  • For the following '01\#01', the address portion starts with "#", so the network and address portions are calculated as a total of 2 bytes.
	  • If you add all of the above sizes, the size of the route path is 18 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 30 bytes.
	Example 2: When the Route Path Is 02\192.168.250.2\01\#00
	  • The first hop in the route path size is not included, so ignore '02\192.168.250.2' at the start of the path.
	  • For the following '01#01', the address portion starts with "#", so the network and address portions are calculated as a total of 2 bytes.
	  • Therefore, the size of the route path is 2 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 14 bytes.
	Example 3: When the Route Path Is 02\192.168.250.2
	  • If there are no hops, the path information size is 0 bytes.

* A hop is routing between the sending node and receiving node. For example, if the route path is 02\192.168.250.2\01\#00, the message is first routed to the node with an IP address of 192.168.250.2 to send the message to unit address 00. This involves one hop.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)

- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You can use this instruction only through a built-in EtherNet/IP port on an NJ-series CPU Unit or a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to write data. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot directly specify an address in local memory for CJ-series Units. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *SrcDat*.
- The characters that can be used in *DstDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and '_' (underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with "P_"

- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The text string in *RoutePath* does not end in a NULL character.
  - The value of *TimeOut* is outside of the valid range.
  - The value of *Size* is outside of the valid range.
  - The value of *SrcDat* is outside of the valid range.
- For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#05000000	The specified destination variable does not exist on the other Controller.
16#0C008010	The specified destination variable is being downloaded.
16#0C008011	
16#1F000102	The specified destination variable has a Constant attribute, so it cannot be written.
16#20008017	The specified destination variable is not an array and the number of elements to write is not 1.
16#20008018	The specified destination variable is an array and the number of elements to write exceeds the number of elements in the array.

Value	Error
16#20008028	• The specified destination variable is an enumeration and the write data is not the value of an enumerator. • The specified destination variable has a Range Specification attribute and the write data is out of range.
16#26000000	The specified destination variable name is only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPUCMMSend instruction (page 2-716).

CIPUCMMSend

The CIPUCMMSend instruction sends a UCMM CIP message to a specified device on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPUCMM Send	Send Explicit Message UCMM	FB	CIPUCMMSend_instance CIPUCMMSend Execute / Done RoutePath / Busy TimeOut / Error ServiceCode / ErrorID RqPath / ErrorIDEx ServiceDat / RespSize Size RespServiceDat	CIPUCMMSend_instance(Execute, RoutePath, TimeOut, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2.0 s)
ServiceCode	Service code		Service code	Depends on data type.	---	---
RqPath	Request path		Request path (class ID, instance ID, attribute ID)	---		---
ServiceDat	Command data		Data to send	Depends on data type.		*
Size	Number of elements to send		Number of elements to send			1
RespServiceDat	Response data	In-out	Response data	Depends on data type.	---	---
RespSize	Response size	Output	Response data size	Depends on data type.	Bytes	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				OK
TimeOut							OK													
Service Code		OK																		
ReqPath	Refer to *Function* for details on the structure _sREQUEST_PATH.																			
ServiceDat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
Size							OK													

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RespSer-viceDat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
RespSize							OK													

Function

The CIPUCMMSend instruction sends command data *ServiceDat* for the service specified with service code *ServiceCode* as a UCMM explicit message.
The destination is specified with route path *RoutePath*.
RqPath specifies the request path.

Size specifies the number of elements to send. If *ServiceDat* is an array, specify the number of elements to send with *Size*. If *ServiceDat* is not an array, always specify 1 for *Size*. If no service data is required, set *Size* to 0.

The response data received later is stored in *RespServiceDat*. The number of bytes of the response data is stored in *RespSize*.

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed.

The data type of *RqPath* is structure _sREQUEST_PATH. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
RqPath	Request path	Request path (class ID, instance ID, attribute ID)	_sREQUEST_PATH	---	---	---
ClassID	Class ID	Class ID	UINT	Depends on data type.	---	0
InstanceID	Instance ID	Instance ID	UINT			0
isAttributeID	Attribute usage	TRUE:Attribute ID used. FALSE:Attribute ID not used.	BOOL			FALSE
AttributeID	Attribute ID	Attribute ID	UINT			0

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*. The meaning and values of *ErrorIDEx* depend on the remote node. Refer to the manual for the remote node.

Sending and Receiving Arrays

If *ServiceDat* or *RespServiceDat* is an array, pass a subscripted array element to it as the parameter.

Maximum Read/Write Data Size

You can read a maximum of 492 bytes of data. The maximum size of the data that you can write depends on whether there is a request path attribute and the route path that is used, as given below.

Maximum write data size [bytes] = Base size − Attribute usage − Path information size

Item in above formula	Meaning
Base size	500 bytes
Attribute usage	Attribute ID used: 12 bytes Attribute ID not used: 8 bytes

Item in above formula	Meaning

Path information size	• If there are no hops, the path information size is 0 bytes.*
	• If there are hops, the path information size is the route path size plus 12 bytes.
	• The route path size is the bytes size of the ASCII characters in the route path.
	• However, the following precautions apply.
	  • If the address portion starts with “#”, calculate the network and address portions as a total of 2 bytes.
	  • If the address portion does not start with “#”, calculate the network portion as 2 bytes.
	  • If the address portion does not start with “#” and the number of bytes in the ASCII characters for the address portion is an odd number, add 1 byte.
	  • Do not include the level separator, “\”, between levels of the route path in the route path size.
	  • Do not include the first hop in the route path size.
	Example 1: When the Route Path Is 01\#11\02\192.168.250.2\01\#01
	  • The first hop in the route path size is not included, so ignore ‘01\#11’ at the start of the path.
	  • The network type is ‘02’, so use 2 bytes in the calculation.
	  • The address portion is ‘192.168.250.2’, so use 13 bytes in the calculation. It is an odd number, so 1 is added to make 14 bytes.
	  • For the following ‘01\#01’, the address portion starts with “#”, so the network and address portions are calculated as a total of 2 bytes.
	  • If you add all of the above sizes, the size of the route path is 18 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 30 bytes.
	Example 2: When the Route Path Is 02\192.168.250.2\01\#00
	  • The first hop in the route path size is not included, so ignore ‘02\192.168.250.2’ at the start of the path.
	  • For the following ‘01#01’, the address portion starts with “#”, so the network and address portions are calculated as a total of 2 bytes.
	  • Therefore, the size of the route path is 2 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 14 bytes.
	Example 3: When the Route Path Is 02\192.168.250.2
	  • If there are no hops, the path information size is 0 bytes.

* A hop is routing between the sending node and receiving node. For example, if the route path is 02\192.168.250.2\01\#00, the message is first routed to the node with an IP address of 192.168.250.2 to send the message to unit address 00. This involves one hop.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506)

- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You can use this instruction only through a built-in EtherNet/IP port on an NJ-series CPU Unit or a port on an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The text string in *RoutePath* does not end in a NULL character.
  - The value of *TimeOut* is outside of the valid range.
  - The value of *ServiceCode* is outside of the valid range.
  - The value of a member of *RqPath* is outside of the valid range.
  - The value of *Size* is outside of the valid range.

Sample Programming

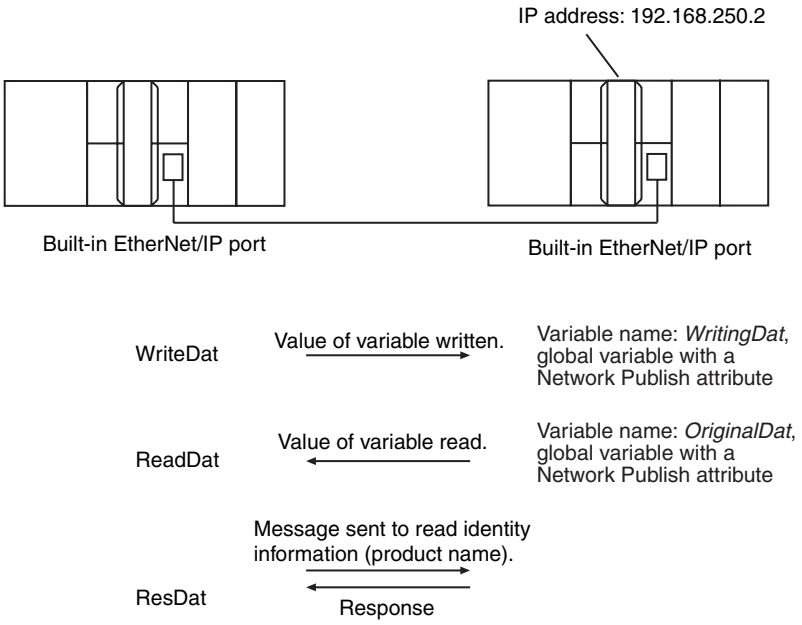
This sample uses CIP UCMM messages to write a variable, read a variable, and send a message. The Controllers are connected to an EtherNet/IP network. The IP address of the remote node is 192.168.250.2.
The following procedure is used.

1 The CIPUCMMWrite instruction is used to write the value of a variable at a remote node. The variable name at the remote node is *WritingDat* and the contents of the *WriteDat* is written to it. *WritingDat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.

2 The CIPUCMMRead instruction is used to read the value of a variable at a remote node. The value of the variable *OriginalDat* at the other node is read and the read value is stored in the *ReadDat* variable. *OriginalDat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.

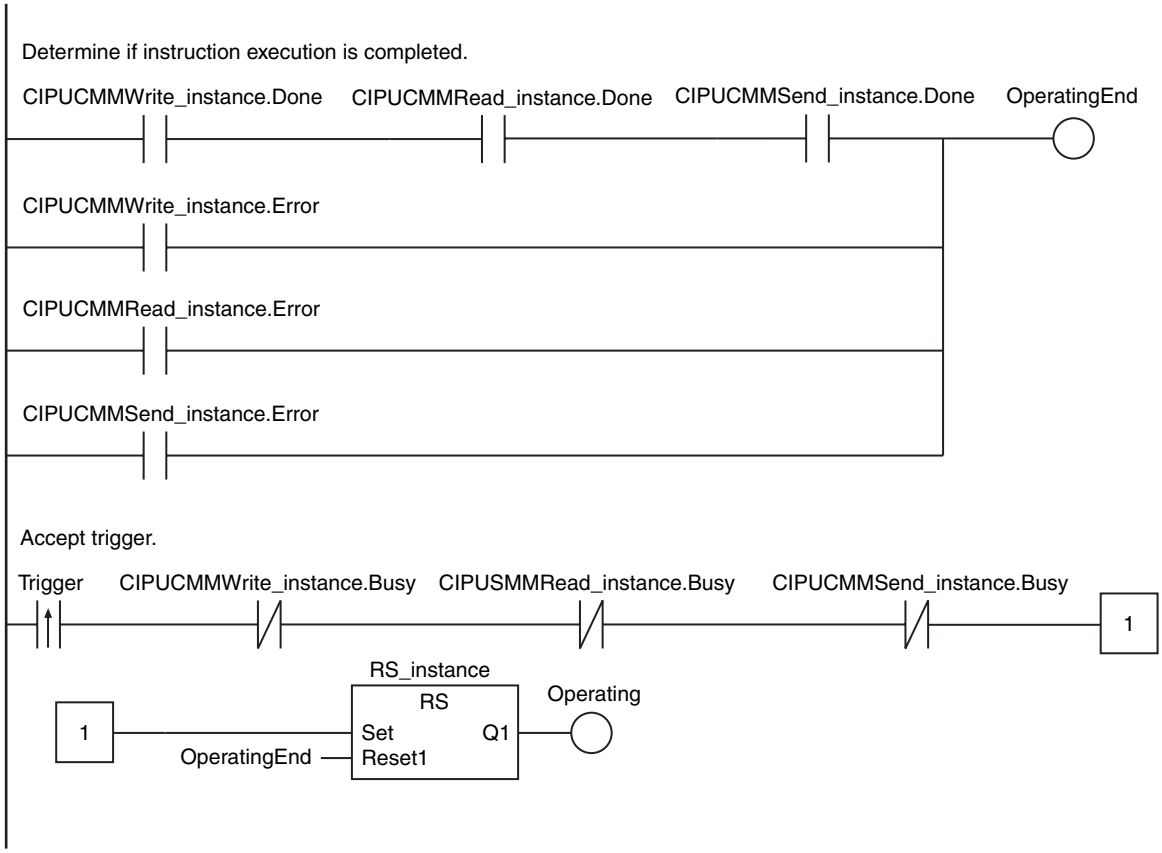
3 The CIPUCMMSend instruction is used to send an explicit message to a remote node. The contents of the message is to read identity information (product name). The class ID, instance ID, attribute ID, and service code are as follows: The response data is stored in the *ResDat* variable.

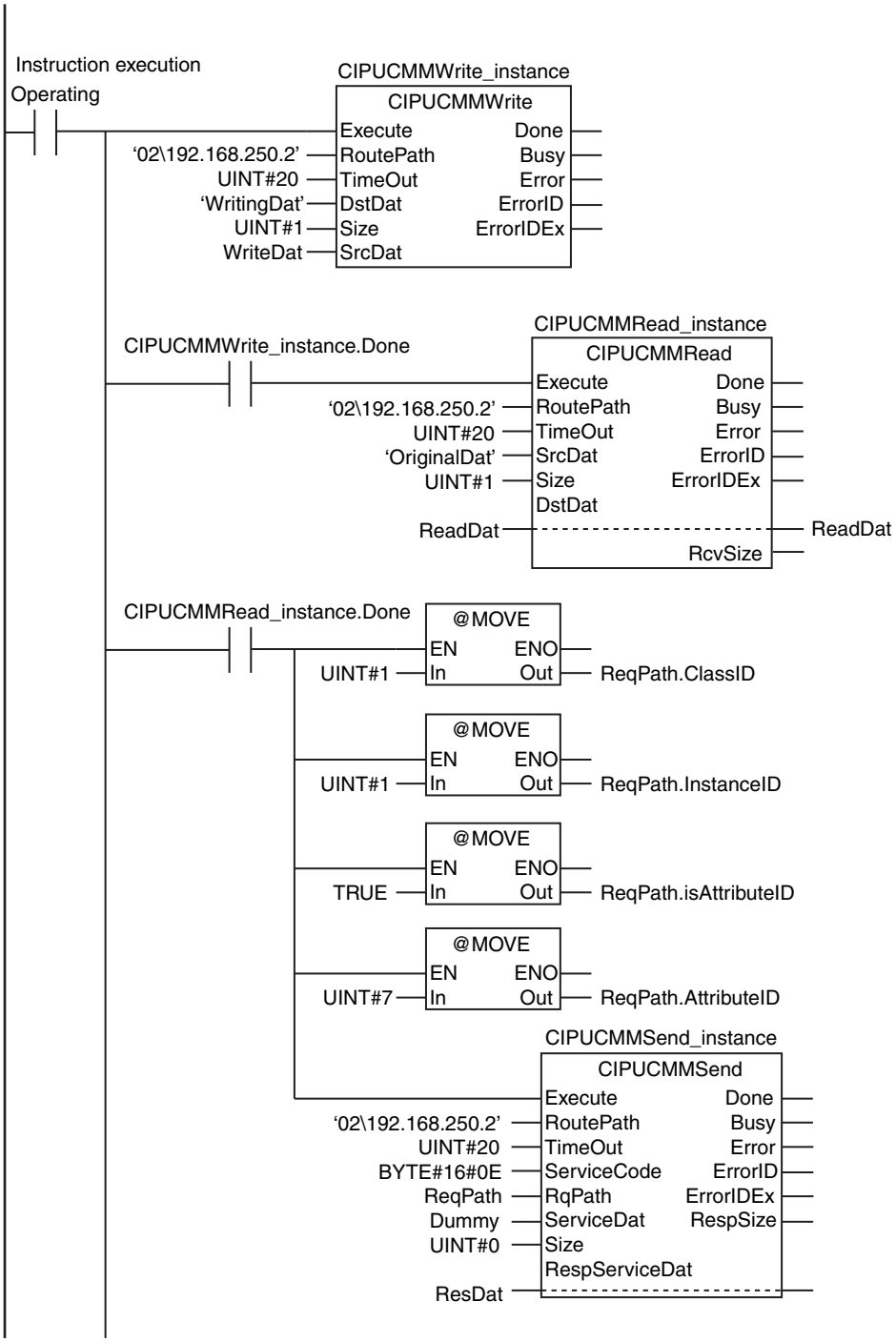
Item	Value
Class ID	1
Instance ID	1
Attribute ID	7
Service code	16#0E

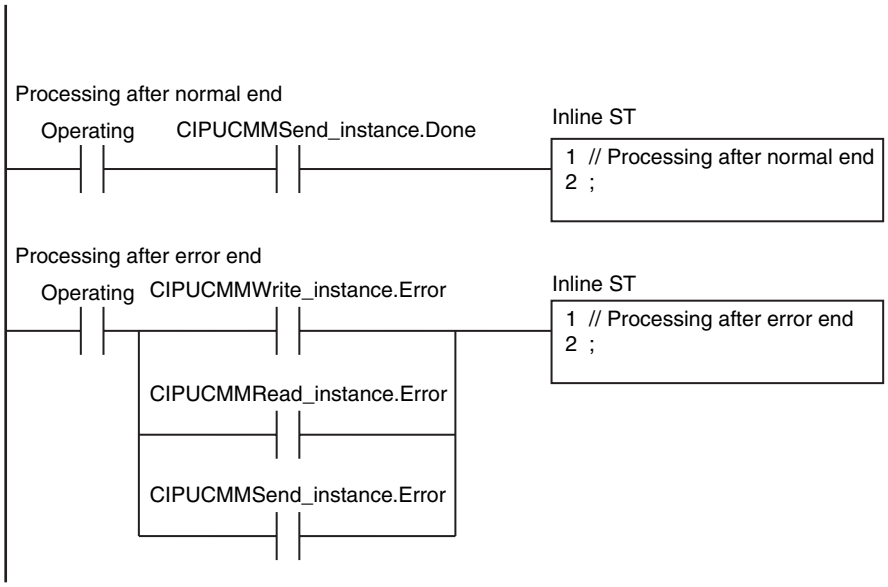

IP address: 192.168.250.2
Built-in EtherNet/IP port
Built-in EtherNet/IP port
WriteDat
Value of variable written.
Variable name: *WritingDat*, global variable with a Network Publish attribute
ReadDat
Value of variable read.
Variable name: *OriginalDat*, global variable with a Network Publish attribute
Message sent to read identity information (product name).
ResDat
Response

LD

Variable	Data type	Initial value	Comment
OperatingEnd	BOOL	False	Processing completed
Trigger	BOOL	False	Execution condition
Operating	BOOL	False	Processing
WriteDat	INT	1234	Write data
ReadDat	INT	0	Read data
ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=False, AttributeID:=0)	Request path
ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
Dummy	BYTE	16#0	Dummy
RS_instance	RS		
CIPUCMMWrite_instance	CIPUCMMWrite		
CIPUCMMRead_instance	CIPUCMMRead		
CIPUCMMSend_instance	CIPUCMMSend		




Instruction execution
Operating
CIPUCMMWrite_instance
CIPUCMMWrite
Execute
Done
'02\192.168.250.2'
RoutePath
Busy
UINT#20
TimeOut
Error
'WritingDat'
DstDat
ErrorID
UINT#1
Size
ErrorIDEx
WriteDat
SrcDat
CIPUCMMWrite_instance.Done
CIPUCMMRead_instance
CIPUCMMRead
Execute
Done
'02\192.168.250.2'
RoutePath
Busy
UINT#20
TimeOut
Error
'OriginalDat'
SrcDat
ErrorID
UINT#1
Size
ErrorIDEx
DstDat
ReadDat
ReadDat
RcvSize
CIPUCMMRead_instance.Done
@MOVE
EN
ENO
UINT#1
In
Out
ReqPath.ClassID
@MOVE
EN
ENO
UINT#1
In
Out
ReqPath.InstanceID
@MOVE
EN
ENO
TRUE
In
Out
ReqPath.isAttributeID
@MOVE
EN
ENO
UINT#7
In
Out
ReqPath.AttributeID
CIPUCMMSend_instance
CIPUCMMSend
Execute
Done
'02\192.168.250.2'
RoutePath
Busy
UINT#20
TimeOut
Error
BYTE#16#0E
ServiceCode
ErrorID
ReqPath
RqPath
ErrorIDEx
Dummy
ServiceDat
RespSize
UINT#0
Size
RespServiceDat
ResDat


Processing after normal end
Operating
CIPUCMMSend_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
CIPUCMMWrite_instance.Error
Inline ST
1 // Processing after error end
2 ;
CIPUCMMRead_instance.Error
CIPUCMMSend_instance.Error

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	DoUCMMTrigger	BOOL	False	Processing
	Stage	INT	0	Stage change
	WriteDat	INT	0	Write data
	ReadDat	INT	0	Read data
	ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=False, AttributeID:=0)	Request path
	ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
	Dummy	BYTE	16#0	Dummy
	CIPUCMMWrite_instance	CIPUCMMWrite		
	CIPUCMMRead_instance	CIPUCMMRead		
	CIPUCMMSend_instance	CIPUCMMSend		

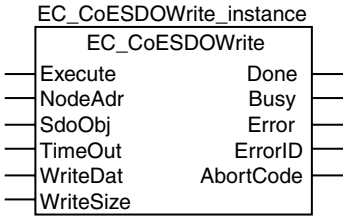
External Variables	Variable	Constant	Data type	Comment
	_EIP_EtnOnlineSta	☑	BOOL	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoUCMMTrigger=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoUCMMTrigger:=TRUE;
    Stage        :=INT#1;
    CIPUCMMWrite_instance(
        Execute          :=FALSE,          // Initialize instance.
        SrcDat           :=WriteDat);      // Dummy
    CIPUCMMRead_instance(                  // Initialize instance.
        Execute          :=FALSE,          // Dummy
        DstDat           :=ReadDat);       // Dummy
    CIPUCMMSend_instance(
        Execute          :=FALSE,          // Initialize instance.
        ServiceDat       := Dummy,         // Dummy
        RespServiceDat:=ResDat);           // Dummy
END_IF;

IF (DoUCMMTrigger=TRUE) THEN
    CASE Stage OF
    1 :                                    // Request writing value of variable.
        CIPUCMMWrite_instance(
            Execute    :=TRUE,
            RoutePath:='02\192.168.250.2',     // Route path
            TimeOut    :=UINT#20,              // Timeout time
            DstDat     :='WritingDat',         // Source variable name
            Size       :=UINT#1,               // Number of elements to write
            SrcDat     :=WriteDat);            // Write data

        IF (CIPUCMMWrite_instance.Done=TRUE) THEN
            Stage:=INT#2;                      // Normal end
        ELSIF (CIPUCMMWrite_instance.Error=TRUE) THEN
            Stage:=INT#10;                     // Error end
        END_IF;

    2 :                                    // Request reading value of variable.
        CIPUCMMRead_instance(
            Execute    :=TRUE,
            RoutePath:='02\192.168.250.2',     // Route path
            TimeOut    :=UINT#20,              // Timeout time
            SrcDat     :='OriginalDat',        // Source variable name
            Size       :=UINT#1,               // Number of elements to read
            DstDat     :=ReadDat);             // Read data

        IF (CIPUCMMRead_instance.Done=TRUE) THEN
            Stage:=INT#3;                      // Normal end
        ELSIF (CIPUCMMRead_instance.Error=TRUE) THEN
            Stage:=INT#40;                     // Error end
        END_IF;
```

```
    3 :                                          // Send message
        ReqPath.ClassID      :=UINT#01;
        ReqPath.InstanceID   :=UINT#01;
        ReqPath.isAttributeID:=TRUE;
        ReqPath.AttributeID  :=UINT#07;
        CIPUCMMSend_instance(
            Execute          :=TRUE,
            RoutePath        :='02\192.168.250.2',  // Route path
            TimeOut          :=UINT#20,             // Timeout time
            ServiceCode      :=BYTE#16#0E,          // Service code
            RqPath           :=ReqPath,             // Request path
            ServiceDat       := Dummy,              // Service data
            Size             :=UINT#0,              // Number of elements
            RespServiceDat:=ResDat);                // Response data

        IF (CIPUCMMSend_instance.Done=TRUE) THEN
            Stage:=INT#0;                           // Normal end
        ELSIF (CIPUCMMSend_instance.Error=TRUE) THEN
            Stage:=INT#30;                          // Error end
        END_IF;

    0:                                  // Processing after normal end
        DoUCMMTrigger:=FALSE;
        Trigger           :=FALSE;

    ELSE                                // Processing after error end
        DoUCMMTrigger:=FALSE;
        Trigger           :=FALSE;
    END_CASE;
END_IF;
```

EC_CoESDOWrite

The EC_CoESDOWrite instruction writes a value to a CoE* object of a specified slave on an EtherCAT network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_CoESDO Write	Write Ether-CAT CoE SDO	FB		EC_CoESDOWrite_instance(Execute, NodeAdr, SdoObj, TimeOut, WriteDat, WriteSize, Done, Busy, Error, ErrorID, AbortCode);

* CoE stands for CAN Application Protocol over EtherCAT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to access	1 to 192	---	---
SdoObj	SDO parameter		SDO parameter	---		
TimeOut	Timeout time		0: 2.0s 1 to 65535: 0.1 to 6553.5 s	Depends on data type.	0.1 s	20 (2.0 s)
WriteDat	Write data		Write data		---	---
WriteSize	Write data size		Write data size	1 to 2048	Bytes	---
AbortCode	Abort code	Output	Response code for SDO access specified by CoE 0: Normal end	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													
SdoObj	Refer to *Function* for details on the structure _sSDO_ACCESS.																			
TimeOut							OK													
WriteDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, structure member, or union member can also be specified.																			
WriteSize							OK													
AbortCode				OK																

Function

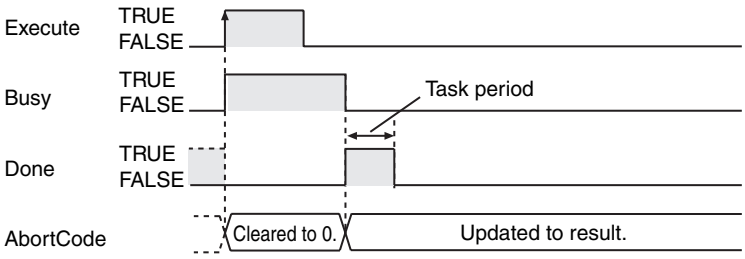
The EC_CoESDOWrite instruction writes data to the CoE object of the node specified with slave node address *NodeAdr*. The content of *WriteDat* is written to the object. The number of bytes of data to write is specified with *WriteSize*. The SDO parameter is specified with *SdoObj*.

The data type of *SdoObj* is structure _sSDO_ACCESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
SdoObj	SDO parameter	SDO parameter	_sSDO_ ACCESS	---	---	---
Index	Index	Index number in the object dictionary defined in CoE	UINT	1 to 65535	---	---
Subindex	Subindex	Subindex number in the object dictionary defined in CoE	USINT	Depends on data type.		
IsCompleteAccess	Complete access	Specification of complete access of SDO TRUE: Access data for all subindexes FALSE: Access data for the specified subindex	BOOL			

After the write is completed, the instruction waits for the response for the time specified with timeout time *TimeOut*. The response is stored in *AbortCode*. *AbortCode* is 0 for a normal response. A value is stored in *AbortCode* only when the value of *ErrorID* is 16#1804 (SDO abort response).
The meaning and values of *AbortCode* depend on the slave. Refer to the manual for the slave.

The following figure shows a timing chart. A value is stored in *AbortCode* when *Busy* changes to FALSE after the completion of instruction processing.



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates when communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.

Additional Information

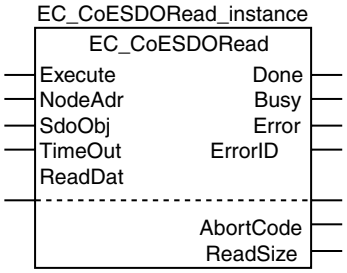
- Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.
- Refer to *A-4 SDO Abort Codes* on page A-47 for the SDO abort codes.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The EtherCAT master is not in a state that allows message communications.
  - The slave specified with *NodeAdr* does not exist.
  - The slave specified with *NodeAdr* is not in a state that allows communications.
  - The slave returns an error response.

EC_CoESDORead

The EC_CoESDORead instruction reads a value from a CoE* object of a specified slave on an EtherCAT network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_CoESDORead	Read EtherCAT CoE SDO	FB		EC_CoESDORead_instance(Execute, NodeAdr, SdoObj, TimeOut, ReadDat, Done, Busy, Error, ErrorID, AbortCode, ReadSize);

* CoE stands for CAN Application Protocol over EtherCAT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to access	1 to 192	---	---
SdoObj	SDO parameter		SDO parameter	---		
TimeOut	Timeout time		0: 2.0s 1 to 65535: 0.1 to 6553.5 s	Depends on data type.	0.1 s	0 (2.0 s)
AbortCode	Abort code	Output	Response code for SDO access specified by CoE 0: Normal end	Depends on data type.	---	---
ReadSize	Read data size		Size of data stored in *ReadDat* after the data is read		Bytes	
ReadDat	Read data	In-out	Read data buffer	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													
SdoObj	Refer to *Function* for details on the structure _sSDO_ACCESS.																			
TimeOut							OK													
AbortCode				OK																
ReadSize							OK													
ReadDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, structure member, or union member can also be specified.																			

Function

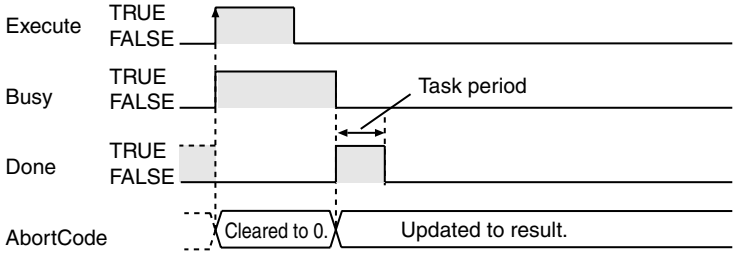
The EC_CoESDORead instruction reads data from the CoE object of the node specified with slave node address *NodeAdr*. The read data is stored in *ReadDat*. Then size of data that was stored is stored in *ReadSize*. The value of *ReadSize* is valid only when the data was stored successfully.
The SDO parameter is specified with *SdoObj*.

The data type of *SdoObj* is structure _sSDO_ACCESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
SdoObj	SDO parameter	SDO parameter	_sSDO_ ACCESS	---	---	---
Index	Index	Index number in the object dictionary defined in CoE	UINT	1 to 65535		
Subindex	Subindex	Subindex number in the object dictionary defined in CoE	USINT			
IsCompleteAccess	Complete access	Specification of complete access of SDO TRUE:Access data for all subindexes FALSE:Access data for the specified subindex	BOOL	Depends on data type.	---	---

After the read is completed, the instruction waits for the response for the time specified with timeout time *TimeOut*. The response is stored in *AbortCode*. *AbortCode* is 0 for a normal response. A value is stored in *AbortCode* only when the value of *ErrorID* is 16#1804 (SDO abort response).
The meaning and values of *AbortCode* depend on the slave. Refer to the manual for the slave.

The following figure shows a timing chart. A value is stored in *AbortCode* when *Busy* changes to FALSE after the completion of instruction processing.



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlaveTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates when communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.

Additional Information

- Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.
- Refer to *A-4 SDO Abort Codes* on page A-47 for the SDO abort codes.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The EtherCAT master is not in a state that allows message communications.
  - The slave specified with *NodeAdr* does not exist.
  - The slave specified with *NodeAdr* is not in a state that allows communications.
  - The slave returns an error response.
  - The read data size is larger than the size of *ReadDat*.

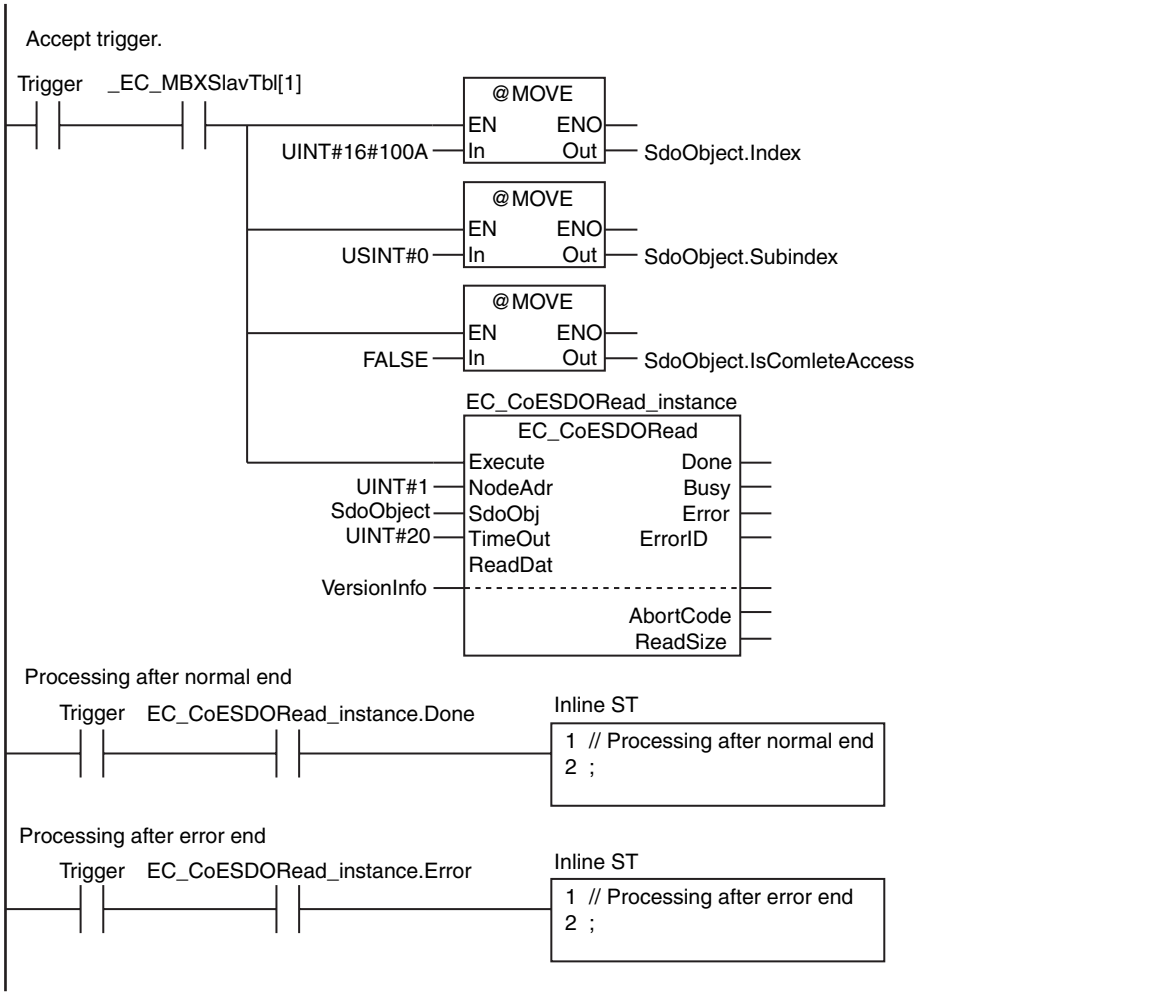
Sample Programming

This sample uses an EtherCAT SDO message to read the software version of an OMRON V1.02 R88D-KN01L-ECT Servo Drive. The node address of the slave is 1.
The object index for the software version is 16#100A. The subindex is 0. The read value is stored in STRING variable *VersionInfo*.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=False)	SDO parameter
	VersionInfo	STRING[256]	''	Read data
	EC_CoESDORead_instance	EC_CoESDORead		

External Variables	Variable	Data type	Constant	Comment
	_EC_MBXSlavTbl	ARRAY[1..192] OF BOOL	☑	Message Communications Enabled Slave Table



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=False)	SDO parameter
	DoSdoRead	BOOL	False	Processing
	VersionInfo	STRING[256]	''	Read data
	NormalEnd	UINT	0	Normal end
	ErrorEnd	UINT	0	Error end
	EC_CoESDORead_instance	EC_CoESDORead		

External Variables	Variable	Data type	Constant	Comment
	_EC_MBXSlavTbl	ARRAY[1..192] OF BOOL	☑	Message Communications Enabled Slave Table

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoSdoRead=FALSE) AND (_EC_MBXSlavTbl[1]=TRUE) ) THEN
    DoSdoRead                 :=TRUE;
    SdoObject.Index           :=UINT#16#100A;
    SdoObject.Subindex        :=USINT#0;
    SdoObject.IsCompleteAccess:=FALSE;
    EC_CoESDORead_instance(
        Execute:=FALSE,       // Initialize instance.
        ReadDat:=VersionInfo); // Dummy
END_IF;

// Execute EC_CoESDORead instruction.
IF (DoSdoRead=TRUE) THEN
    EC_CoESDORead_instance(
        Execute  :=TRUE,
        NodeAdr :=UINT#1,     // Node address 1
        SdoObj   :=SdoObject, // SDO parameter
        TimeOut :=UINT#20,    // Timeout time: 2.0 s
        ReadDat:=VersionInfo); // Read data

    IF (EC_CoESDORead_instance.Done=TRUE) THEN
        // Processing after normal end
        NormalEnd:=NormalEnd+UINT#1;
    ELSIF (EC_CoESDORead_instance.Error=TRUE) THEN
        // Processing after error end
        ErrorEnd    :=ErrorEnd+UINT#1;
    END_IF;

    DoSdoRead:=FALSE;
END_IF;
```

EC_StartMon

The EC_StartMon instruction starts execution of packet monitoring for EtherCAT communications.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_StartMon	Start EtherCAT Packet Monitor	FB	EC_StartMon_instance EC_StartMon Execute Done Busy Error ErrorID	EC_SatrtMon_instance(Execute, Done, Busy, Error, ErrorID);

Variables

Only common variables are used.

Function

The EC_StartMon instruction starts execution of packet monitoring for EtherCAT communications. The packet monitor function collects a specified number of the most recent EtherCAT communications packets. When the specified number of packets is exceeded, old packets are discarded in order. After the EC_StartMon instruction is executed, packet monitoring continues until the EC_StopMon instruction is executed.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktMonStop	Packet Monitoring Stopped	BOOL	This variable shows if packet monitoring is stopped. TRUE: Stopped. FALSE: Not stopped.
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

- You cannot save collected packet data in an internal file of the main memory of the CPU Unit during ECATStartMonitor execution.
- Do the following to save packet data in an internal file in the main memory of the CPU Unit: First, execute the EC_StopMon instruction to stop packet monitoring. Then execute the EC_SaveMon instruction to save the packets.
- Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.
- An error occurs in the following case. *Error* will change to TRUE.
  - A packet data save operation to an internal file in the main memory of the CPU Unit is in progress.

Sample Programming

This sample transfers EtherCAT communications packets to an SD Memory Card when an EtherCAT slave error occurs. The file name is 'PacketFile.' The processing procedure is as follows:

1 The system-defined variable *_EC_ErrSta* (EtherCAT Error) is monitored and processing is started if an error occurs.

2 The EC_StopMon instruction is used to stop execution of packet monitoring for EtherCAT communications.

3 The EC_SaveMon instruction is used to save EtherCAT communications packet data to an internal file in the main memory of the CPU Unit.

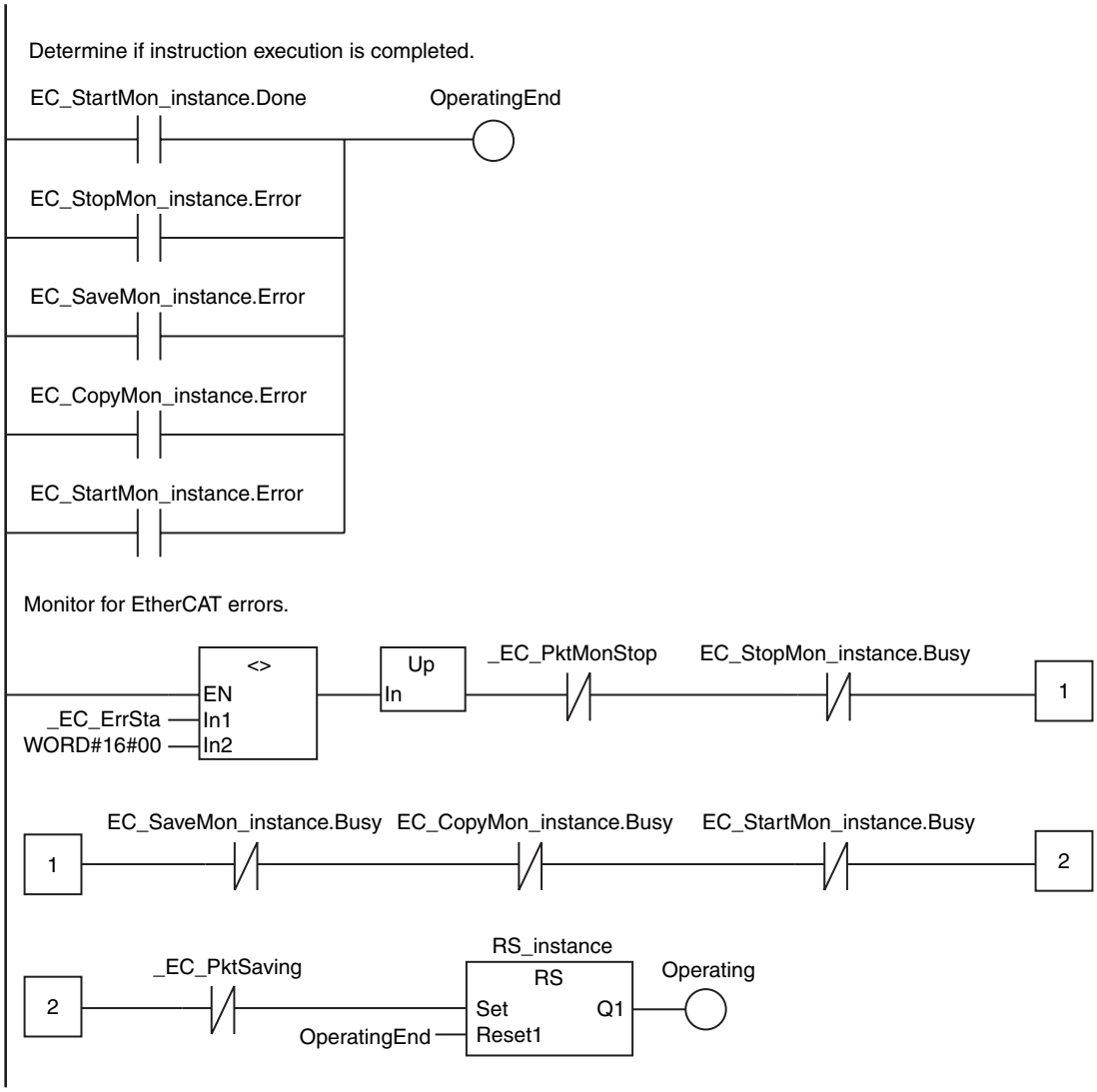
4 The EC_CopyMon instruction is used to copy that file to the SD Memory Card.

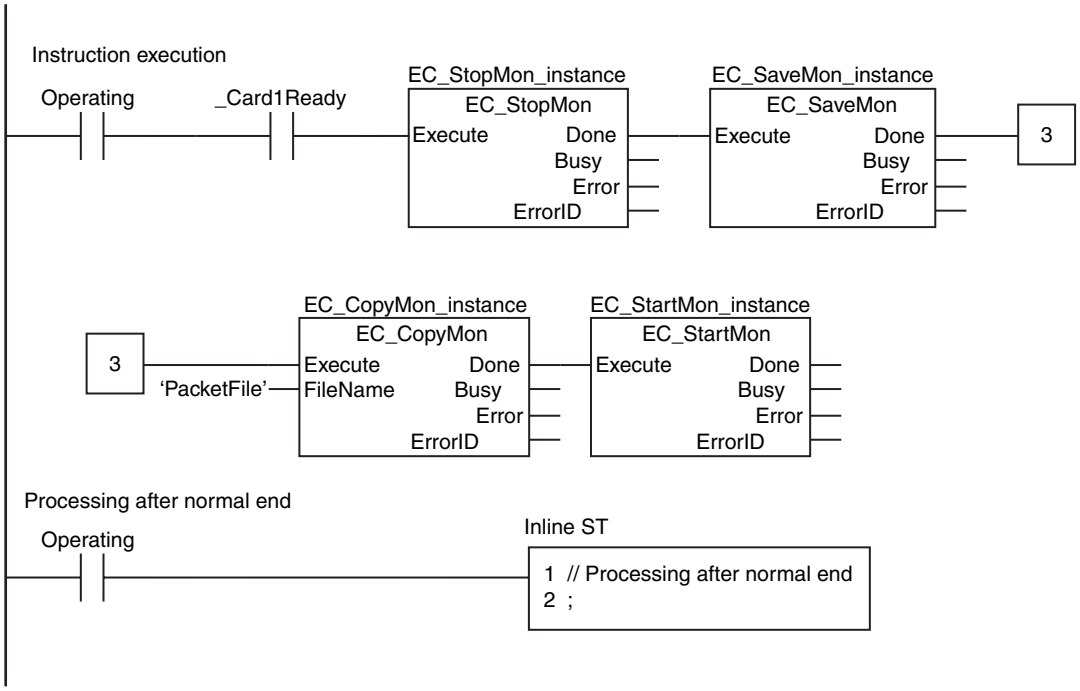
5 The EC_StartMon instruction is used to restart execution of packet monitoring for EtherCAT communications.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed
	Operating	BOOL	False	Execution condition
	RS_instance	RS		
	EC_StopMon_instance	EC_StopMon		
	EC_SaveMon_instance	EC_SaveMon		
	EC_CopyMon_instance	EC_CopyMon		
	EC_StartMon_instance	EC_StartMon		

External Variables	Variable	Data type	Constant	Comment
	_EC_ErrSta	WORD	☑	Built-in EtherCAT Error
	_EC_PktMonStop	BOOL	☑	Packet Monitoring Stopped
	_EC_PktSaving	BOOL	☑	Saving Packet Data File
	_Card1Ready	BOOL	☑	SD Memory Card Ready Flag




Instruction execution
Operating
_Card1Ready
EC_StopMon_instance
EC_StopMon
Execute
Done
Busy
Error
ErrorID
EC_SaveMon_instance
EC_SaveMon
Execute
Done
Busy
Error
ErrorID
3
3
'PacketFile'
EC_CopyMon_instance
EC_CopyMon
Execute
FileName
Done
Busy
Error
ErrorID
EC_StartMon_instance
EC_StartMon
Execute
Done
Busy
Error
ErrorID
Processing after normal end
Operating
Inline ST
1 // Processing after normal end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	EC_Err	BOOL	False	Controller error in the EtherCAT Master Function Module.
	EC_Err_Trigger	BOOL	False	Detect when *EC_Err* changes to TRUE.
	DoEC_PktSave	BOOL	False	Processing
	Stage	INT	0	Stage change
	R_TRIG_instance	R_TRIG		
	EC_StopMon_instance	EC_StopMon		
	EC_SaveMon_instance	EC_SaveMon		
	EC_CopyMon_instance	EC_CopyMon		
	EC_StartMon_instance	EC_StartMon		

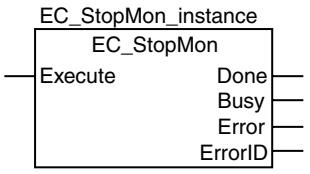
External Variables	Variable	Data type	Constant	Comment
	_EC_ErrSta	WORD	☑	Built-in EtherCAT Error
	_EC_PktMonStop	BOOL	☑	Packet Monitoring Stopped
	_EC_PktSaving	BOOL	☑	Saving Packet Data File
	_Card1Ready	BOOL	☑	SD Memory Card Ready Flag

```
// Start sequence when _EC_ErrSta changes to TRUE.
EC_Err:=(_EC_ErrSta <> WORD#16#00);
R_TRIG_instance(Clk:=EC_Err, Q=>EC_Err_Trigger);

IF ( (EC_Err_Trigger=TRUE) AND (DoEC_PktSave=FALSE) AND (_EC_PktMonStop=FALSE)
     AND (_EC_PktSaving=FALSE) AND (_Card1Ready=TRUE) ) THEN
    DoEC_PktSave:=TRUE;
    Stage             :=INT#1;
    EC_StopMon_instance(Execute:=FALSE);    // Initialize instance.
    EC_SaveMon_instance(Execute:=FALSE);
    EC_CopyMon_instance(Execute:=FALSE);
    EC_StartMon_instance(Execute:=FALSE);
END_IF;

 // Instruction execution
IF (DoEC_PktSave=TRUE) THEN
    CASE Stage OF
    1 :                     // Stop EtherCAT packet monitor.
        EC_StopMon_instance(
            Execute    :=TRUE);

        IF (EC_StopMon_instance.Done=TRUE) THEN
            Stage:=INT#2;               // Normal end
        ELSIF (EC_StopMon_instance.Error=TRUE) THEN
            Stage:=INT#10;              // Error end
        END_IF;

    2 :                     // Save EtherCAT packet data in an internal file.
        EC_SaveMon_instance(
            Execute :=TRUE);

        IF (EC_SaveMon_instance.Done=TRUE) THEN
            Stage:=INT#3;               // Normal end
        ELSIF (EC_SaveMon_instance.Error=TRUE) THEN
            Stage:=INT#20;              // Error end
        END_IF;

    3 :                     // Copy EtherCAT packet data file to the SD Memory Card.
        EC_CopyMon_instance(
            Execute  :=TRUE,
            FileName:='PacketFile');

        IF (EC_CopyMon_instance.Done=TRUE) THEN
            Stage:=INT#4;               // Normal end
        ELSIF (EC_CopyMon_instance.Error=TRUE) THEN
            Stage:=INT#30;              // Error end
        END_IF;
```

```
    4 :       // Restart EtherCAT packet monitor.
        EC_StartMon_instance(
            Execute  :=TRUE);

        IF (EC_StartMon_instance.Done=TRUE) THEN
            Stage:=INT#0;    // Normal end
        ELSIF (EC_StartMon_instance.Error=TRUE) THEN
            Stage:=INT#40;  // Error end
        END_IF;

    0 :                  // Processing after normal end
        DoEC_PktSave:=FALSE;

    ELSE                 // Processing after error end
        DoEC_PktSave:=FALSE;
    END_CASE;
END_IF;
```

EC_StopMon

The EC_StopMon instruction stops execution of packet monitoring for EtherCAT communications.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_StopMon	Stop EtherCAT Packet Monitor	FB		EC_StopMon_instance(Execute, Done, Busy, Error, ErrorID);

Variables

Only common variables are used.

Function

The EC_StopMon instruction stops execution of packet monitoring for EtherCAT communications. The packet monitor function collects a specified number of the most recent EtherCAT communications packets.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktMonStop	Packet Monitoring Stopped	BOOL	This variable shows if packet monitoring is stopped. TRUE: Stopped. FALSE: Not stopped.
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

- Do the following to save collected packet data in an internal file in the main memory of the CPU Unit: First, stop packet monitoring. Then execute the EC_SaveMon instruction to save the packets.
- Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.

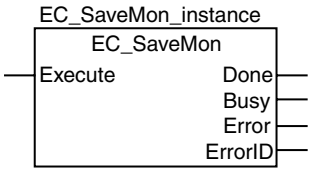
- An error occurs in the following case. *Error* will change to TRUE.
  - Packet monitoring is already stopped.

Sample Programming

Refer to the sample programming that is provided for the EC_StartMon instruction (page 2-734).

EC_SaveMon

The EC_SaveMon instruction saves EtherCAT communications packet data to an internal file in the main memory of the CPU Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_SaveMon	Save Ether-CAT Packets	FB		EC_SaveMon_instance(Execute, Done, Busy, Error, ErrorID);

Variables

Only common variables are used.

Function

The EC_SaveMon instruction saves EtherCAT communications packet data that was collected by the packet monitoring function to an internal file in the main memory of the CPU Unit. The packet monitor function collects a specified number of the most recent EtherCAT communications packets.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktMonStop	Packet Monitoring Stopped	BOOL	This variable shows if packet monitoring is stopped. TRUE: Stopped. FALSE: Not stopped.
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

- You cannot execute packet monitoring while this instruction is in execution.
- Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.

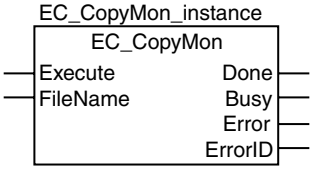
- You cannot execute this instruction while packet monitoring is in progress. Execute the EC_StopMon instruction in advance to stop packet monitoring.
- An error occurs in the following case. *Error* will change to TRUE.
  - Packet monitoring is in progress.

Sample Programming

Refer to the sample programming that is provided for the EC_StartMon instruction (page 2-734).

EC_CopyMon

The EC_CopyMon instruction transfers packet data in an internal file in the main memory of the CPU Unit to a SD Memory Card.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_CopyMon	Transfer Ether-CAT Packets	FB		EC_CopyMon_instance(Execute, File-Name, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	File name on the SD Memory Card	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK

Function

The EC_CopyMon instruction transfers packet data in an internal file in the main memory of the CPU Unit to a SD Memory Card. *FileName* specifies the file name on the SD Memory Card.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.

Precautions for Correct Use

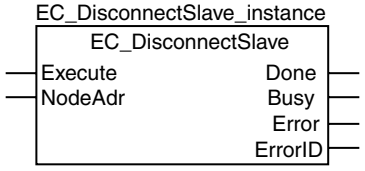
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.
- You cannot execute this instruction while a packet save operation is in progress.
- To use this instruction, execute the EC_SaveMon instruction in advance to save the packet data in an internal file in the main memory of the CPU Unit.
- An error occurs in the following case. *Error* will change to TRUE.
  - A packet data file save operation is in progress.

Sample Programming

Refer to the sample programming that is provided for the EC_StartMon instruction (page 2-734).

EC_DisconnectSlave

The EC_DisconnectSlave instruction disconnects the specified slave from the network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_Disconnect Slave	Disconnect EtherCAT Slave	FB		EC_DisconnectSlave_instance(Execute, NodeAdr, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to disconnect	1 to 192	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													

Function

The EC_DisconnectSlave instruction disconnects the slave specified with slave node address *NodeAdr* from the EtherCAT network.
Here, disconnection from the network means that the slave is placed in a state in which it does not operate even though it still exists on the network.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_EntrySlavTbl[i] "i" is the node address.	Network Connected Slave Table	BOOL[]	This variable shows if slaves are part of (i.e., exist on) the network. TRUE: Part of the network. FALSE: Not part of the network.
_EC_DisconnSlavTbl[i] "i" is the node address.	Disconnected Slave Table	BOOL[]	This variable shows the slaves for which there are currently disconnect commands in effect. TRUE: Disconnect command is in effect. FALSE: Disconnect command is not in effect.
_EC_DisableSlavTbl[i] "i" is the node address.	Disabled Slave Table	BOOL[]	This variable shows if slaves are disabled on the network. TRUE: Disabled. FALSE: Not disabled.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.
- If there are slaves with daisy-chain connections (i.e., connected to the output port) after the disconnected slave, they are disconnected from the EtherCAT network also.
- An error occurs in the following case. *Error* will change to TRUE.
  - The slave specified with *NodeAdr* is not part of the EtherCAT network. That is, the value of *_EC_EntrySlavTbl[i]* (Network Connected Slave Table) is FALSE.
  - The slave specified with *NodeAdr* is disabled.

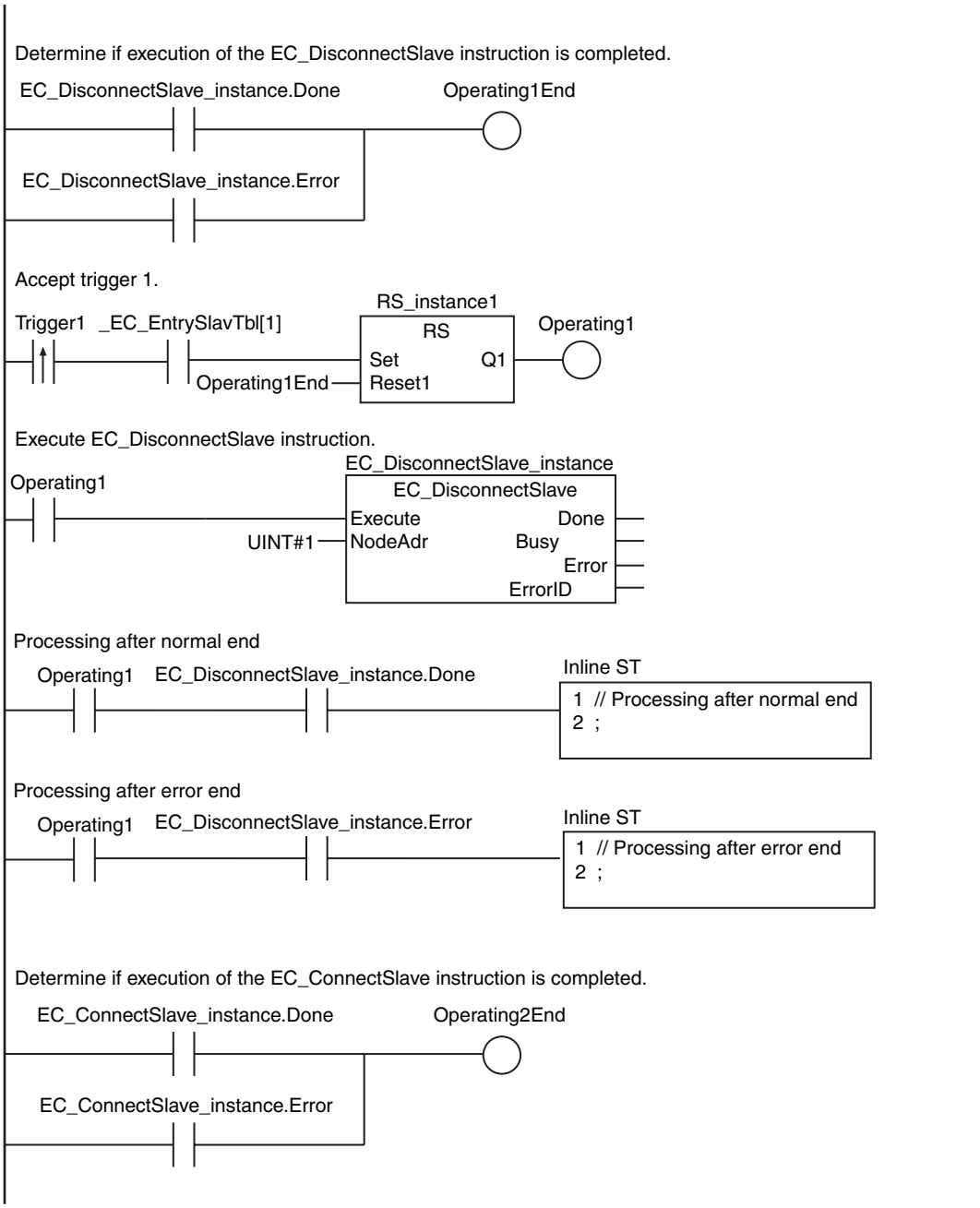
Sample Programming

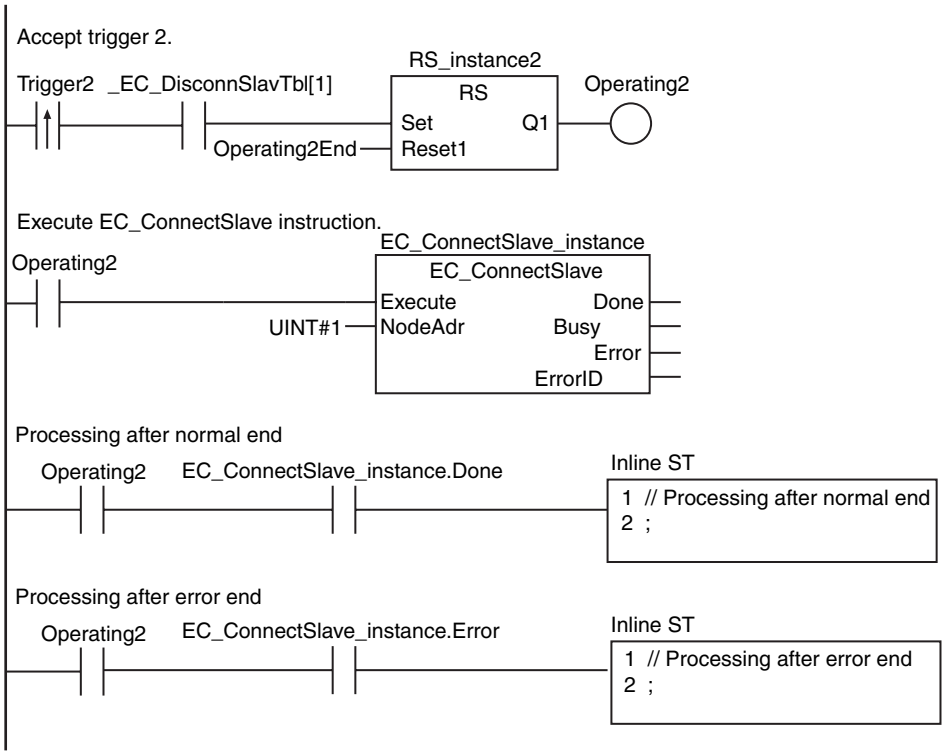
This sample disconnects slave 1 from the EtherCAT network and then connects it again. When Trigger 1 changes to TRUE, the EC_DisconnectSlave instruction is executed to disconnect slave 1. When Trigger 2 changes to TRUE, the EC_ConnectSlave instruction is executed to connect slave 1 again.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	Operating1End	BOOL	False	Processing 1 completed.
	Trigger1	BOOL	False	Execution condition 1
	Operating1	BOOL	False	Processing 1
	RS_instance1	RS		
	EC_DisconnectSlave_instance	EC_DisconnectSlave		
	Operating2End	BOOL	False	Processing 1 completed.
	Trigger2	BOOL	False	Execution condition 2
	Operating2	BOOL	False	Processing 2
	RS_instance2	RS		
	EC_ConnectSlave_instance	EC_ConnectSlave		

External Variables	Variable	Data type	Constant	Comment
	_EC_EntrySlavTbl	ARRAY[1..192] OF BOOL	☑	Network Connected Slave Table
	_EC_DisconnSlavTbl	ARRAY[1..192] OF BOOL	☑	Disconnected Slave Table




Accept trigger 2.
RS_instance2
RS
Trigger2
_EC_DisconnSlavTbl[1]
Set
Q1
Operating2
Operating2End
Reset1
Execute EC_ConnectSlave instruction.
EC_ConnectSlave_instance
EC_ConnectSlave
Operating2
Execute
Done
UINT#1
NodeAdr
Busy
Error
ErrorID
Processing after normal end
Operating2
EC_ConnectSlave_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating2
EC_ConnectSlave_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger1	BOOL	False	Execution condition 1
	LastTrigger1	BOOL	False	Value of *Trigger1* from previous task period
	Operating1Start	BOOL	False	Processing 1 started.
	Operating1	BOOL	False	Processing 1
	EC_DisconnectSlave_instance	EC_DisconnectSlave		
	Trigger2	BOOL	False	Execution condition 2
	LastTrigger2	BOOL	False	Value of *Trigger2* from previous task period
	Operating2Start	BOOL	False	Processing 2 started.
	Operating2	BOOL	False	Processing 2
	EC_ConnectSlave_instance	EC_ConnectSlave		

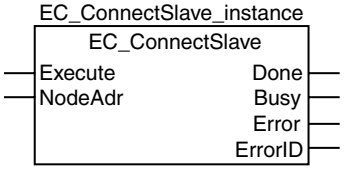
External Variables	Variable	Data type	Constant	Comment
	_EC_EntrySlavTbl	ARRAY[1..192] OF BOOL	☑	Network Connected Slave Table
	_EC_DisconnSlavTbl	ARRAY[1..192] OF BOOL	☑	Disconnected Slave Table

```
// Detect when Trigger1 changes to TRUE.
IF ( (Trigger1=TRUE) AND (LastTrigger1=FALSE) AND (_EC_EntrySlavTbl[1]=TRUE) ) THEN
    Operating1Start:=TRUE;
    Operating1      :=TRUE;
END_IF;
LastTrigger1:=Trigger1;

// Initialize EC_DisconnectSlave instruction.
IF (Operating1Start=TRUE) THEN
    EC_DisconnectSlave_instance(Execute:=FALSE);
    Operating1Start:=FALSE;
END_IF;

 // Execute EC_DisconnectSlave instruction.
IF (Operating1=TRUE) THEN
    EC_DisconnectSlave_instance(
        Execute :=TRUE,
        NodeAdr:=UINT#1);

    IF (EC_DisconnectSlave_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating1:=FALSE;
    END_IF;

    IF (EC_DisconnectSlave_instance.Error=TRUE) THEN
        // Processing after error end
        Operating1:=FALSE;
    END_IF;
END_IF;

// Detect when Trigger2 changes to TRUE.
IF ( (Trigger2=TRUE) AND (LastTrigger2=FALSE) AND (_EC_DisconnSlavTbl[1]=TRUE) ) THEN
    Operating2Start:=TRUE;
    Operating2      :=TRUE;
END_IF;
LastTrigger2:=Trigger2;

// Initialize EC_ConnectSlave instruction.
IF (Operating2Start=TRUE) THEN
    EC_ConnectSlave_instance(Execute:=FALSE);
    Operating2Start:=FALSE;
END_IF;
```

```
// Execute EC_ConnectSlave instruction.
IF (Operating2=TRUE) THEN
    EC_ConnectSlave_instance(
        Execute :=TRUE,
        NodeAdr:=UINT#1);

    IF (EC_ConnectSlave_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating2:=FALSE;
    END_IF;

    IF (EC_ConnectSlave_instance.Error=TRUE) THEN
        // Processing after error end
        Operating2:=FALSE;
    END_IF;
END_IF;
```

EC_ConnectSlave

The EC_ConnectSlave instruction connects the specified slave to the EtherCAT network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_ConnectSlave	Connect EtherCAT Slave	FB		EC_ConnectSlave_instance(Execute, NodeAdr, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to connect	1 to 192	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													

Function

The EC_ConnectSlave instruction connects the slave specified with slave node address *NodeAdr* to the EtherCAT network.
Here, connection to the network means that the slave exists on the network and it is placed in a state in which it operates.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_EntrySlavTbl[i] "i" is the node address.	Network Connected Slave Table	BOOL[]	This variable shows if slaves are part of (i.e., exist on) the network. TRUE: Part of the network. FALSE: Not part of the network.
_EC_DisconnSlavTbl[i] "i" is the node address.	Disconnected Slave Table	BOOL[]	This variable shows the slaves for which there are currently disconnect commands in effect. TRUE: Disconnect command is in effect. FALSE: Disconnect command is not in effect.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ-series EtherCAT ports.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The slave specified with *NodeAdr* is not part of the EtherCAT network. That is, the value of *_EC_EntrySlavTbl[i]* (Network Connected Slave Table) is FALSE.
  - The slave specified with *NodeAdr* is not disconnected from the network.

Sample Programming

Refer to the sample programming that is provided for the EC_DisconnectSlave instruction (page 2-746).

SktUDPCreate

The SktUDPCreate instruction creates a UDP socket request to open a servo port for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktUDP Create	Create UDP Socket	FB	None	SktUDPCreate_instance(Execute, SrcUdpPort, Done, Busy, Error, ErrorID, Socket);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcUdpPort	Local UDP port number	Input	Local UDP port number	1 to 65535	---	1
Socket	Socket	Output	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcUdpPort							OK													
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktUDPCreate instruction opens the port specified with the local UDP port number *ScrUdpPort*. To do this, it executes the Socket() and Bind() socket functions. Information on the socket that is opened is stored in *Socket*. The UDP port is open when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

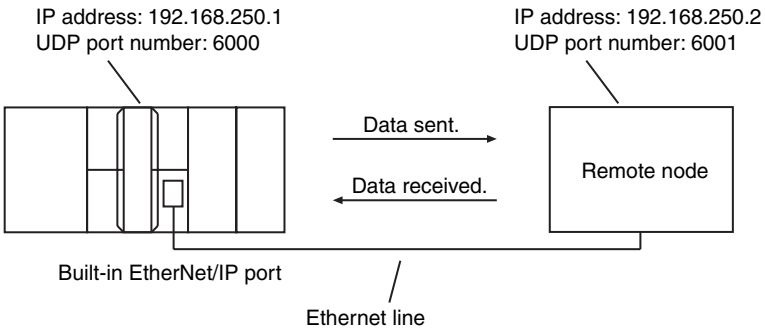
Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- Use the SktClose instruction to close handles that are created with this instruction.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, and SktClearBuf.
- You can open a maximum of 16 sockets combined for UDP and TCP sockets.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *SrcUdpPort* is outside of the valid range.
  - The port that is specified with *SrcUdpPort* is already open or close processing is in progress for it.

- The port that is specified with *ScrUdpPort* is already in use.

Sample Programming

In this sample, the UDP socket service is used for data communications between the NJ-series Controller and a remote node.



The processing procedure is as follows:

1. The SktUDPCreate instruction is used to request creating a UDP socket.
2. The SktUDPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.
3. The SktUDPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.
4. The SktClose instruction is used to close the socket.

ST

Internal Variables

Variable	Data type	Initial value	Comment
Trigger	BOOL	False	Execution condition
DoSendAndRcv	BOOL	False	Processing
Stage	INT	0	Stage change
RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
SktUDPCreate_instance	SktUDPCreate		
SktUDPSend_instance	SktUDPSend		
SktUDPRcv_instance	SktUDPRcv		
SktClose_instance	SktClose		

External Variables

Variable	Data type	Constant	Comment
_EIP_EtnOnlineSta	BOOL	☑	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoSendAndRcv=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoSendAndRcv:=TRUE;
    Stage             :=INT#1;
    SktUDPCreate_instance(Execute:=FALSE);  // Initialize instance.
    SktUDPSend_instance(                    // Initialize instance.
          Execute :=FALSE,
          SendDat:=SendSocketDat[0]);       // Dummy
    SktUDPRcv_instance(                     // Initialize instance.
          Execute:=FALSE,
          RcvDat :=RcvSocketDat[0]);        // Dummy
    SktClose_instance(Execute:=FALSE);      // Initialize instance.
END_IF;


IF (DoSendAndRcv=TRUE) THEN
    CASE Stage OF
    1 :                   // Request creating socket.
        SktUDPCreate_instance(
            Execute     :=TRUE,
            SrcUdpPort:=UINT#6000,    // Local UDP port number
            Socket      =>WkSocket);  // Socket

        IF (SktUDPCreate_instance.Done=TRUE) THEN
            Stage:=INT#2;             // Normal end
        ELSIF (SktUDPCreate_instance.Error=TRUE) THEN
            Stage:=INT#10;            // Error end
        END_IF;

    2 :                   // Request sending data
        WkSocket.DstAdr.PortNo:=UINT#6001;
        WkSocket.DstAdr.IpAdr  :='192.168.250.2';
        SktUDPSend_instance(
            Execute :=TRUE,
            Socket  :=WkSocket,       // Socket
            SendDat:=SendSocketDat[0], // Send data
            Size    :=UINT#2000);     // Send data size

        IF (SktUDPSend_instance.Done=TRUE) THEN
            Stage:=INT#3;             // Normal end
        ELSIF (SktUDPSend_instance.Error=TRUE) THEN
            Stage:=INT#20;            // Error end
        END_IF;
```

```
    3 :                     // Request receiving data.
        SktUDPRcv_instance(
            Execute :=TRUE,
            Socket   :=WkSocket,          // Socket
            TimeOut:=UINT#0,              // Timeout time
            Size     :=UINT#2000,         // Receive data size
            RcvDat  :=RcvSocketDat[0]);  // Receive data

        IF (SktUDPRcv_instance.Done=TRUE) THEN
            Stage:=INT#4;                 // Normal end
        ELSIF (SktUDPRcv_instance.Error=TRUE) THEN
            Stage:=INT#30;                // Error end
        END_IF;

    4 :                     // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket  :=WkSocket);          // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;                 // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;                // Error end
        END_IF;

    0 :                     // Normal end
        DoSendAndRcv:=FALSE;
        Trigger           :=FALSE;

    ELSE                    // Interrupted by error.
        DoSendAndRcv:=FALSE;
        Trigger           :=FALSE;
    END_CASE;

END_IF;
```

● Programming in the Remote Node

In this example, programming is also required in the remote node. The order of sending and receiving is reversed in comparison with the above procedure.

1 The SktUDPCreate instruction is used to request creating a UDP socket.

2 The SktUDPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.

3 The SktUDPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.

4 The SktClose instruction is used to close the socket.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	DoSendAndRcv	BOOL	False	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktUDPCreate_instance	SktUDPCreate		
	SktUDPSend_instance	SktUDPSend		
	SktUDPRcv_instance	SktUDPRcv		
	SktClose_instance	SktClose		

External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	☑	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoSendAndRcv=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoSendAndRcv:=TRUE;
    Stage             :=INT#1;
    SktUDPCreate_instance(Execute:=FALSE);  // Initialize instance.
    SktUDPSend_instance(                    // Initialize instance.
            Execute :=FALSE,
            SendDat:=SendSocketDat[0]);     // Dummy
    SktUDPRcv_instance(                     // Initialize instance.
            Execute:=FALSE,
            RcvDat :=RcvSocketDat[0]);      // Dummy
    SktClose_instance(Execute:=FALSE);      // Initialize instance.
END_IF;


IF (DoSendAndRcv=TRUE) THEN
    CASE Stage OF
    1 :                     // Request creating socket.
        SktUDPCreate_instance(
            Execute     :=TRUE,
            SrcUdpPort:=UINT#6001,    // Local UDP port number
            Socket      =>WkSocket);  // Socket

        IF (SktUDPCreate_instance.Done=TRUE) THEN
            Stage:=INT#2;             // Normal end
        ELSIF (SktUDPCreate_instance.Error=TRUE) THEN
            Stage:=INT#10;            // Error end
        END_IF;

    2 :                     // Request receiving data
        WkSocket.DstAdr.PortNo:=UINT#6000;
        WkSocket.DstAdr.IpAdr  :='192.168.250.1';
        SktUDPRcv_instance(
            Execute :=TRUE,
            Socket  :=WkSocket,       // Socket
            TimeOut:=UINT#0,          // Timeout time
            Size    :=UINT#2000,      // Receive data size
            RcvDat  :=RcvSocketDat[0]); // Receive data

        IF (SktUDPRcv_instance.Done=TRUE) THEN
            Stage:=INT#3;             // Normal end
        ELSIF (SktUDPRcv_instance.Error=TRUE) THEN
            Stage:=INT#20;            // Error end
        END_IF;
```

```
    3 :                     // Request sending data.
        SendSocketDat:=RcvSocketDat;
        SktUDPSend_instance(
            Execute :=TRUE,
            Socket   :=WkSocket,            // Socket
            SendDat:=SendSocketDat[0],  // Send data
            Size      :=UINT#2000);         // Send data size

         IF (SktUDPSend_instance.Done=TRUE) THEN
            Stage:=INT#4;                   // Normal end
        ELSIF (SktUDPSend_instance.Error=TRUE) THEN
            Stage:=INT#30;                  // Error end
        END_IF;

    4 :                     // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket  :=WkSocket);            // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;                   // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;                  // Error end
        END_IF;

    0 :                     // Normal end
        DoSendAndRcv:=FALSE;
        Trigger             :=FALSE;

    ELSE                    // Interrupted by error.
        DoSendAndRcv:=FALSE;
        Trigger             :=FALSE;
    END_CASE;

END_IF;
```

SktUDPRcv

The SktUDPRcv instruction reads the data from the receive buffer for a UDP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktUDPRcv	UDP Socket Receive	FB	None	SktUDPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize, SendNodeAdr);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
TimeOut	Timeout time		0: No timeouts 1 to 65535: 0.1 to 6553.5s	Depends on data type.	0.1 s	0
Size	Stored size		The number of bytes to read from the receive buffer	0 to 2000	Bytes	1
RcvDat[] (array)	Receive data	In-out	Receive data	Depends on data type.	---	---
RcvSize	Receive data size	Output	The number of bytes actually stored in *RcvDat[]*	0 to 2000	Bytes	---
SendNodeAdr	Source node address		Source node address	---	---	

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
TimeOut							OK													
Size							OK													
RcvDat[] (array)		OK																		
RcvSize							OK													
SendNodeAdr	Refer to *Function* for details on the structure _sSOCKET_ADDRESS.																			

Function

The SktUDPRcv instruction stores the data in the receive buffer for the socket that is specified with *Socket* in receive data *RcvDat[].* The number of bytes to store is specified with *Size*. The number of bytes that is actually stored is assigned to *RcvSize*. The node address of the node that sent the data is stored in *SendNodeAdr*.

If there is no data in the receive buffer, the instruction waits for data for the time that is set with timeout time *TimeOut*. Storage of the data to *RcvDat[]* is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

The data type of *SendNodeAdr* is structure _sSOCKET_ADDRESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
SendNodeAdr	Source node address	Source node address	_sSOCKET_ ADDRESS	---	---	---
PortNo	Port number	UPD port number of the source node	UINT	1 to 65535	---	---
IpAdr	IP address	IP address of the source node	STRING	Depends on data type.		

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- Up to 2,000 bytes of data can be read from the receive buffer with one instruction.

- If the size of data that was received by the specified socket is smaller than the value of *Size*, then all of the received data is stored in *RecDat[]*. Then size of data that was stored is stored in *RcvSize*.
- If the size of data that was received by the specified socket is larger than the value of *Size*, then the size of received data specified by *Size* is stored in *RecDat[]*.
- The receive data is not read if the value of *Size* is 0.
- If the SktClose instruction closes the connection when there is no data in the receive buffer, a normal end occurs without waiting to receive data even if a timeout has not occurred. The value of *RcvSize* is 0 in that case.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - Data reception is in progress for the socket specified with *Socket*.
  - The socket specified with *Socket* is not open.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming that is provided for the SktUDPCreate instruction (page 2-754).

SktUDPSend

The SktUDPSend instruction sends data from a UDP port for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktUDPSend	UDP Socket Send	FB	None	SktUDPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
SendDat[] (array)	Send data		Send data	Depends on data type.		
Size	Send data size		Send data size	0 to 2000	Bytes	1

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
SendDat[] (array)		OK																		
Size							OK													

Function

The SktUDPSend instruction sends send data *SendDat[]* from the socket that is specified with *Socket*. The number of bytes to send is specified with *Size*. The remote node is specified with *Socket.DstAdr*. Transmission of *SendDat[]* to the send buffer is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	---
IpAdr	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- Up to 2,000 bytes of data can be sent with one instruction. A maximum of 2,000 bytes is sent even if the *SendDat[]* array is larger than 2,000 bytes. Only 1,472 bytes can be sent if the broadcast address is specified.
- If the value of *Size* is 0, then 0 bytes of send data is transmitted on the line.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.

- Data transmission is in progress for the socket specified with *Socket.*
- The socket specified with *Socket* is not open.
- The remote node for *Socket* was specified with a domain name and address resolution failed.
- The handle specified by *Socket.Handle* does not exist.
- The value of *Size* exceeds the number of elements in *SendDat[]*.

Sample Programming

Refer to the sample programming that is provided for the SktUDPCreate instruction (page 2-754).

SktTCPAccept

The SktTCPAccept instruction requests accepting a TCP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCPAccept	Accept TCP Socket	FB	None	SktTCPAccept_instance(Execute, SrcTcpPort, TimeOut, Done, Busy, Error, ErrorID, Socket);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcTcpPort	Local TCP port number	Input	Local TCP port number	1 to 65535	---	1
TimeOut	Timeout time		0: No timeouts 1 to 65535: 0.1 to 6553.5s	Depends on data type.	0.1 s	0
Socket	Socket	Output	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcTcpPort							OK													
TimeOut							OK													
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktTCPAccept instruction requests accepting the port specified with the local TCP port number *ScrTcpPort*. To do this, it executes the Socket(), Bind(), Listen(), and Accept() socket functions. The instruction waits for the time set with timeout time *TimeOut* for a connection to be established with the remote node. The connection is established when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	---
IpAdr	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

- Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.
- You can execute this instruction more than once to open connections to more than one client with one local port number. A different socket is returned for each connection.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- Use the SktClose instruction to close handles that are created with this instruction.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- You can open a maximum of 16 sockets combined for UDP and TCP sockets.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.

- The value of *SrcTcpPort* is outside of the valid range.
- Open processing is in progress for the socket specified with *SrcTcpPort.*
- Close processing is in progress for the socket specified with *SrcTcpPort.*
- A connection is not opened within the time that is specified with *TimeOut*.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-770).

SktTCPConnect

The SktTCPConnect instruction connects to a remote TCP port from the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCP Connect	Connect TCP Socket	FB	None	SktTCPConnect_instance(Execute, SrcTcpPort, DstAdr, DstTcpPort, Done, Busy, Error, ErrorID, Socket);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcTcpPort	Local TCP port number	Input	Local TCP port number. If 0 is specified, an available TCP port that is 1024 or higher is automatically assigned. Well-known port numbers are not assigned.	Depends on data type.	---	0
DstAdr	Destination address		Destination IP address or host name	200 bytes max.		---
DstTcpPort	Destination TCP port number		Destination TCP port number	1 to 65,535		1
Socket	Socket	Output	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcTcpPort							OK													
DstAdr																				OK
DstTcpPort							OK													
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktTCPConnect instruction requests a connection between local TCP port number *SrcTcpPort* and destination TCP port number *DstTcpPort* at destination address *DstAdr*. To do this, it executes the Connect() socket function. The connection is established when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

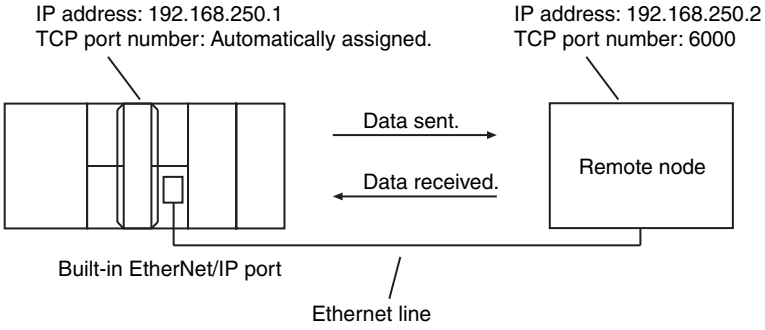
Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- Use the SktClose instruction to close handles that are created with this instruction.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.

- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- You can open a maximum of 16 sockets combined for UDP and TCP sockets.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *DstAdr* is outside of the valid range.
  - The value of *DstTcpPort* is outside of the valid range.
  - The TCP port that is specified with *SrcTcpPort* is already open.
  - The remote node that is specified with *DstAdr* does not exist.
  - The remote node that is specified with *DstAdr* and *DstTcpPort* is not waiting for a connection.
  - Address resolution failed for the host name that is specified with *DstAdr*.
  - A connection is already open for the same client (IP address and TCP port).

Sample Programming

In this sample, the TCP socket service is used for data communications between the NJ-series Controller and a remote node.



The processing procedure is as follows:

1. The SktTCPConnect instruction is used to request connecting to the TCP port on the remote node.
2. The SktClearBuf instruction is used to clear the receive buffer for a TCP socket.
3. The SktGetTCPStatus instruction is used to read the status of a TCP socket.
4. The SktTCPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.
5. The SktTCPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.
6. The SktClose instruction is used to close the socket.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	DoTCP	BOOL	False	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktTCPConnect_instance	SktTCPConnect		
	SktClearBuf_instance	SktClearBuf		
	SktGetTCPStatus_instance	SktGetTCPStatus		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	☑	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
    Stage  :=INT#1;
    SktTCPConnect_instance(Execute:=FALSE);     // Initialize instance.
    SktClearBuf_instance(Execute:=FALSE);       // Initialize instance.
    SktGetTCPStatus_instance(Execute:=FALSE);   // Initialize instance.
    SktTCPSend_instance(                        // Initialize instance.
        Execute:=FALSE,
        SendDat:=SendSocketDat[0]);             // Dummy
    SktTCPRcv_instance(                         // Initialize instance.
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);              // Dummy
    SktClose_instance(Execute:=FALSE);          // Initialize instance.
END_IF;


IF (DoTCP=TRUE) THEN
    CASE Stage OF
    1 :                 // Request a connection.
        SktTCPConnect_instance(
            Execute    :=TRUE,
            SrcTcpPort:=UINT#0,             // Local UDP port number: Automatically assigned.
            DstAdr     :='192.168.250.2',   // Remote IP address
            DstTcpPort:=UINT#6000,          // Destination TCP port number
            Socket     =>WkSocket);         // Socket

        IF (SktTCPConnect_instance.Done=TRUE) THEN
            Stage:=INT#2;                   // Normal end
        ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
            Stage:=INT#10;                  // Error end
        END_IF;

    2 :                 // Clear receive buffer.
        SktClearBuf_instance(
            Execute:=TRUE,
            Socket  :=WkSocket);            // Socket

        IF (SktClearBuf_instance.Done=TRUE) THEN
            Stage:=INT#3;                   // Normal end
        ELSIF (SktClearBuf_instance.Error=TRUE) THEN
            Stage:=INT#20;                  // Error end
        END_IF;
```

```
    3 :                     // Request reading status.
        SktGetTCPStatus_instance(
            Execute:=TRUE,
            Socket  :=WkSocket);          // Socket

        IF (SktGetTCPStatus_instance.Done=TRUE) THEN
            Stage:=INT#4;                 // Normal end
        ELSIF (SktGetTCPStatus_instance.Error=TRUE) THEN
            Stage:=INT#30;                // Error end
        END_IF;

    4 :                     // Request sending data
        SktTCPSend_instance(
            Execute :=TRUE,
            Socket  :=WkSocket,           // Socket
            SendDat:=SendSocketDat[0],// Send data
            Size    :=UINT#2000);         // Send data size

        IF (SktTCPSend_instance.Done=TRUE) THEN
            Stage:=INT#5;                 // Normal end
        ELSIF (SktTCPSend_instance.Error=TRUE) THEN
            Stage:=INT#40;                // Error end
        END_IF;

    5 :                     // Request receiving data
        SktTCPRcv_instance(
            Execute :=TRUE,
            Socket  :=WkSocket,           // Socket
            TimeOut:=UINT#0,              // Timeout time
            Size    :=UINT#2000,          // Receive data size
            RcvDat  :=RcvSocketDat[0]);// Receive data

        IF (SktTCPRcv_instance.Done=TRUE) THEN
            Stage:=INT#6;                 // Normal end
        ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
            Stage:=INT#50;                // Error end
        END_IF;

    6 :                     // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket  :=WkSocket);          // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;                 // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;                // Error end
        END_IF;

    0 :                     // Normal end
        DoTCP:=FALSE;
        Trigger :=FALSE;

    ELSE                    // Interrupted by error.
        DoTCP:=FALSE;
        Trigger :=FALSE;
    END_CASE;

END_IF;
```

● Programming in the Remote Node

In this example, programming is also required in the remote node. The order of sending and receiving is reversed in comparison with the above procedure.

1 The SktTCPAccept instruction is used to request accepting a TCP socket.

2 The SktTCPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.

3 The SktTCPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.

4 The SktClose instruction is used to close the socket.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	DoTCP	BOOL	False	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktTCPAccept_instance	SktTCPAccept		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	☑	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
    Stage  :=INT#1;
    SktTCPConnect_instance(Execute:=FALSE); // Initialize instance.
    SktTCPSend_instance(                    // Initialize instance.
        Execute  :=FALSE,
        SendDat :=SendSocketDat[0]);        // Dummy
    SktTCPRcv_instance(                     // Initialize instance.
        Execute:=FALSE,
        RcvDat :=RcvSocketDat[0]);          // Dummy
    SktClose_instance(Execute:=FALSE);      // Initialize instance.
END_IF;


IF (DoTCP=TRUE) THEN
    CASE Stage OF    // Request accepting a socket connection.
    1 :
        SktTCPAccept_instance(
            Execute    :=TRUE,
            SrcTcpPort:=UINT#6000,      // Local TCP port number
            TimeOut    :=UINT#0         // Timeout time
            Socket     =>WkSocket);     // Socket

        IF (SktTCPAccept_instance.Done=TRUE) THEN
            Stage:=INT#2;               // Normal end
        ELSIF (SktTCPAccept_instance.Error=TRUE) THEN
            Stage:=INT#10;              // Error end
        END_IF;

    2 :                 // Request receiving data
        SktTCPRcv_instance(
            Execute :=TRUE,
            Socket   :=WkSocket,        // Socket
            TimeOut:=UINT#0,            // Timeout time
            Size      :=UINT#2000,      // Receive data size
            RcvDat  :=RcvSocketDat[0]); // Receive data

        IF (SktTCPRcv_instance.Done=TRUE) THEN
            Stage:=INT#3;               // Normal end
        ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
            Stage:=INT#20;              // Error end
        END_IF;
```

```
    3 :                         // Request sending data.
        SendSocketDat:=RcvSocketDat;
        SktTCPSend_instance(
            Execute :=TRUE,
            Socket   :=WkSocket,            // Socket
            SendDat:=SendSocketDat[0], // Send data
            Size       :=UINT#2000);        // Send data size

         IF (SktTCPSend_instance.Done=TRUE) THEN
            Stage:=INT#4;                   // Normal end
        ELSIF (SktTCPSend_instance.Error=TRUE) THEN
            Stage:=INT#30;                  // Error end
        END_IF;

    4 :                         // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket  :=WkSocket);            // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;                   // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;                  // Error end
        END_IF;

    0 :                         // Normal end
        DoTCP:=FALSE;
        Trigger:=FALSE;

ELSE                            // Interrupted by error.
        DoTCP:=FALSE;
        Trigger:=FALSE;
    END_CASE

END_IF;
```

SktTCPRcv

The SktTCPRcv instruction reads the data from the receive buffer for a TCP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCPRcv	TCP Socket Receive	FB	None	SktTCPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
TimeOut	Timeout time		0: No timeouts 1 to 65535: 0.1 to 6553.5s	Depends on data type.	0.1 s	0
Size	Stored size		The number of bytes to read from the receive buffer	0 to 2000	Bytes	1
RcvDat[] (array)	Receive data	In-out	Receive data	Depends on data type.	---	---
RcvSize	Receive data size	Output	The number of bytes actually stored in *RcvDat[]*	1 to 2000	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
TimeOut							OK													
Size							OK													
RcvDat[] (array)		OK																		
RcvSize							OK													

Function

The SktTCPRcv instruction stores the data in the receive buffer for the socket that is specified with *Socket* in receive data *RcvDat[].* The number of bytes to store is specified with *Size*. The number of bytes that is actually stored is assigned to *RcvSize*. If there is no data in the receive buffer, the instruction waits for data for the time that is set with timeout time *TimeOut*. Storage of the data to *RcvDat[]* is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- Up to 2,000 bytes of data can be read with one instruction. A maximum of 2,000 bytes is read even if the *RcvDat[]* array is larger than 2,000 bytes.

- If the size of data that was received by the specified socket is smaller than the value of *Size*, then all of the received data is stored in *RecDat[]*. Then size of data that was stored is stored in *RcvSize*.
- If the size of data that was received by the specified socket is larger than the value of *Size*, then the size of received data specified by *Size* is stored in *RecDat[].*
- The receive data is not read if the value of *Size* is 0.
- If the SktClose instruction closes the connection when there is no data in the receive buffer, an error end occurs even if a timeout has not occurred.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.
  - Data reception is in progress for the socket specified with *Socket.*
  - The socket specified with *Socket* is not connected.
  - The handle specified by *Socket.Handle* does not exist.
  - Data was not received before the time that is specified with *TimeOut* expired.
  - The socket was closed with the SktClose instruction.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-770).

SktTCPSend

The SktTCPSend instruction sends data from a TCP port for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCPSend	TCP Socket Send	FB	None	SktTCPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
SendDat[] (array)	Send data		Send data	Depends on data type.		
Size	Send data size		Send data size	0 to 2000	Bytes	1

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
SendDat[] (array)		OK																		
Size							OK													

Function

The SktTCPSend instruction sends send data *SendDat[]* from the socket that is specified with *Socket*. The number of bytes to send is specified with *Size*.

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- Up to 2,000 bytes of data can be sent with one instruction. A maximum of 2,000 bytes is sent even if the *SendDat[]* array is larger than 2,000 bytes.
- Data is not sent if the value of *Size* is 0.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, and SktClearBuf.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.
  - Data transmission is in progress for the socket specified with *Socket*.

- The socket specified with *Socket* is not connected.
- The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-770).

SktGetTCPStatus

The SktGetTCPStatus instruction reads the status of a TCP socket.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktGetTCP Status	Read TCP Socket Status	FB	None	SktGetTCPStatus_instance(Execute, Socket, Done, Busy, Error, ErrorID, TcpStatus, DatRcvFlag);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
TcpStatus	TCP connection status	Output	TCP connection status	*	---	---
DatRcvFlag	Data Received Flag		TRUE: Data is received. FALSE: Data is not received.	Depends on data type.		

* _CLOSED, _LISTEN, _SYS_SENT, _SYN_RECEIVED, _ESTABLISHED, _CLOSE_WAIT, _FIN_WAIT1, _CLOSING, _LAST_ACK, _FIN_WAIT2, or _TIME_WAIT

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
TcpStatus	Refer to *Function* for the enumerators of the enumerated type _eCONNECTION_STATE.																			
DatRcvFlag	OK																			

Function

The SktGetTCPStatus instruction gets the TCP connection status *TcpStatus* of the socket that is specified with *Socket*. If there is receive data in the receive buffer, the value of data received flag *DatRecvFlag* changes to TRUE. Storage of the data to *TcpStatus* and *DatRcvFlag* is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

The data type of *TcpSta* is enumerated type _eCONNECTION_STATE. The meanings of the enumerators of enumerated type _eCONNECTION_STATE are as follows:

Enumerators	Meaning
_CLOSED	CLOSED status
_LISTEN	LISTEN status
_SYN SENT	SYN SENT status
_SYN RECEIVED	SYN RECEIVED status
_ESTABLISHED	ESTABLISHED status
_CLOSE WAIT	CLOSE WAIT status
_FIN WAIT1	FIN WAIT1 status
_CLOSING	CLOSING status
_LAST ACK	LAST ACK status
_FIN WAIT2	FIN WAIT2 status
_TIME WAIT	TIME WAIT status

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of a member of *Socket* is outside of the valid range.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-770).

SktClose

The SktClose instruction closes the specified TCP or UDP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktClose	Close TCP/UDP Socket	FB	None	SktClose_instance(Execute, Socket, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktClose instruction closes the socket that is specified with *Socket*. If a TCP socket is specified, the socket is disconnected before it is closed. If the socket handle *Socket.Handle* is 0, all TCP and UDP ports that currently use the socket service are closed. Close processing for the TCPUDP sockets is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle of the connection to close. 0: Closes all TCP connections that currently use the socket service.	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- If the SktUDPRcv or SktTCPRcv instruction is executed and then the SktClose instruction is executed while the socket for the specified handle is on standby to received data, the standby status is canceled.
- If more than one connection is open for the same local port number, only the connection for the specified socket is closed.
- If the value of the socket handle *Socket.Handle* is 0, all connections that are on standby for the SktTCPAccept instruction are canceled.

- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming for the following instructions: SktUDPCreate (page 2-754) and Skt-TCPConnect (page 2-770).

SktClearBuf

The SktClearBuf instruction clears the receive buffer for the specified TCP or UDP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktClearBuf	Clear TCP/UDP Socket Receive Buffer	FB	None	SktClearBuf_instance(Execute, Socket, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktClearBuf instruction clears the receive buffer for the socket that is specified with *Socket*. Clear processing of the receive buffer is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	The handle of the socket for which to clear the receive buffer	UDINT	Depends on data type.	---	---
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	---
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta	Online	BOOL	Status of built-in EtherNet/IP port communications TRUE: Can be used. FALSE: Cannot be used.

Additional Information

Refer to the *NJ-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ-series CPU Units.
- This instruction must be used in ST. It cannot be used in a ladder diagram.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, and SktClearBuf.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of a member of *Socket* is outside of the valid range.
  - The socket that is specified by *Socket* does not exist.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

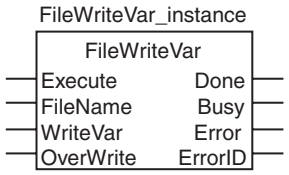
Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-770).

SD Memory Card Instructions

Instruction	Name	Page
FileWriteVar	Write Variable to File	2-794
FileReadVar	Read Variable from File	2-799
FileOpen	Open File	2-803
FileClose	Close File	2-806
FileSeek	Seek File	2-809
FileRead	Read File	2-812
FileWrite	Write File	2-819
FileGets	Get Text String	2-826
FilePuts	Put Text String	2-833
FileCopy	Copy File	2-840
FileRemove	Delete File	2-848
FileRename	Change File Name	2-852
DirCreate	Create Directory	2-857
DirRemove	Delete Directory	2-860

FileWriteVar

The FileWriteVar instruction writes the value of a variable to the specified file in the SD Memory Card. The value is written in binary format.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileWriteVar	Write Variable to File	FB		FileWriteVar_instance(Execute, FileName, WriteVar, OverWrite, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to which to write variable	Depends on data type.	---	"
WriteVar	Variable		Variable to write			*
OverWrite	Overwrite enable		TRUE: Enable overwrite. FALSE: Prohibit overwrite.			FALSE

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
WriteVar	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
OverWrite	OK																			

Function

The FileWriteVar instruction writes the value of variable *WriteVar* to the file specified by *FileName* in the SD Memory Card. The value is written in binary format. You can specify an enumeration, array, array element, structure, or structure member for *WriteVar*.

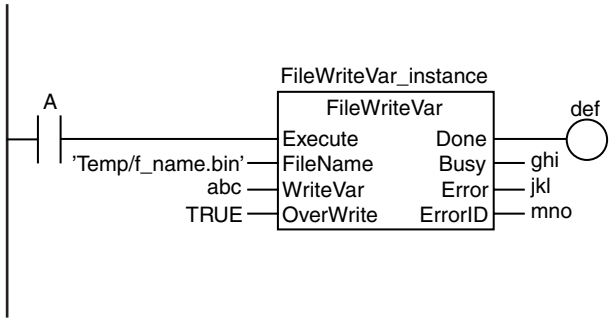
If a file with the name *FileName* does not exist on the SD Memory Card, it is created. *FileName* includes the path. If a specified directory does not exist in the SD Memory Card, it is created.

If a file with the name *FileName* already exists in the SD Memory Card, the following processing is performed depending on the value of overwrite enable *OverWrite*.

Value of *OverWrite*	Processing
TRUE (Enable overwrite.)	The existing file is overwritten.
FALSE (Prohibit overwrite.)	The file is not overwritten and an error occurs.

The following figure shows a programming example. The contents of array variable *abc[0]* is written to a file named 'Temp\f_name.bin.'

LD



ST

```
FileWriteVar_instance(A, 'Temp/f_name.bin', abc,
                      TRUE, def, ghi, jkl, mno);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the specified file is larger than the size of *WriteVar*, an error does not occur and only data that corresponds to the size of *WriteVar* is written. Once this instruction is executed, the specified file is reduced to the size of *WriteVar*.
- Data is written in byte increments. The lower bytes are written before the upper bytes (little endian).
- If *WriteVar* is a structure, adjustment areas between members may be inserted depending on the composition.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The value of *FileName* is not a valid file name.
  - The maximum number of files or directories is exceeded.
  - A file with the name *FileName* already exits and the file is being accessed.
  - A file with the name *FileName* already exits and the value of *OverWrite* is FALSE.
  - A file with the name *FileName* already exits and the file is write protected.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.

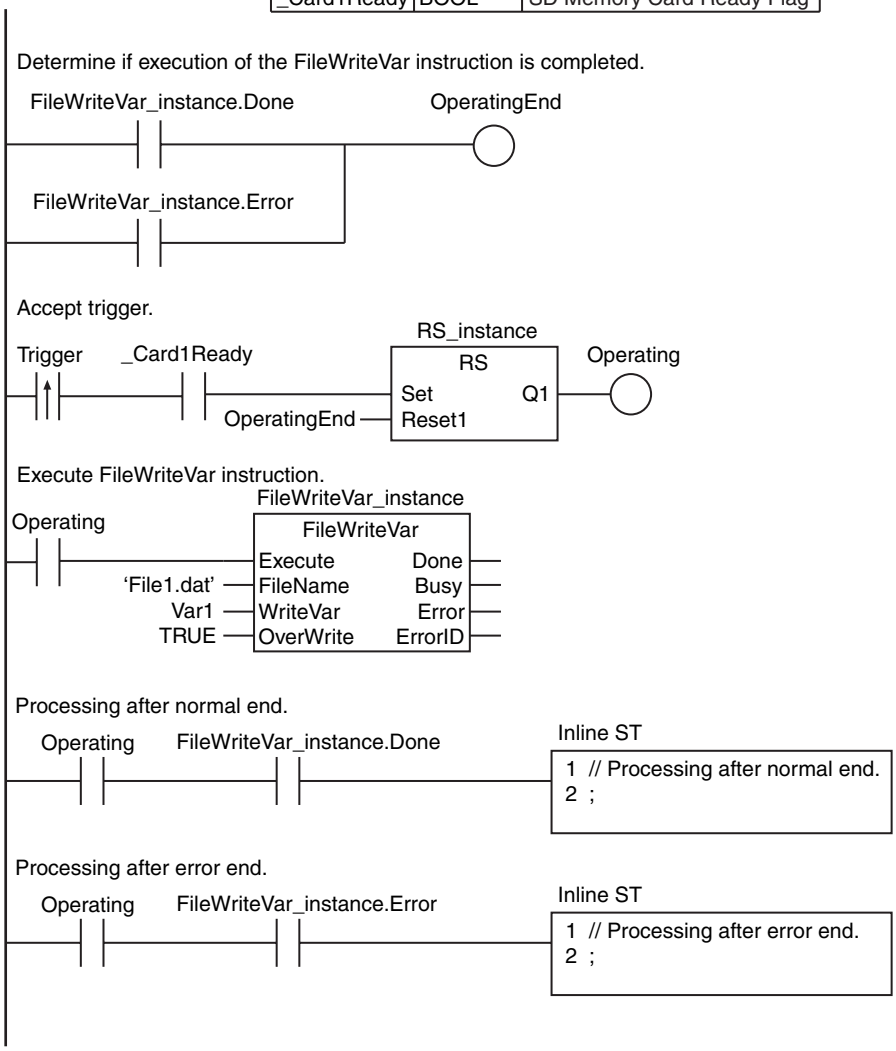
Sample Programming

This sample writes all of array variable *Var1[]* to the file 'File1.dat.'

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Write data
	RS_instance	RS		
	FileWriteVar_instance	FileWriteVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag



ST

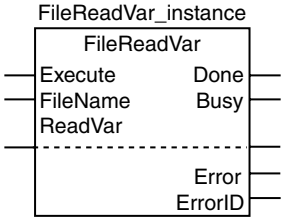
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Variable
	FileWriteVar_instance	FileWriteVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize FileWriteVar instruction.
IF (OperatingStart=TRUE) THEN
    FileWriteVar_instance(
        Execute   :=FALSE,
        WriteVar  :=Var1),
    OperatingStart :=FALSE;
END_IF;

// Execute FileWriteVar instruction.
IF (Operating=TRUE) THEN
    FileWriteVar_instance(
        Execute   :=TRUE,
        FileName :='File1.dat',    // File name
        WriteVar  :=Var1,          // Variable
        OverWrite:=TRUE);          // Enable overwrite.

    IF (FileWriteVar_instance.Done=TRUE) THEN
        // Processing after normal end.
        Operating:=FALSE;
    END_IF;

    IF (FileWriteVar_instance.Error=TRUE) THEN
        // Processing after error end.
        Operating:=FALSE;
    END_IF;
END_IF;
```

FileReadVar

The FileReadVar instruction reads the contents of the specified file on the SD Memory Card as binary data and writes it to a variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileReadVar	Read Variable from File	FB		FileReadVar_instance(Execute, FileName, ReadVar, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to read	Depends on data type.	---	''
ReadVar	Variable to write	In-out	Variable to which to write the value that was read	Depends on data type.	---	---

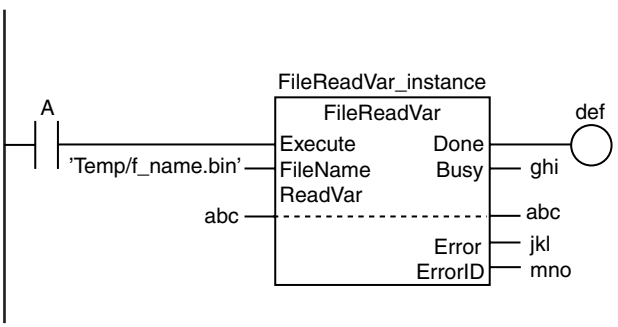
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
ReadVar	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			

Function

The FileReadVar instruction reads the contents of the file specified by *FileName* from the SD Memory Card as binary data. The contents that is read is assigned to variable to write *ReadVar*. You can specify an enumeration, array, array element, structure, or structure member for *ReadVar*.

The following figure shows a programming example. Here, the contents of the file called 'Temp/f_name.bin' is read and written to the array variable *abc[]*.

LD



ST

```
FileReadVar_instance(A, 'Temp/f_name.bin', abc,
                     def, ghi, jkl, mno);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the specified file is larger than the size of *ReadVar*, an error does not occur and only data that corresponds to the size of *ReadVar* is read.
- If the specified file is smaller than the size of *ReadVar*, an error does not occur and only data that corresponds to the size of the specified file is read. The remaining area in *ReadVar* will retain the values from before execution of this instruction.
- Data is read in byte increments. The lower bytes are read before the upper bytes (little endian).
- If *ReadVar* is a structure, adjustment areas between members may be inserted depending on the composition.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The file specified by *FileName* does not exist.
  - The value of *FileName* is not a valid file name.
  - The file specified by *FileName* is being accessed.

- If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
- The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
- An error that prevents access occurs during SD Memory Card access.

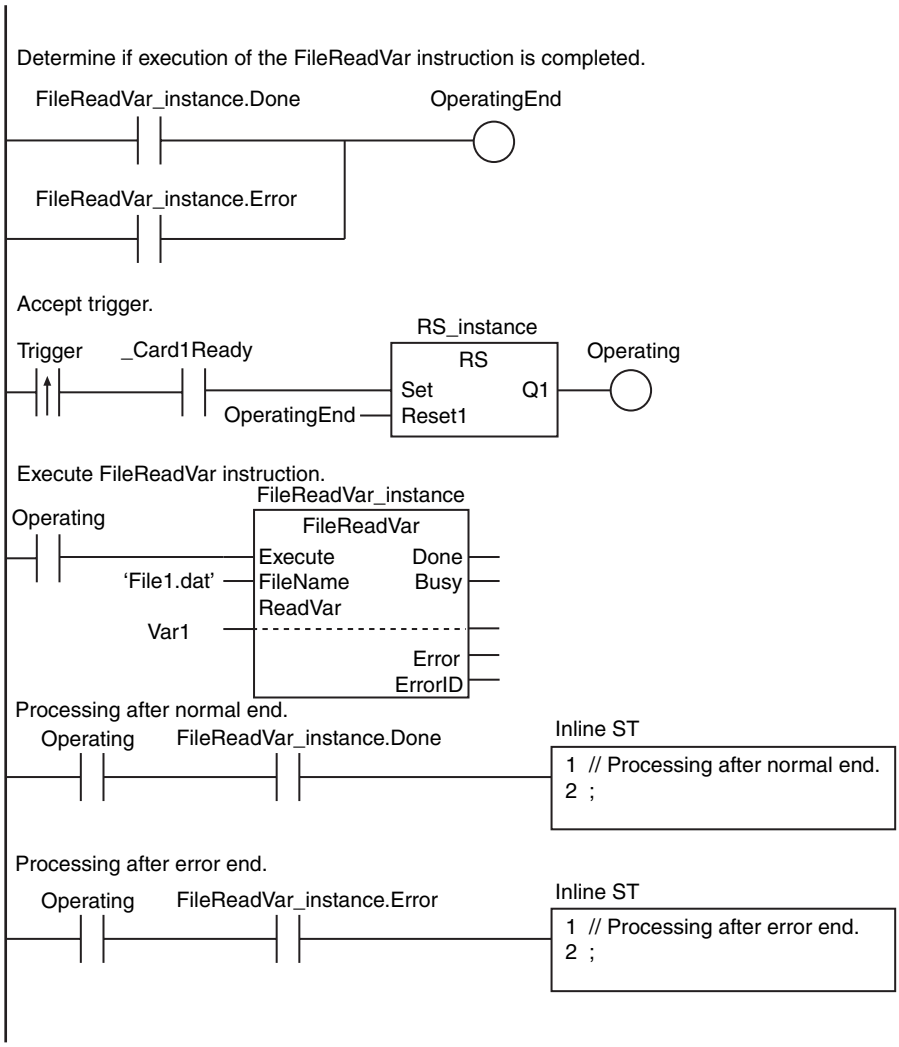
Sample Programming

This sample reads the contents of the file 'File1.dat' and stores it in array variable *Var1*.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Read size
	RS_instance	RS		
	FileReadVar_instance	FileReadVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag



ST

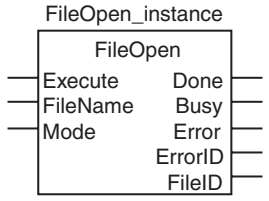
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Variable to read
	FileReadVar_instance	FileReadVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize FileReadVar instruction.
IF (OperatingStart=TRUE) THEN
    FileReadVar_instance(
        Execute :=FALSE,
        ReadVar:=Var1);
    OperatingStart :=FALSE;
END_IF;

// Execute FileReadVar instruction.
IF (Operating=TRUE) THEN
    FileReadVar_instance(
        Execute    :=TRUE,
        FileName :='File1.dat',    // File name
        ReadVar   :=Var1);         // Variable to read

    IF (FileReadVar_instance.Done=TRUE) THEN
        // Processing after normal end.
        Operating:=FALSE;
    END_IF;

    IF (FileReadVar_instance.Error=TRUE) THEN
        // Processing after error end.
        Operating:=FALSE;
    END_IF;
END_IF;
```

FileOpen

The FileOpen instruction opens the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileOpen	Open File	FB		FileOpen_instance(Execute, FileName, Mode, Done, Busy, Error, ErrorID, FileID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to open	Depends on data type.	---	''
Mode	Open mode		Mode in which to open file	*		_READ_EXIST
FileID	File ID	Output	ID of file that was opened	Depends on data type.	---	---

* _READ_EXIST, _RDWR_EXIST, _WRITE_CREATE, _RDWR_CREATE, _WRITE_APPEND and _RDWR_APPEND

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
Mode	Refer to *Function* for the enumerators for the enumerated type _eFOPEN_MODE.																			
FileID				OK																

Function

The FileOpen instruction opens the file specified by *FileName* in the SD Memory Card in the mode specified by *Mode*. The result is output to file ID *FileID*. *FileID* is used to specify the file in other instructions, such as FileRead and FileWrite.

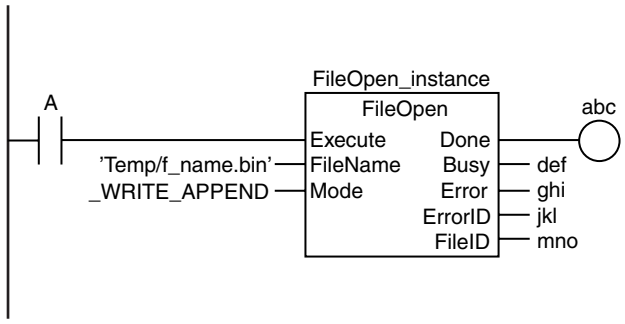
The data type of *Mode* is enumerated type _eFOPEN_MODE. The meanings of the enumerators are as follows:

Enumerator	Meaning
_READ_EXIST	Use this value to open a text file to read it. The file is read from the beginning.
_RDWR_EXIST	Use this value to open a file to read and write it. The file is read and written from the beginning.
_WRITE_CREATE	Use this value to open a file to write it. If the file already exists, the contents is discarded and the file size is set to 0. If the file does not exist, a new file is created. The file is written from the beginning. However, if the file already exists and it is write-protected, an error occurs and the file is not opened.

Enumerator	Meaning
_RDWR_CREATE	Use this value to open a file to read and write it. If the file already exists, the contents is discarded and the file size is set to 0. If the file does not exist, a new file is created. The file is read and written from the beginning.
_WRITE_APPEND	Use this value to open a file to append data to it. If the file does not exist, a new file is created. The data is appended to the end of the file. However, if the file already exists and it is write-protected, an error occurs and the file is not opened.
_RDWR_APPEND	Use this value to open a file to read and append data to it. If the file does not exist, a new file is created. The file is read from the beginning. The data is appended to the end of the file.

The following figure shows a programming example. The file named 'Temp/f_name' is opened to append data to it. The file ID is assigned to variable *mno*.

LD



ST

```
FileOpen_instance(A, 'Temp/f_name.bin',
                  _WRITE_APPEND, abc,
                  def, ghi, jkl, mno);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction must be executed before any of the following instructions: FileSeek, FileRead, FileWrite, FileGets, and FilePuts.
- You must use the FileClose instruction to close any file that is opened with this instruction after you finish using it.
- A value is stored in *FileID* when the instruction is completed. Specifically, it is stored when the value of *Done* changes from FALSE to TRUE.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- If a file is open when the power supply it stopped with the power switch, the file is not corrupted. The file, however, will remain open. Use the FileClose instruction to close the file.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the file will remain open. Use the FileClose instruction to close the file.
- If a file is open when the power supply is stopped or the SD Memory Card is removed, the file will remain open, but it will not be possible to read or write the file even if the SD Memory Card is inserted again. To read/write the file, close the file and then open it again.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - The value of *Mode* is _READ_EXIST or _RDWR_EXIST and the file specified with *FileName* does not exist.
  - The value of *FileName* is not a valid file name.
  - The maximum number of files or directories is exceeded.
  - The file specified by *FileName* is being accessed.
  - The file specified by *FileName* is write protected.
  - An attempt was made to open more than five files at the same time.
  - The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.
  - The value of *Mode* is outside of the valid range.

Sample Programming

Refer to the sample programming for the following instructions: FileRead (page 2-812), FileWrite (page 2-819), FileGets (page 2-826), and FilePuts (page 2-833).

FileClose

The FileClose instruction closes the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileClose	Close File	FB	FileClose_instance FileClose Execute Done FileID Busy Error ErrorID	FileClose_instance(Execute, FileID, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to close	Depends on data type.	---	0

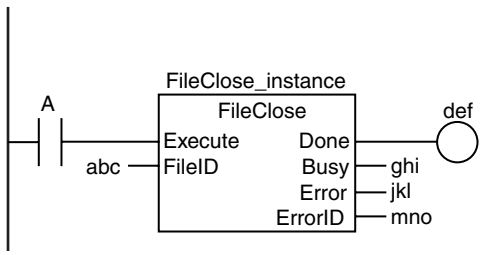
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																

Function

The FileClose instruction closes the file specified by *FileID* in the SD Memory Card.

The following figure shows a programming example. Here, the file whose file ID is the value of variable *abc* is closed.

LD



ST

```
FileClose_instance(A, abc, def, ghi, jkl, mno);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.

Name	Meaning	Data type	Description
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

You must open files with the FileOpen instruction for the following instructions: FileSeek, FileRead, FileWrite, FileGets, and FilePuts.

Precautions for Correct Use

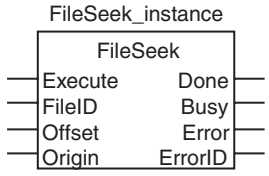
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction in advance to obtain the value for *FileID*.
- You must use this instruction to close any file that is opened with the FileOpen instruction after you finish using it.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- If a file is open when the power supply it stopped with the power switch, the file is not corrupted. The file, however, will remain open. Use the FileClose instruction to close the file.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the file will remain open. Use the FileClose instruction to close the file.
- If a file is open when the power supply is stopped or the SD Memory Card is removed, the file will remain open, but it will not be possible to read or write the file even if the SD Memory Card is inserted again. To read/write the file, close the file and then open it again.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is already closed.
  - The file specified by *FileID* is being accessed.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

Refer to the sample programming for the following instructions: FileRead (page 2-812), FileWrite (page 2-819), FileGets (page 2-826), and FilePuts (page 2-833).

FileSeek

The FileSeek instruction sets a file position indicator in the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileSeek	Seek File	FB		FileSeek_instance(Execute, FileID, Offset, Origin, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file in which to set file position indicator	Depends on data type.	---	0
Offset	Offset		Offset from *Origin*		Bytes	
Origin	Reference position		Reference position for file position indicator	_SEEK_SET, _SEEK_CUR, or _SEEK_END	---	_SEEK_SET

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
Offset												OK								
Origin	Refer to *Function* for the enumerators for the enumerated type _eFSEEK_ORIGIN.																			

Function

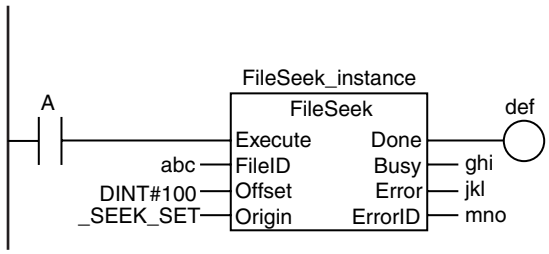
The FileSeek instruction sets a file position indicator in the file specified by file ID *FileID* in the SD Memory Card. A file position indicator is the position in a file at which to start reading or writing when an instruction such as the FileRead or FileWrite instruction is executed. For example, to read from the beginning of a file, set a file position indicator at the beginning of the file with the FileSeek instruction, and then execute the FileRead instruction. The file position indicator is set at offset *Offset* from reference position *Origin*.

The data type of *Origin* is enumerated type _eFSEEK_ORIGIN. The meanings of the enumerators are as follows:

Enumerator	Meaning
_SEEK_SET	Beginning of file
_SEEK_CUR	Location of current file position indicator
_SEEK_END	End of file

The following figure shows a programming example. A file position indicator is set at 100 bytes from the beginning of the file.

LD



ST

```
FileSeek_instance(A, abc, DINT#100,
                  _SEEK_SET, def,
                  ghi, jkl, mno);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *Origin* is outside of the valid range.
  - The SD Memory Card is not in a usable condition.

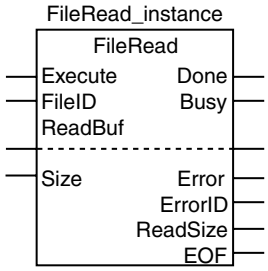
- The file specified by *FileID* does not exist.
- The file specified by *FileID* is being accessed.
- The position specified by *Origin* and *Offset* exceeds the file size.
- An error that prevents access occurs during SD Memory Card access.

Sample Programming

Refer to the sample programming for the following instructions: FileRead (page 2-812) and FileWrite (page 2-819).

FileRead

The FileRead instruction reads the data from the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileRead	Read File	FB		FileRead_instance(Execute, FileID, ReadBuf, Size, Done, Busy, Error, ErrorID, ReadSize, EOF);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to read	Depends on data type.	---	0
Size	Number of elements to read		Number of elements to read			1
ReadBuf[] (array)	Read buffer	In-out	Buffer in which to write data that was read	Depends on data type.	---	---
ReadSize	Number of read elements	Output	Number of elements that were actually read	Depends on data type.	---	---
EOF	End of file		Whether end of file was reached TRUE: Reached. FALSE: Not reached.			---

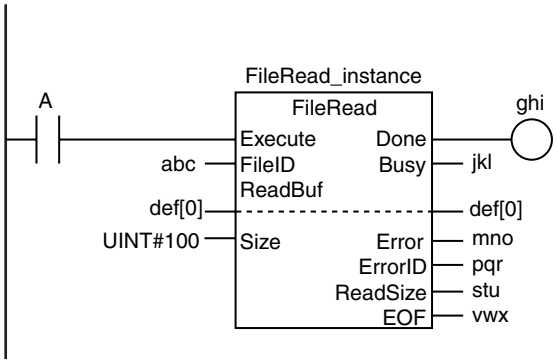
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
Size							OK													
ReadBuf[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays enumerations or structures can also be specified.																			
ReadSize							OK													
EOF	OK																			

Function

The FileRead instruction reads the data from position of the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. It then stores the data in read buffer *ReadBuf[]*. The file position indicator is set at the desired location in advance with the FileSeek instruction. The amount of data that is read is the size of the data type of *ReadBuf[]* times *Size*. You can specify an array of enumerations or structures for *ReadBuf[]*. The actual number of elements that were read is stored in *ReadSize*. Normally, *Size* and *ReadSize* will have the same values. If the amount of data from the file position indicator to the end of the file is smaller than *Size*, an error will not occur and the data to the end of the file is stored in *ReadBuf[]*. If that occurs, the value of *ReadSize* will be smaller than the value of *Size*. If data is read to the end of the file, end of file *EOF* changes to TRUE. Otherwise, the value of *EOF* will be FALSE.

The following figure shows a programming example. If the read buffer *def[]* is a BYTE array, 100 bytes of data is read from the file.

LD



ST

```
FileRead_instance(A, abc, def[0], UINT#100,
                  ghi, jkl, mno, pqr, stu, vwx);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.

Name	Meaning	Data type	Description
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the data is read to the end of the file and the size of the data is not evenly divisible by the size of the data type of *ReadBuf[]*, the data that is insufficient for the data size of *ReadBuf[]* is discarded. The file position indicator advances to the end of the file, and the value of *EOF* changes to TRUE.
- Elements beyond *Size* times *ReadBuf[]* (i.e., the elements not overwritten when data is read) will retain the values from before execution of this instruction.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- A value is stored in *EOF* when the instruction is completed. Specifically, it is stored when the value of *Done* changes from FALSE to TRUE.
- If *ReadBuf[]* is an array of structures, adjustment areas between members may be inserted depending on the composition.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The number of array elements in *ReadBuf[]* is smaller than the value of *Size*.
  - The SD Memory Card is not in a usable condition.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The file specified by *FileID* was not opened in a reading mode.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

In this sample, four bytes of data are read from the second byte from beginning of the file named 'ABC.bin.' The data is written to BYTE array variable *InDat[]*. The processing procedure is as follows:

1 The FileOpen instruction is used to open the file ‘ABC.bin.’

2 The FileSeek instruction is used to set a file position indicator at the second byte from the beginning of the file.

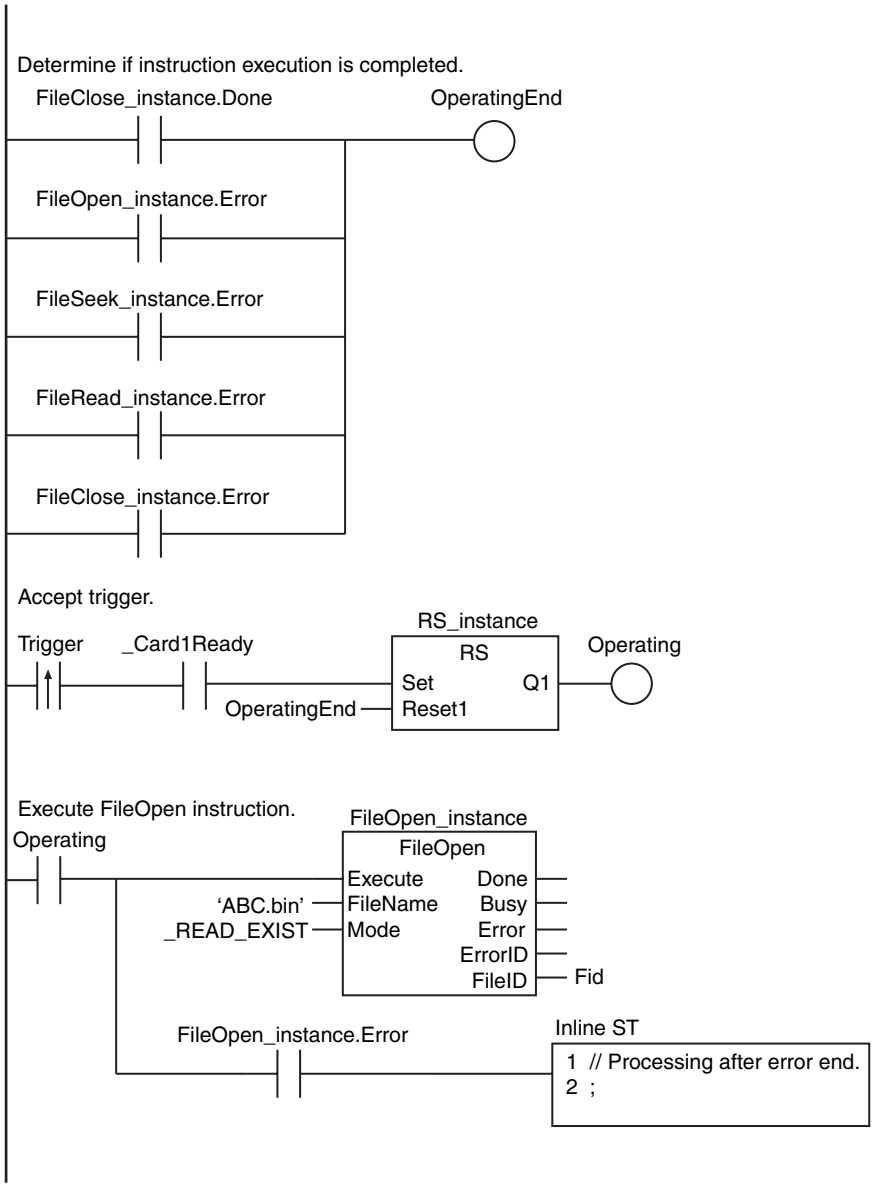
3 The FileRead instruction is used to read four bytes of data from the position of the file position indicator and store it in array variable *InDat[]*.

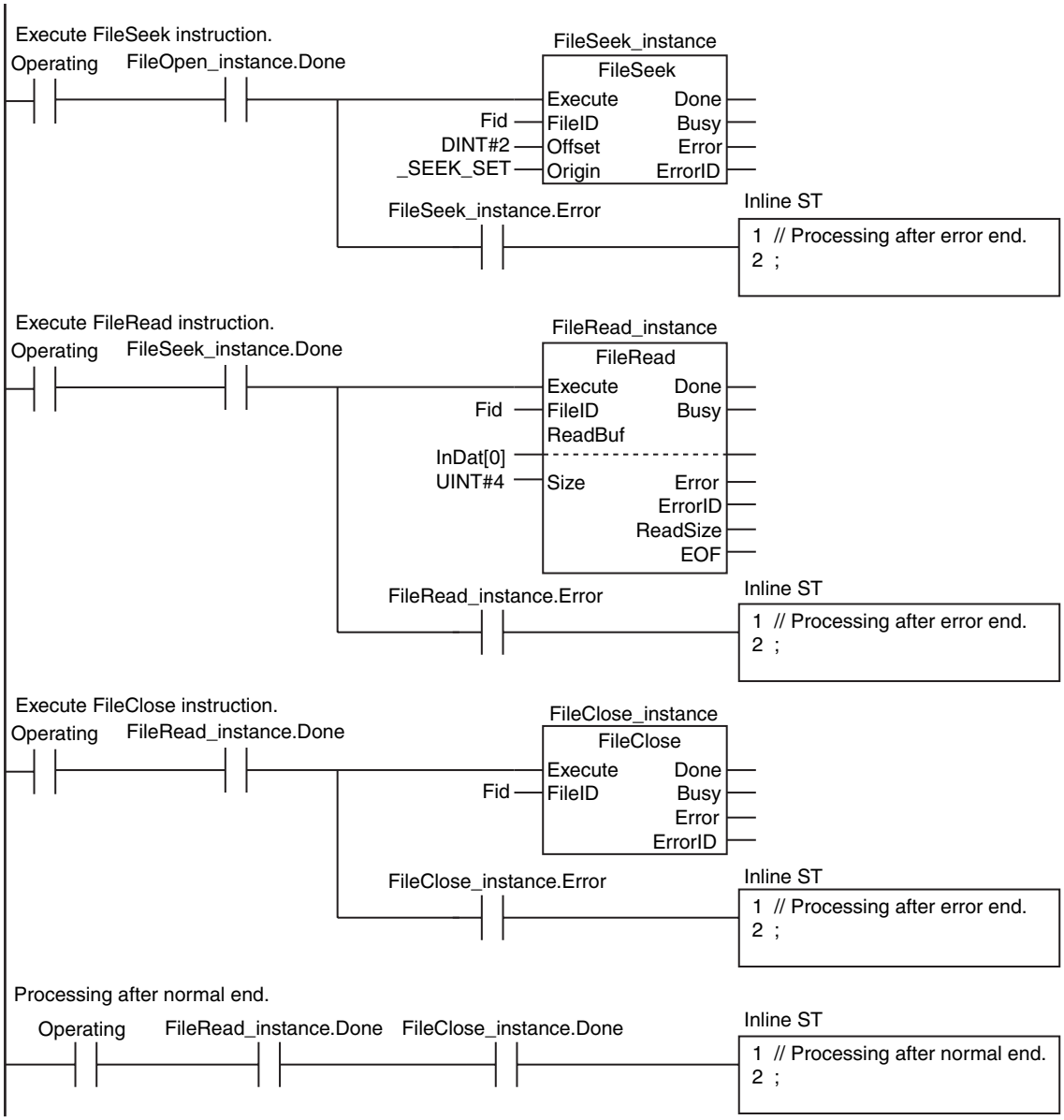
4 The FileClose instruction is used to close the file ‘ABC.bin.’

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Fid	DWORD	16#0	File ID
	InDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Read data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileRead_instance	FileRead		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag




Execute FileSeek instruction.
Operating
FileOpen_instance.Done
FileSeek_instance
FileSeek
Execute
Done
Fid
FileID
Busy
DINT#2
Offset
Error
_SEEK_SET
Origin
ErrorID
FileSeek_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileRead instruction.
Operating
FileSeek_instance.Done
FileRead_instance
FileRead
Execute
Done
Fid
FileID
Busy
ReadBuf
InDat[0]
UINT#4
Size
Error
ErrorID
ReadSize
EOF
FileRead_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileClose instruction.
Operating
FileRead_instance.Done
FileClose_instance
FileClose
Execute
Done
Fid
FileID
Busy
Error
ErrorID
FileClose_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileRead_instance.Done
FileClose_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	InDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Read data
	Stage	INT	0	Stage change
	Fid	DWORD	16#0	File ID
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileRead_instance	FileRead		
	FileClose_instance	FileClose		

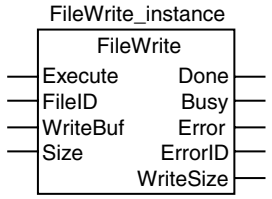
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE); // Initialize instance.
    FileSeek_instance(Execute:=FALSE); // Initialize instance.
    FIleRead_instance(
        Execute:=FALSE,                // Initialize instance.
        ReadBuf:=InDat[0]);            // Dummy
    FileClose_instance(Execute:=FALSE); // Initialize instance.
    Stage          :=INT#1;
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                        // Open file.
        FileOpen_instance(
            Execute  :=TRUE,
            FileName:='ABC.bin',   // File name
            Mode     :=_READ_EXIST, // Read file.
            FileID   =>Fid);       // File ID

        IF (FIleOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;          // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;         // Error end
        END_IF;

    2 :                        // Seek file.
        FileSeek_instance(
            Execute:=TRUE,
            FileID  :=Fid,             /// File ID
            Offset  :=DINT#2,          // File position indicator goes to second byte from the beginning.
            Origin  :=_SEEK_SET);      //

        IF (FileSeek_instance.Done=TRUE) THEN
            Stage:=INT#3;          // Normal end
        END_IF;

        IF (FileSeek_instance.Error=TRUE) THEN
            Stage:=INT#99;         // Error end
        END_IF;
```

```
    3 :                         // Read file.
        FileRead_instance(
            Execute :=TRUE,
            FileID    :=Fid,                    // File ID
            ReadBuf:=InDat[0],                  // Read buffer
            Size      :=UINT#4);                // Number of elements to read: 4 bytes

        IF (FIleRead_instance.Done=TRUE) THEN
            Stage:=INT#4;             // Normal end
        END_IF;

        IF (FileRead_instance.Error=TRUE) THEN
            Stage:=INT#99;            // Error end
        END_IF;

    4 :                         // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID    :=Fid);                   // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;         // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;            // Error end
        END_IF;

    99 :
        Operating:=FALSE;             // Processing after error end.
    END_CASE;
END_IF;
```

FileWrite

The FileWrite instruction writes data to the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileWrite	Write File	FB		FileWrite_instance(Execute, FileID, WriteBuf, Size, Done, Busy, Error, ErrorID, WriteSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to write	Depends on data type.	---	0
WriteBuf[] (array)	Write buffer		Write data			*
Size	Number of elements to write		Number of elements to write			1
WriteSize	Number of written elements	Output	Number of elements that were actually written	Depends on data type.	---	---

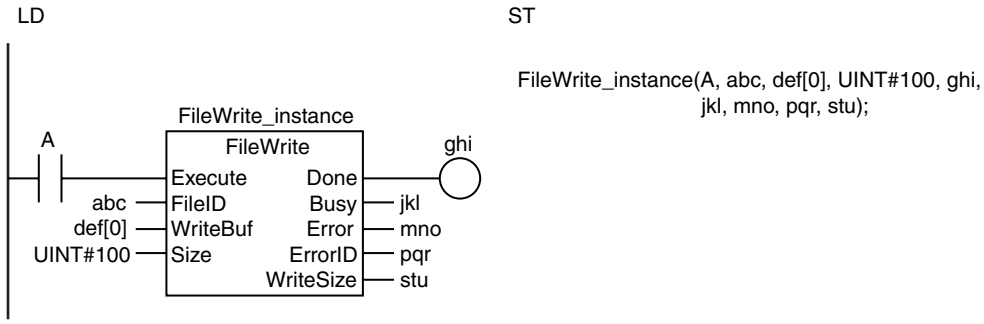
* If you omit the input parameter, the default value is not applied. A compiling error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
WriteBuf[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Size							OK													
WriteSize							OK													

Function

The FileWrite instruction writes data to the position of the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. The file position indicator is set at the desired location in advance with the FileSeek instruction. The contents of the write buffer *WriteBuf[]* is written to the file. The amount of data that is written is the size of the data type of *WriteBuf[]* times *Size*. You can specify an array of enumerations or structures for *WriteBuf[]*. The data size that is actually written is output to *WriteSize*.

The following figure shows a programming example. If the write buffer *def[]* is BYTE data, 100 bytes of data is written to the file.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- Data is written in byte increments. The lower bytes are written before the upper bytes (little endian).

- If *WriteBuf[]* is an array of structures, adjustment areas between members may be inserted depending on the composition.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The number of array elements in *WriteBuf[]* is smaller than the value of Size.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The file specified by *FileID* was not opened in a writing mode.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

Here, four bytes of data are written from the second byte from the beginning of the file 'ABC.bin.' The contents of the BYTE array variable *OutDat[]* is written to the file. The processing procedure is as follows:

1 The FileOpen instruction is used to open the file ‘ABC.bin.’

2 The FileSeek instruction is used to set a file position indicator at the second byte from the beginning of the file.

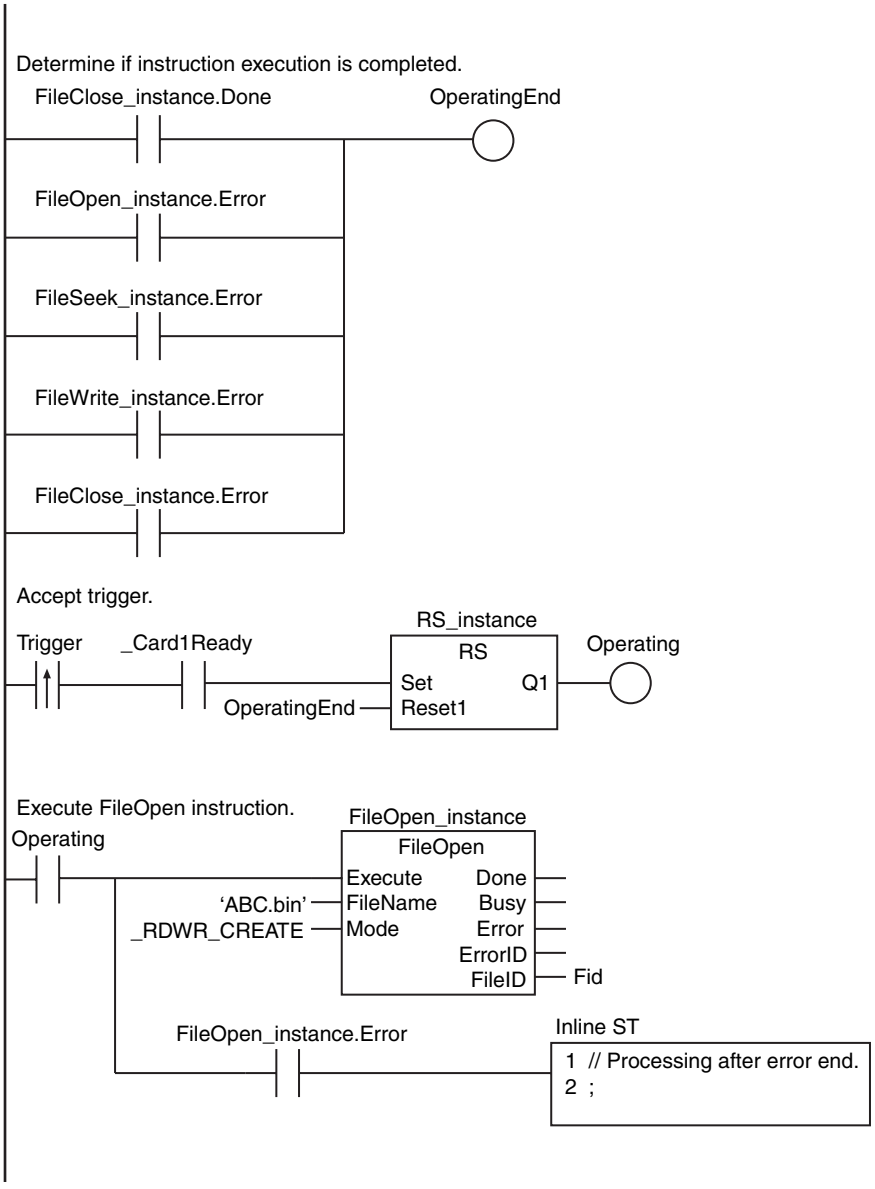
3 The FileWrite instruction is used to write four bytes from array variable *OutDat[]* to the position of the file position indicator.

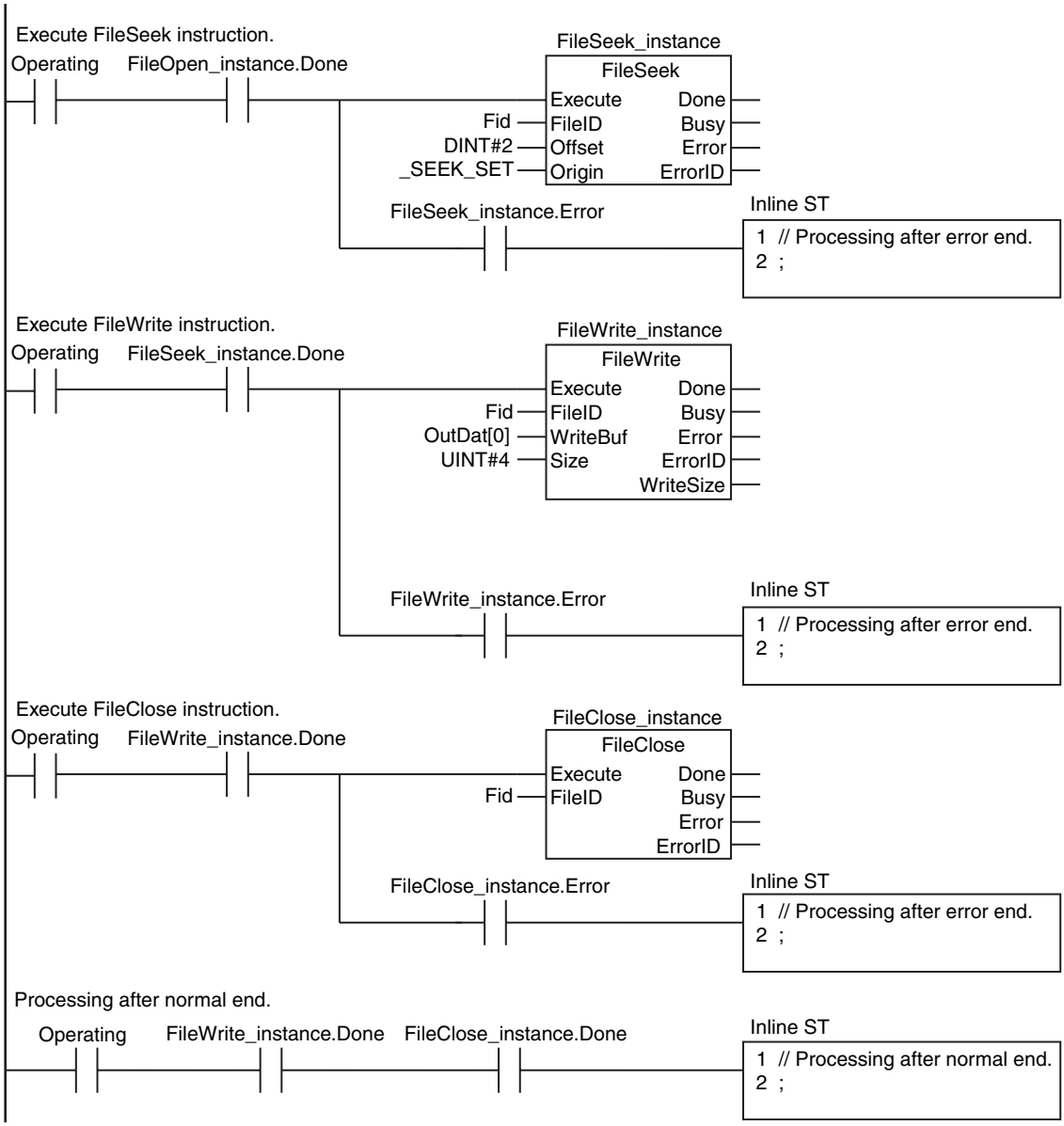
4 The FileClose instruction is used to close the file ‘ABC.bin.’

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Fid	DWORD	16#0	File ID
	OutDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Write data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileWrite_instance	FileWrite		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag




Execute FileSeek instruction.
Operating
FileOpen_instance.Done
FileSeek_instance
FileSeek
Execute
Done
Fid
FileID
Busy
DINT#2
Offset
Error
_SEEK_SET
Origin
ErrorID
FileSeek_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileWrite instruction.
Operating
FileSeek_instance.Done
FileWrite_instance
FileWrite
Execute
Done
Fid
FileID
Busy
OutDat[0]
WriteBuf
Error
UINT#4
Size
ErrorID
WriteSize
FileWrite_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileClose instruction.
Operating
FileWrite_instance.Done
FileClose_instance
FileClose
Execute
Done
Fid
FileID
Busy
Error
ErrorID
FileClose_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileWrite_instance.Done
FileClose_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	OutDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Source data
	Stage	INT	0	Stage change
	Fid	DWORD	16#0	File ID
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileWrite_instance	FileWrite		
	FileClose_instance	FileClose		

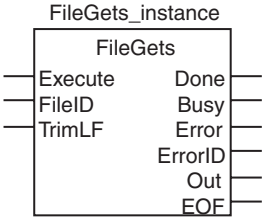
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FileSeek_instance(Execute:=FALSE);
    FileWrite_instance(
            Execute   :=FALSE,
            WriteBuf  :=OutDat[0]);
    FileClose_instance(Execute:=FALSE);
    Stage          :=INT#1;
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                         // Open file.
        FileOpen_instance(
            Execute   :=TRUE,
            FileName:='ABC.bin',          // File name
            Mode      :=_RDWR_CREATE,     // Read file and write.
            FileID    =>Fid);             // File ID

        IF (FIleOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;                 // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;                // Error end
        END_IF;

    2 :                         // Seek file.
        FileSeek_instance(
            Execute:=TRUE,
            FileID   :=Fid,               // File ID
            Offset   :=DINT#2,            // File position indicator goes to second byte from the beginning.
            Origin   :=_SEEK_SET);        //

        IF (FileSeek_instance.Done=TRUE) THEN
            Stage:=INT#3;
        END_IF;                           // Normal end

        IF (FileSeek_instance.Error=TRUE) THEN
            Stage:=INT#99;
        END_IF;                           // Error end
```

```
    3 :                         // Write file.
        FileWrite_instance(
            Execute :=TRUE,
            FileID     :=Fid,                   // File ID
            WriteBuf:=OutDat[0],                // Write buffer
            Size       :=UINT#4);               // Number of elements to write: 4 bytes

        IF (FIleWrite_instance.Done=TRUE) THEN
            Stage:=INT#4;               // Normal end
        END_IF;

        IF (FileWrite_instance.Error=TRUE) THEN
            Stage:=INT#99;              // Error end
        END_IF;

    4 :                         // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID    :=Fid);                   // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;         // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;              // Error end
        END_IF;

    99 :
        Operating:=FALSE;               // Processing after error end.
    END_CASE;
END_IF;
```

FileGets

The FileGets instruction reads a text string of one line from the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileGets	Get Text String	FB		FileGets_instance(Execute, FileID, TrimLF, Done, Busy, Error, ErrorID, Out, EOF);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to read	Depends on data type.	---	0
TrimLF	Line feed designation		Handling of the line feed code of text string that was read TRUE: Delete. FALSE: Do not delete.			FALSE
Out	Read text string	Output	Text string that was read	Depends on data type.	---	---
EOF	End of file		Whether end of file was reached TRUE: Reached. FALSE: Not reached.			

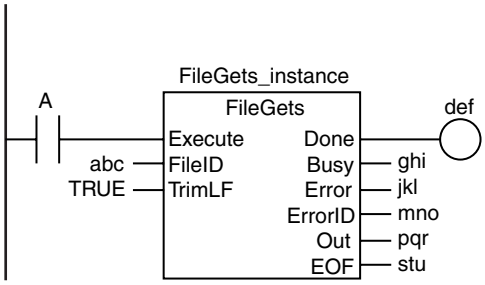
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
TrimLF	OK																			
Out																				OK
EOF	OK																			

Function

The FileGets instruction reads a text string of one line from the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. The file position indicator is set at the desired location in advance with the FileSeek instruction. Line endings are determined by a line feed code. The text string that is read is written to read text string *Out*. The following three line feeds are automatically detected: CR, LF, and CR+LF. If line feed designation *TrimLF* is TRUE, the line feed code is deleted from the text string before it is written to *Out*. If data is read to the end of the file, end of file *EOF* changes to TRUE. Otherwise, the value of *EOF* will be FALSE.

The following figure shows a programming example. Here, a text string of one line is read from a file, the line feed code is deleted, and the result is written to *pqr*.

LD



ST

```
FileGets_instance(A, abc, TRUE, def, ghi,
                  jkl, mno, pqr, stu);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

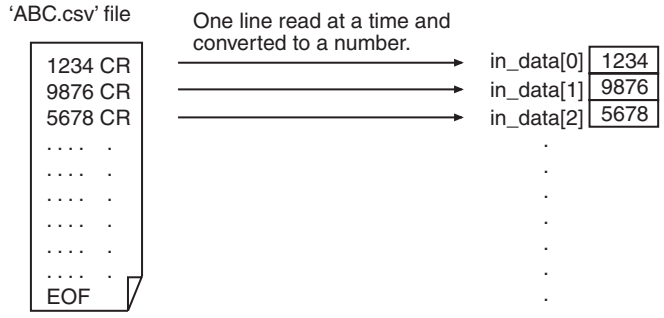
Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the length of the one-line text string exceeds 1,986 bytes (with UTF-8 character codes, including the final NULL character), the first 1,985 bytes of the text string are stored in *Out* with a NULL character attached.

- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The file specified by *FileID* was not opened in a reading mode.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

Here, multiple text strings that are separated by CR codes are stored in a file named 'ABC.csv.' All of them are text strings of numbers. One line at a time is read from the file, the text strings are converted to integers, and the results are stored in INT array variable *InDat[]*. Processing is ended when all of the data to the end of the file is read.
It is assumed that this sample programming is in a periodic task.



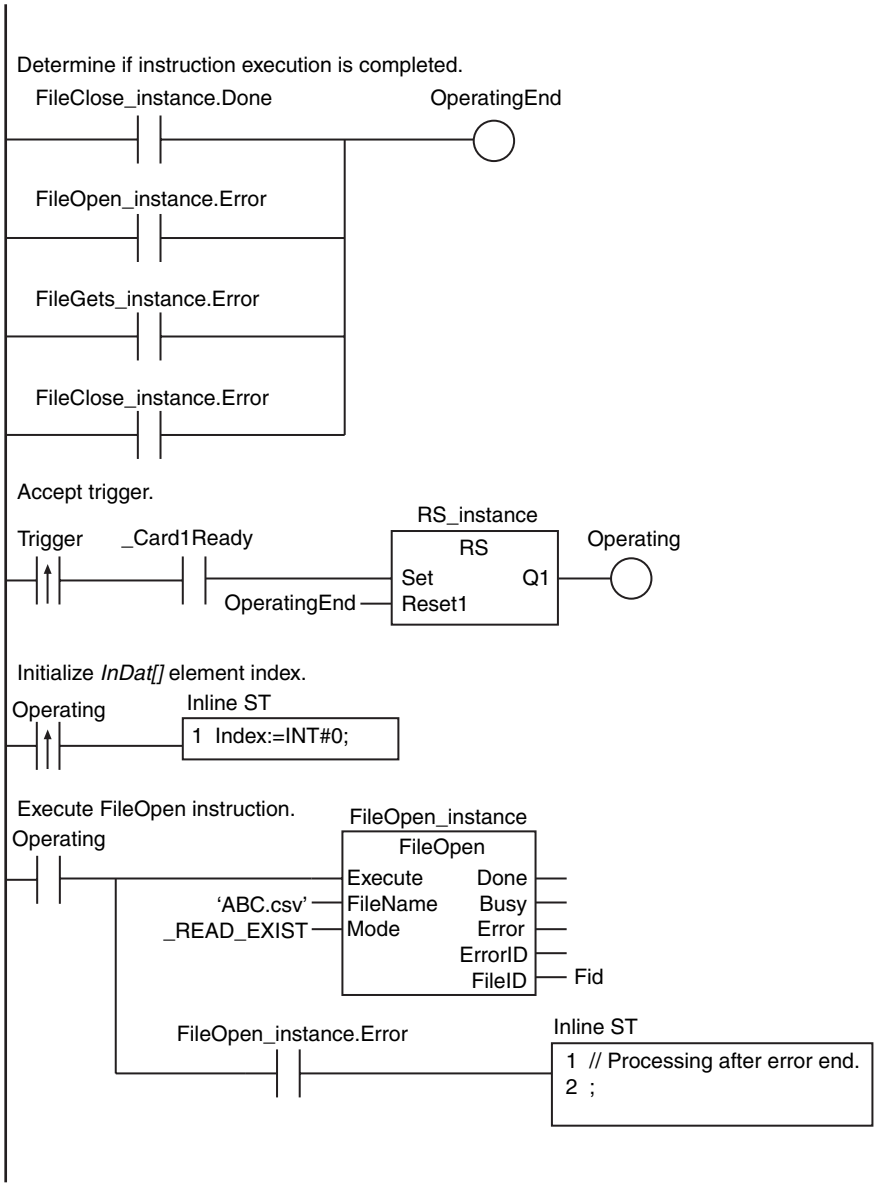
The processing procedure is as follows:

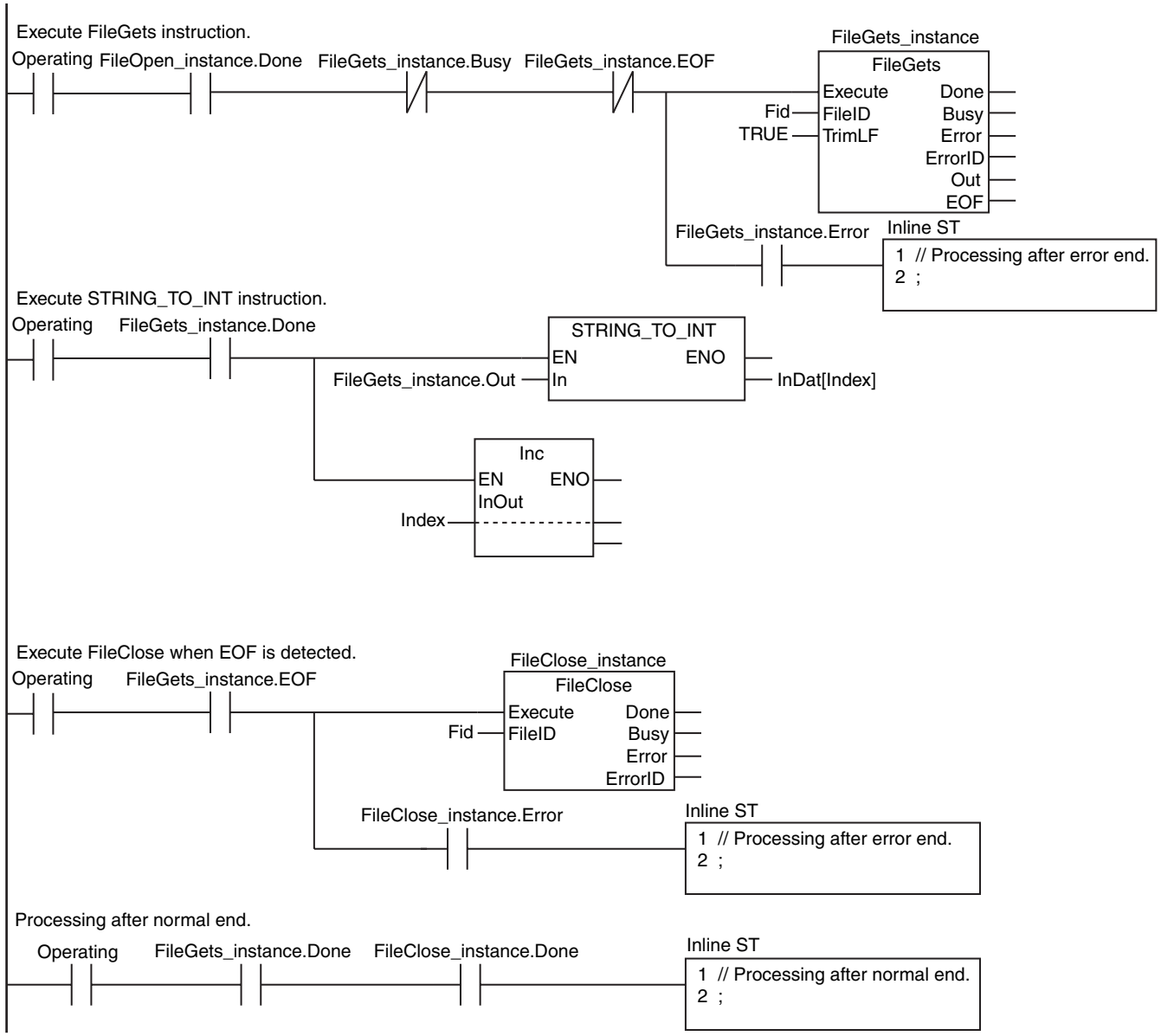
1. The FileOpen instruction is used to open the file 'ABC.csv.'
2. The FileGets instruction is used to read one line from the file.
3. The STRING_TO_INT instruction is used to convert the text string that was read to an integer and store it in *InDat[]*.
4. Steps 2 and 3 are repeated until the EOF (end of file).
5. The FileClose instruction is used to close the file.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Index	INT	0	*InDat[]* element index
	Fid	DWORD	16#0	File ID
	InDat	ARRAY[0..999] OF INT	[1000(0)]	Integer data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag




Execute FileGets instruction.
Operating
FileOpen_instance.Done
FileGets_instance.Busy
FileGets_instance.EOF
FileGets_instance
FileGets
Execute
FileID
TrimLF
Done
Busy
Error
ErrorID
Out
EOF
Fid
TRUE
FileGets_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute STRING_TO_INT instruction.
Operating
FileGets_instance.Done
STRING_TO_INT
EN
ENO
In
FileGets_instance.Out
InDat[Index]
Inc
EN
ENO
InOut
Index
Execute FileClose when EOF is detected.
Operating
FileGets_instance.EOF
FileClose_instance
FileClose
Execute
FileID
Done
Busy
Error
ErrorID
Fid
FileClose_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileGets_instance.Done
FileClose_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	InDat	ARRAY[0..999] OF INT	[1000(0)]	Integer data
	Stage	INT	0	Stage change
	Index	INT	0	*InDat[]* element index
	Fid	DWORD	16#0	File ID
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FileGets_instance(Execute:=FALSE);
    FileClose_instance(Execute:=FALSE);
    Stage          :=INT#1;
    Index          :=INT#0;
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                         // Open file.
        FileOpen_instance(
            Execute  :=TRUE,
            FileName:='ABC.csv',          // File name
            Mode     :=_READ_EXIST,       // Read file.
            FileID   =>Fid);              // File ID

        IF (FIleOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;               // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;              // Error end
        END_IF;
```

```
    2 :                         // Read text string.
        FileGets_instance(
            Execute:=TRUE,
            FileID    :=Fid,
            TrimLF  :=TRUE);

        IF (FileGets_instance.Done=TRUE) THEN
            // Convert the text string that was read to an integer.
            InDat[Index]:=STRING_TO_INT(FileGets_instance.Out);
            Index:=Index+INT#1;

            // Reached end of file.
            IF (FileGets_instance.EOF=TRUE) THEN
                Stage:=INT#3;          // Normal end
            ELSE
                FileGets_instance(Execute:=FALSE);
            END_IF;
        END_IF;

        IF (FileGets_instance.Error=TRUE) THEN
            Stage:=INT#99;           // Error end
        END_IF;

    3 :                         // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID    :=Fid);                    // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;       // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;           // Error end
        END_IF;

    99 :                        // Processing after error end.
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FilePuts

The FilePuts instruction writes a text string to the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FilePuts	Put Text String	FB	FilePuts_instance FilePuts Execute Done FileID Busy In Error ErrorID	FilePuts_instance(Execute, FileID, In, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to write	Depends on data type.	---	0
In	Write text string		Text string to write			"

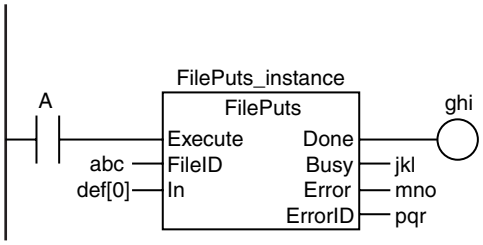
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
In																				OK

Function

The FilePuts instruction writes a text string to the position of the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. The file position indicator is set at the desired location in advance with the FileSeek instruction. The contents of write text string *In* is written to the file.

The following figure shows a programming example. Here, the contents of array element *def[0]* is written to the file.

LD



ST

```
FilePuts_instance(A, abc, def[0], ghi, jkl, mno, pqr);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

To create a line feed after you write the text sting, add a line feed code to the end of *In*.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The file specified by *FileID* was not opened in a writing mode.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

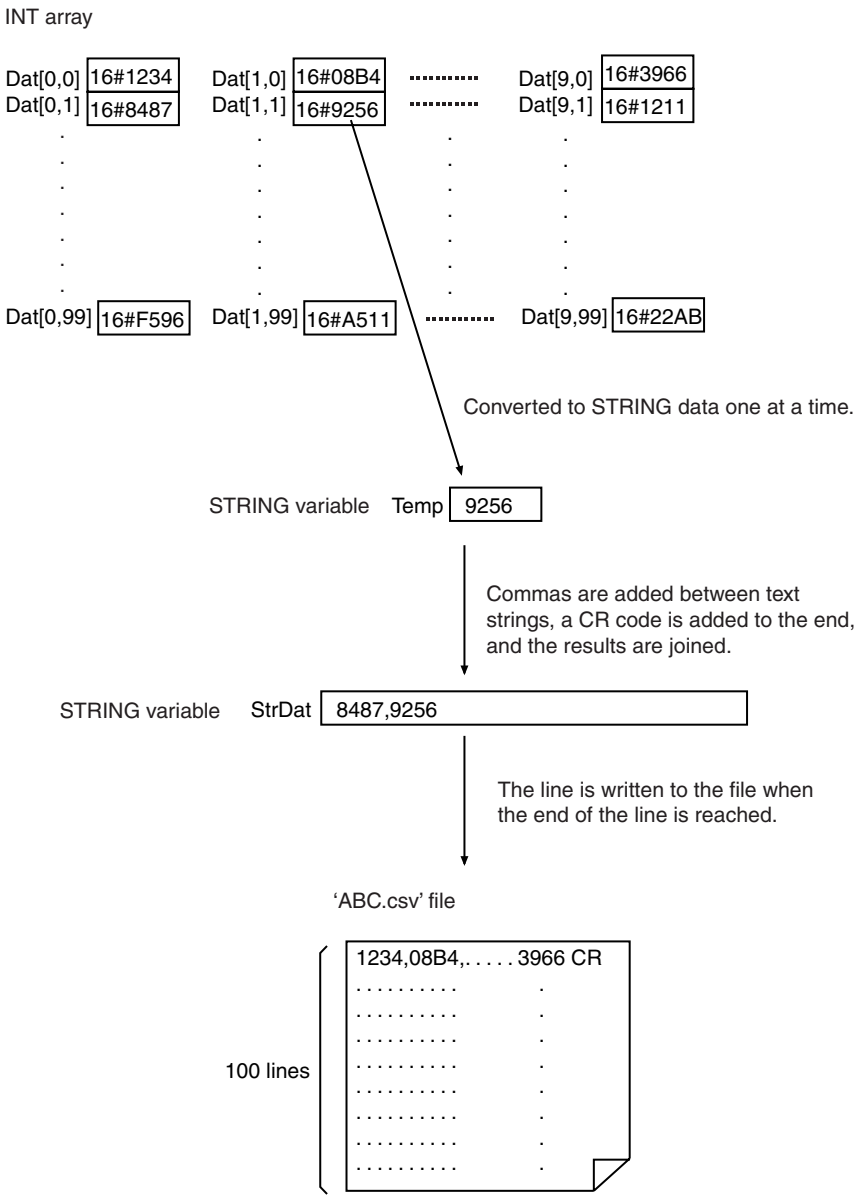
Here, 100 lines of the contents of INT array variable *Dat[0..9,0..99]* are stored in a file named 'ABC.csv' in CSV file format. Each line contains ten text strings of numbers. Commas are inserted between them. A CR+LF code is added to the end of the line. The procedure is as follows:

1 One element of *Dat[]* is converted to one text string and stored in the STRING variable *Temp*.

2 Except at the end of a line, a comma is added to the end of *Temp*. At the end of the line, a CR+LF code is added to the end of *Temp*. These are joined in the STRING variable *StrDat*.

3 When the end of the line is reached, *StrDat* is written to the file.

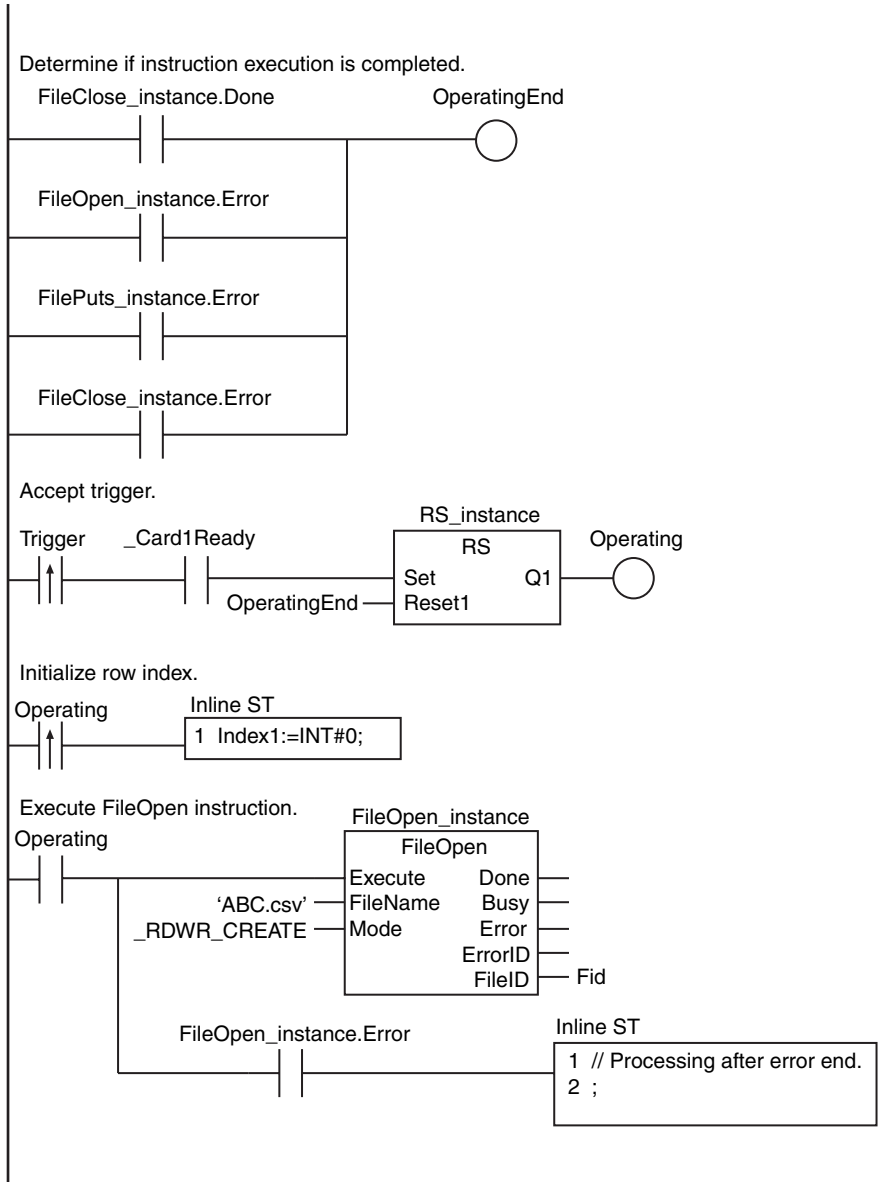
4 Steps 1 to 3 are repeated for 100 lines.

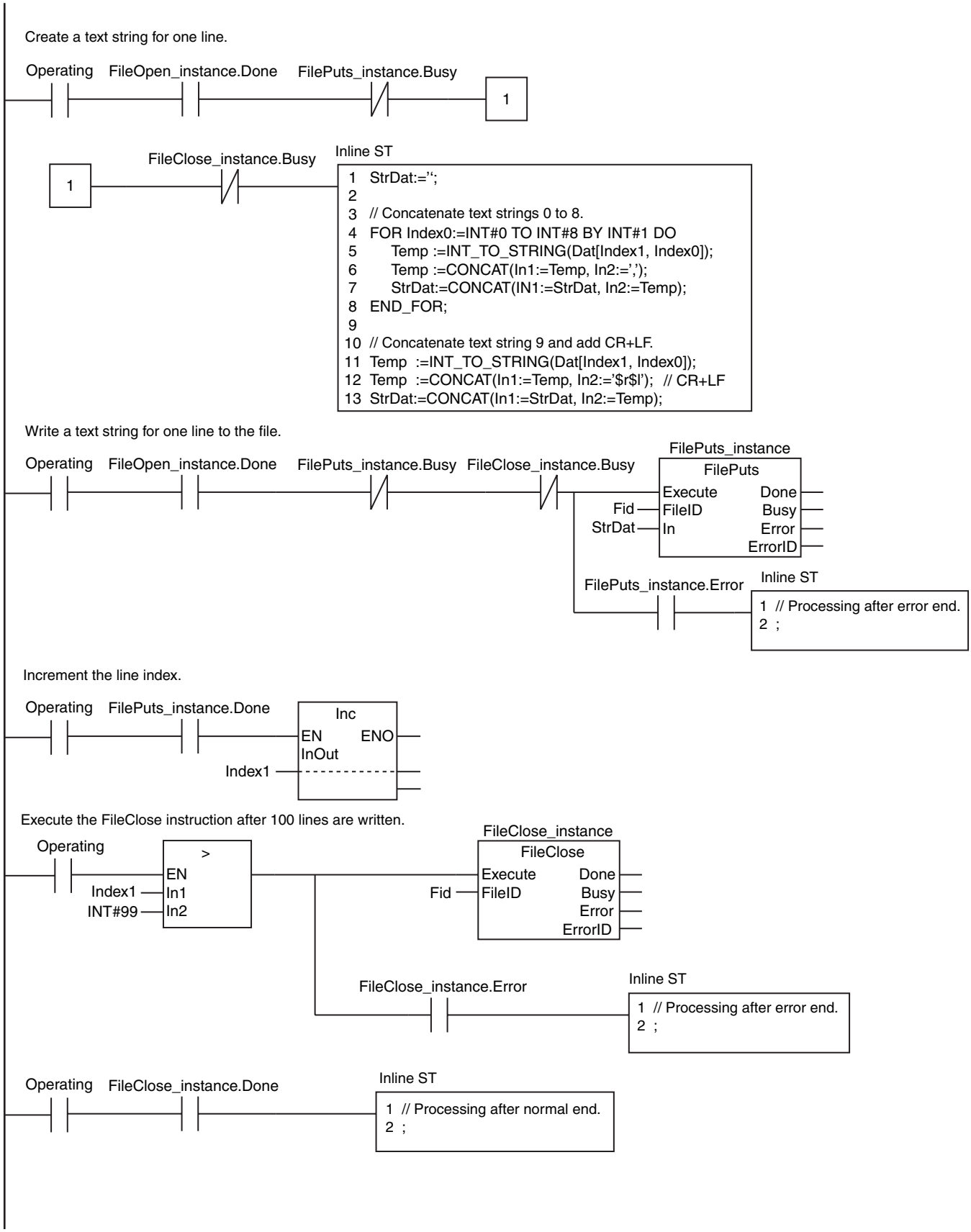


LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Index0	INT	0	Column index
	Index1	INT	0	Row index
	Fid	DWORD	16#0	File ID
	StrDat	STRING[255]	‘’	Text string data
	Dat	ARRAY[0..99,0..9] OF INT	[1000(0)]	Numeric data
	Temp	STRING[255]	”	Temporary data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag




Create a text string for one line.
Operating
FileOpen_instance.Done
FilePuts_instance.Busy
1
1
FileClose_instance.Busy
Inline ST
1 StrDat:='';
2
3 // Concatenate text strings 0 to 8.
4 FOR Index0:=INT#0 TO INT#8 BY INT#1 DO
5 Temp :=INT_TO_STRING(Dat[Index1, Index0]);
6 Temp :=CONCAT(In1:=Temp, In2:=',');
7 StrDat:=CONCAT(IN1:=StrDat, In2:=Temp);
8 END_FOR;
9
10 // Concatenate text string 9 and add CR+LF.
11 Temp :=INT_TO_STRING(Dat[Index1, Index0]);
12 Temp :=CONCAT(In1:=Temp, In2:='rl'); // CR+LF
13 StrDat:=CONCAT(In1:=StrDat, In2:=Temp);
Write a text string for one line to the file.
Operating
FileOpen_instance.Done
FilePuts_instance.Busy
FileClose_instance.Busy
FilePuts_instance
FilePuts
Execute
Done
Fid
FileID
Busy
StrDat
In
Error
ErrorID
FilePuts_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Increment the line index.
Operating
FilePuts_instance.Done
Inc
EN
ENO
InOut
Index1
Execute the FileClose instruction after 100 lines are written.
Operating
>
EN
Index1
In1
INT#99
In2
FileClose_instance
FileClose
Execute
Done
Fid
FileID
Busy
Error
ErrorID
FileClose_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Operating
FileClose_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	Stage	INT	0	Stage change
	Index0	INT	0	Column index
	Index1	INT	0	Row index
	Fid	DWORD	16#0	File ID
	StrDat	STRING[255]	‘’	Text string data
	Dat	ARRAY[0..99,0..9] OF INT	[1000(0)]	Numeric data
	Temp	STRING[255]	‘’	Temporary data
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FilePuts_instance(Execute:=FALSE);
    FileClose_instance(Execute:=FALSE);
    Stage          :=INT#1;
    Index1         :=INT#0;    // Initialize row index.
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                        // Open file.
        FileOpen_instance(
            Execute  :=TRUE,
            FileName:='ABC.csv',          // File name
            Mode     :=_RDWR_CREATE,      // Read file
            FileID   =>Fid);              // File ID

        IF (FIleOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;          // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;         // Error end
        END_IF;

    2 :                        // Create a text string for one line.
        StrDat:='';

        // Concatenate text strings 0 to 8.
        FOR Index0:=INT#0 TO INT#8 BY INT#1 DO
            Temp :=INT_TO_STRING(Dat[Index1, Index0]);
            Temp :=CONCAT(In1:=Temp, In2:=',');
            StrDat:=CONCAT(IN1:=StrDat, In2:=Temp);
        END_FOR;

        // Concatenate text string 9 and add CR+LF.
        Temp  :=INT_TO_STRING(Dat[Index1, Index0]);
        Temp  :=CONCAT(In1:=Temp, In2:='$r$l');
        StrDat:=CONCAT(In1:=StrDat, In2:=Temp);

        Stage:=INT#3;
```

```
    3 :                         // Write text string.
        FilePuts_instance(
            Execute:=TRUE,
            FileID    :=Fid,
            In        :=StrDat);

        IF (FilePuts_instance.Done=TRUE) THEN
            Index1:=Index1+INT#1;

            IF (Index1>INT#99) THEN    // If 100 lines were written...
                Stage:=INT#4;
            ELSE
                FilePuts_instance(Execute:=FALSE);
                Stage:=INT#2;
            END_IF;
        END_IF;

        IF (FilePuts_instance.Error=TRUE) THEN
            Stage:=INT#99;          // Error end
        END_IF;

    4 :                         // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID    :=Fid);                   // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;       // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;          // Error end
        END_IF;

    99 :                        // Processing after error end.
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FileCopy

The FileCopy instruction copies the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileCopy	Copy File	FB	FileCopy_instance FileCopy Execute Done SrcFileName Busy DstFileName Error OverWrite ErrorID	FileCopy_instance(Execute, SrcFileName, DstFileName, OverWrite, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcFile Name	Source file	Input	Name of file to copy	Depends on data type.	---	"
DstFile Name	Destination file		Name of destination file			
OverWrite	Overwrite enable		TRUE: Enable overwrite. FALSE: Prohibit overwrite.			FALSE

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcFile Name																				OK
DstFile Name																				OK
OverWrite	OK																			

Function

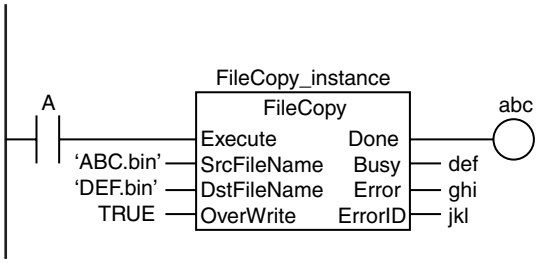
The FileCopy instruction copies the file specified by source file *SrcFileName* to designation file *DstFileName* in the SD Memory Card.

If a file with the name *DstFileName* already exists in the SD Memory Card, the following processing is performed depending on the value of overwrite enable *OverWrite*.

Value of *OverWrite*	Treatment
TRUE (Enable overwrite.)	The existing file is overwritten.
FALSE (Prohibit overwrite.)	The file is not overwritten and an error occurs.

The following figure shows a programming example. Here, the file 'DEF.bin' is overwritten with the file 'ABC.bin.'

LD



ST

```
FileCopy_instance(A, 'ABC.bin', 'DEF.bin',
                  TRUE, abc, def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the copy operation fails, the file specified by *DstFileName* may remain in an incomplete state in the SD Memory Card.

- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- If a file is open when the power supply it stopped with the power switch, the file is not corrupted.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- If a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The file specified by *SrcFileName* does not exist.
  - The value of *SrcFileName* is not a valid file name.
  - The value of *DstFileName* is not a valid file name.
  - The maximum number of files or directories is exceeded.
  - The file specified by *SrcFileName* or *DstFileName* is already being accessed.
  - A file with the name *DstFileName* already exits and the value of *OverWrite* is FALSE.
  - A file with the name *DstFileName* already exits and the file is write protected.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWrite-Var, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *DstFileName* exceeds the maximum number of bytes allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

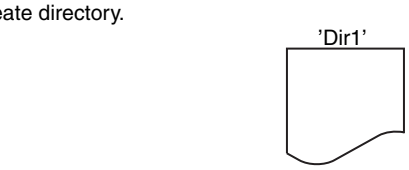
The following procedure is used to move a file.

1 The DirCreate instruction is used to create a directory called 'Dir1' in the SD Memory Card.

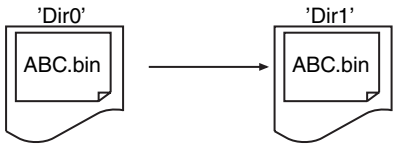
2 The FileCopy instruction is used to copy the file named 'ABC.bin' in the existing directory 'Dir0' to the directory 'Dir1.'

3 The DirRemove instruction is used to delete the directory ‘Dir0’ (the source of the copy).

1. Create directory.



2. Copy file.

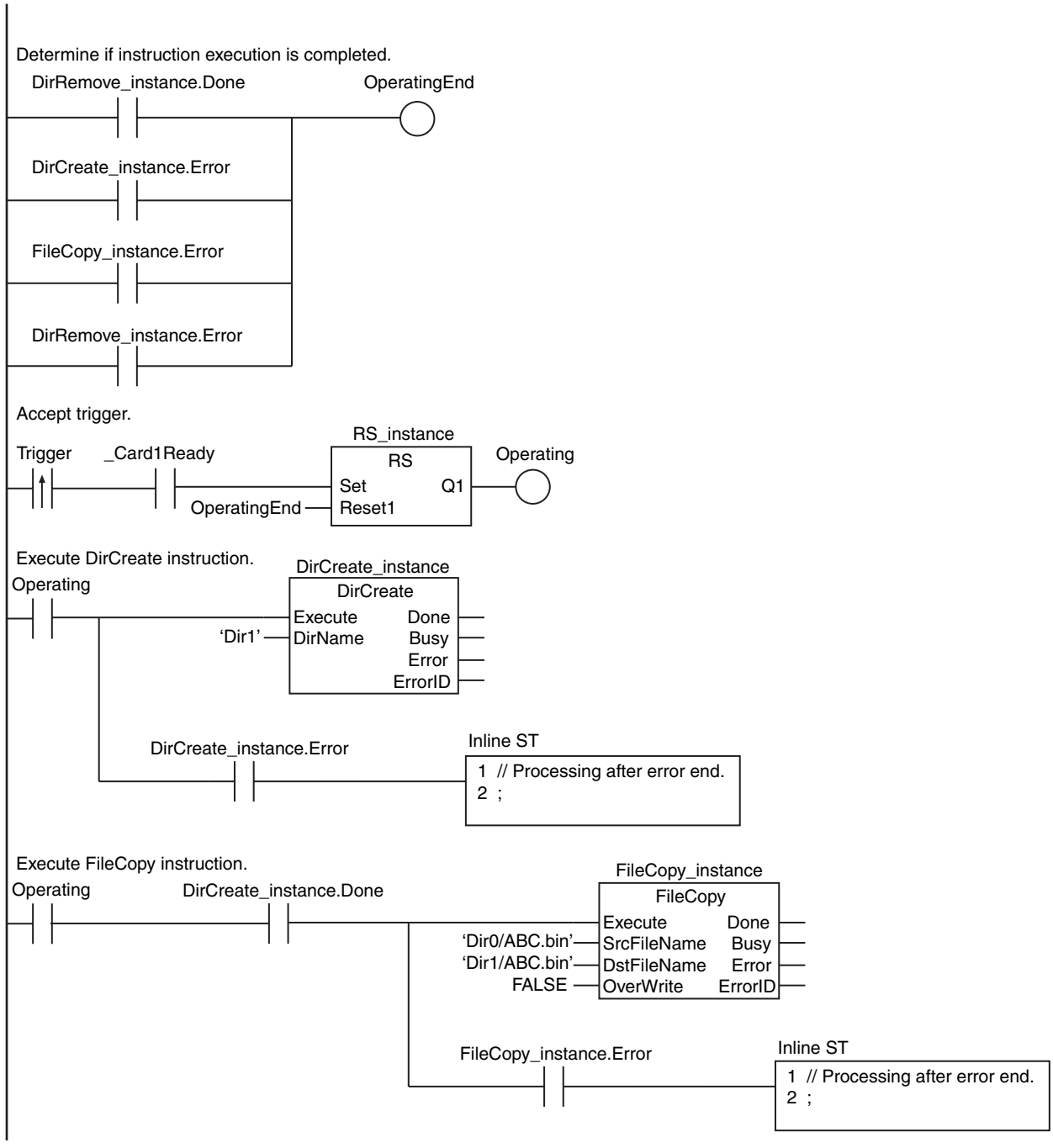


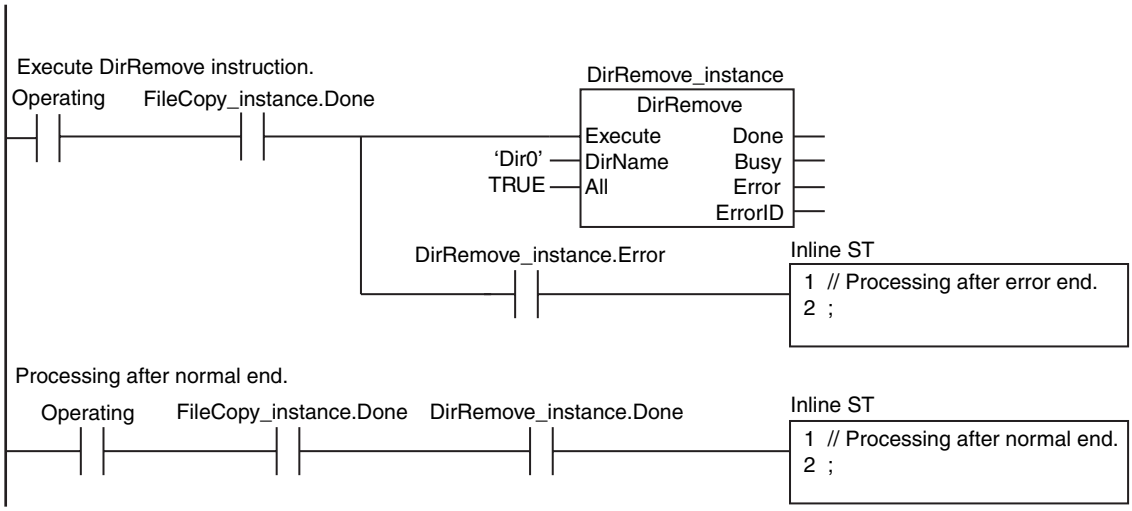
3. Delete directory.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	RS_instance	RS		
	DirCreate_instance	DirCreate		
	FileCopy_instance	FileCopy		
	DirRemove_instance	DirRemove		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag




Execute DirRemove instruction.
Operating
FileCopy_instance.Done
DirRemove_instance
DirRemove
Execute
Done
'Dir0'
DirName
Busy
TRUE
All
Error
ErrorID
DirRemove_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileCopy_instance.Done
DirRemove_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	Stage	INT	0	Stage change
	DirCreate_instance	DirCreate		
	FileCopy_instance	FileCopy		
	DirRemove_instance	DirRemove		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    DirCreate_instance(Execute:=FALSE);
    FileCopy_instance(Execute:=FALSE);
    DirRemove_instance(Execute:=FALSE);
    Stage          :=INT#1;
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                 // Create directory.
        DirCreate_instance(
            Execute :=TRUE,
            DirName:='Dir1');  // Directory name

        IF (DirCreate_instance.Done=TRUE) THEN
            Stage:=INT#2;              // Normal end
        END_IF;

        IF (DirCreate_instance.Error=TRUE) THEN
            Stage:=INT#99;             // Error end
        END_IF;

    2 :                 // Copy file.
        FileCopy_instance(
            Execute :=TRUE,
            SrcFileName:='Dir0/ABC.bin',     // Name of file to copy
            DstFileName:='Dir1/ABC.bin',     // Name of destination file
            OverWrite    :=FALSE);           // Prohibit overwrite.

        IF (FileCopy_instance.Done=TRUE) THEN
            Stage:=INT#3;
        END_IF;

        IF (FileCopy_instance.Error=TRUE) THEN
            Stage:=INT#99;
        END_IF;
```

```
    3 :                 // Delete directory.
        DirRemove_instance(
            Execute :=TRUE,
            DirName:='Dir0',        // Directory name
            All         :=TRUE);    // Delete files and subdirectories.

        IF (DirRemove_instance.Done=TRUE) THEN
            Operating:=FALSE;       // Normal end
        END_IF;

        IF (DirRemove_instance.Error=TRUE) THEN
            Stage:=INT#99;          // Error end
        END_IF;

    99 :                    // Processing after error end.
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FileRemove

The FileRemove instruction deletes the specified file from the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileRemove	Delete File	FB	FileRemove_instance FileRemove Execute Done FileName Busy Error ErrorID	FileRemove_instance(Execute, FileName, Done, Busy, Error, ErrorID);

Variables

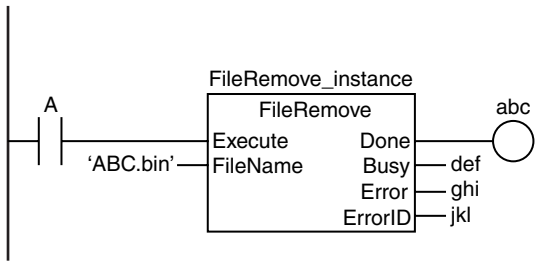
Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to delete	Depends on data type.	---	''

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK

Function

The FileRemove instruction deletes the file specified by file name *FileName* from the SD Memory Card.
The following figure shows a programming example. Here, the file named 'ABC.bin' is deleted.

LD



ST

```
FileRemove_instance(A, 'ABC.bin', abc,
                    def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.

Name	Meaning	Data type	Description
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- If a file is open when the power supply it stopped with the power switch, the file is not corrupted.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- If a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - The file specified by *FileName* does not exist.
  - The file specified by *FileName* is being accessed.
  - A file with the name *FileName* already exits and the file is write protected.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *FileName* exceeds the maximum number of characters allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.

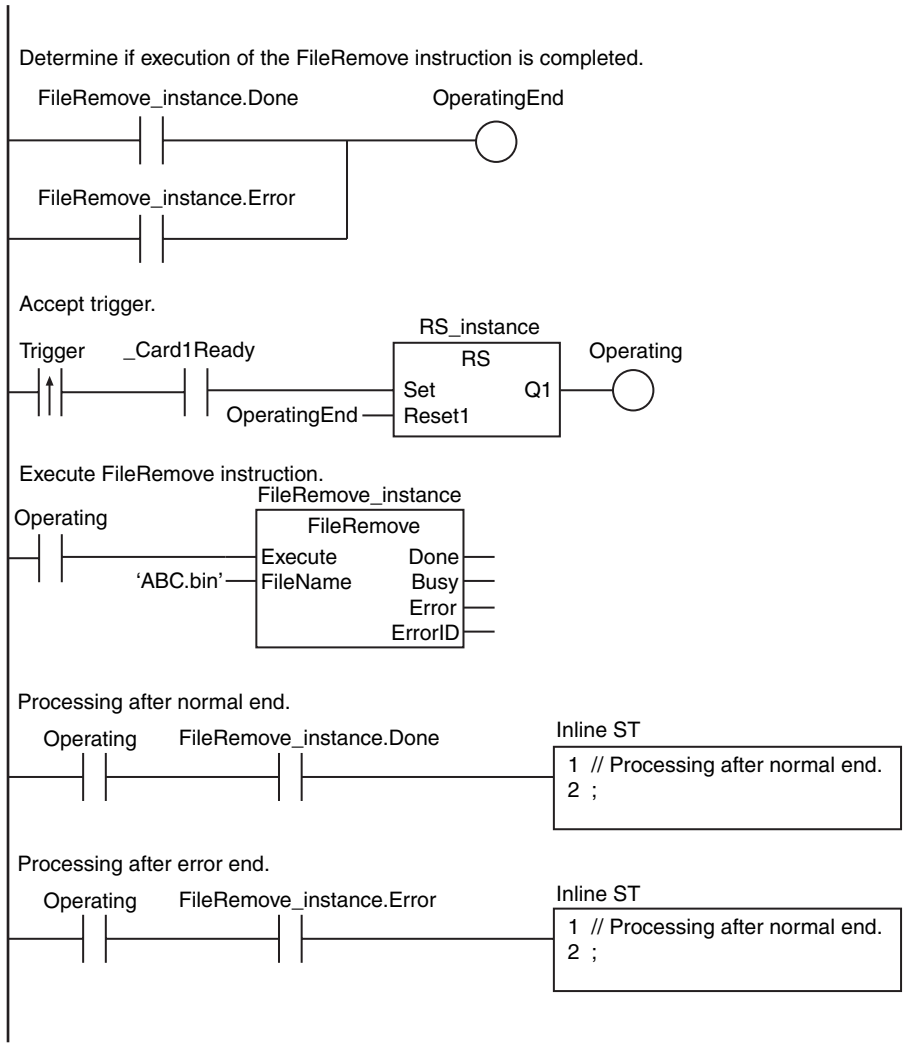
Sample Programming

In this sample, the file named 'ABC.bin' is deleted from the SD Memory Card.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	RS_instance	RS		
	FileRemove_instance	FileRemove		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	FileRemove_instance	FileRemove		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileRemove_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

// Execute FileRemove instruction.
IF (Operating=TRUE) THEN
    FileRemove_instance(
            Execute  :=TRUE,
            FileName:='ABC.bin');     // File name

    IF (FileRemove_instance.Done=TRUE) THEN
        Operating:=FALSE;             // Normal end
    END_IF;

    IF (FileRemove_instance.Error=TRUE) THEN
        Operating:=FALSE;             // Error end
    END_IF;
END_IF;
```

FileRename

The FileRename instruction changes the name of the specified file or directory in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileRename	Change File Name	FB	FileRename_instance FileRename Execute Done FileName Busy NewName Error OverWrite ErrorID	FileRename_instance(Execute, FileName, NewName, OverWrite, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	Original file name	Input	Original file name	Depends on data type.	---	"
NewName	New file name		New file name			
OverWrite	Overwrite enable		TRUE: Enable overwrite. FALSE: Prohibit overwrite.			FALSE

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
NewName																				OK
OverWrite	OK																			

Function

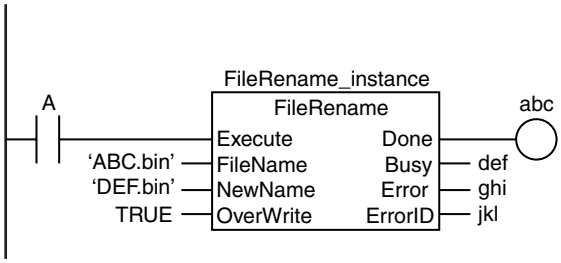
The FileRename instruction changes the name of the file or directory specified by original file name *FileName* to new file name *NewName* in the SD Memory Card.

If a file or directory with the name *NewName* already exists in the SD Memory Card, the following processing is performed depending on the value of overwrite enable *OverWrite*.

Value of *OverWrite*	Treatment
TRUE (Enable overwrite.)	The existing file or directory is overwritten.
FALSE (Prohibit overwrite.)	The file or directory is not overwritten and an error occurs.

The following figure shows a programming example. Here, the name of the file ‘ABC.bin’ is changed to ‘DEF.bin.’

LD



ST

```
FileRename_instance(A, 'ABC.bin', 'DEF.bin',
                    TRUE, abc, def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the directories are different for *FileName* and *NewName*, the file is moved to the directory that is specified with *NewName*.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- If a file is open when the power supply it stopped with the power switch, the file is not corrupted.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- If a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - The file directory specified with *FileName* does not exist.
  - The value of *FileName* or *NewName* is not a valid file name or directory name.
  - The file specified by *FileName* is being accessed.
  - There is a subdirectory in the directory that was specified for *FileName* and the value of *OverWrite* is TRUE.
  - A file with the name *NewName* already exits and the value of *OverWrite* is FALSE.
  - A file with the name *NewName* already exits, the file is write protected, and the value of *OverWrite* is TRUE.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWrite-Var, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *NewName* exceeds the maximum number of characters allowed in a file name or directory name.
  - An error that prevents access occurs during SD Memory Card access.

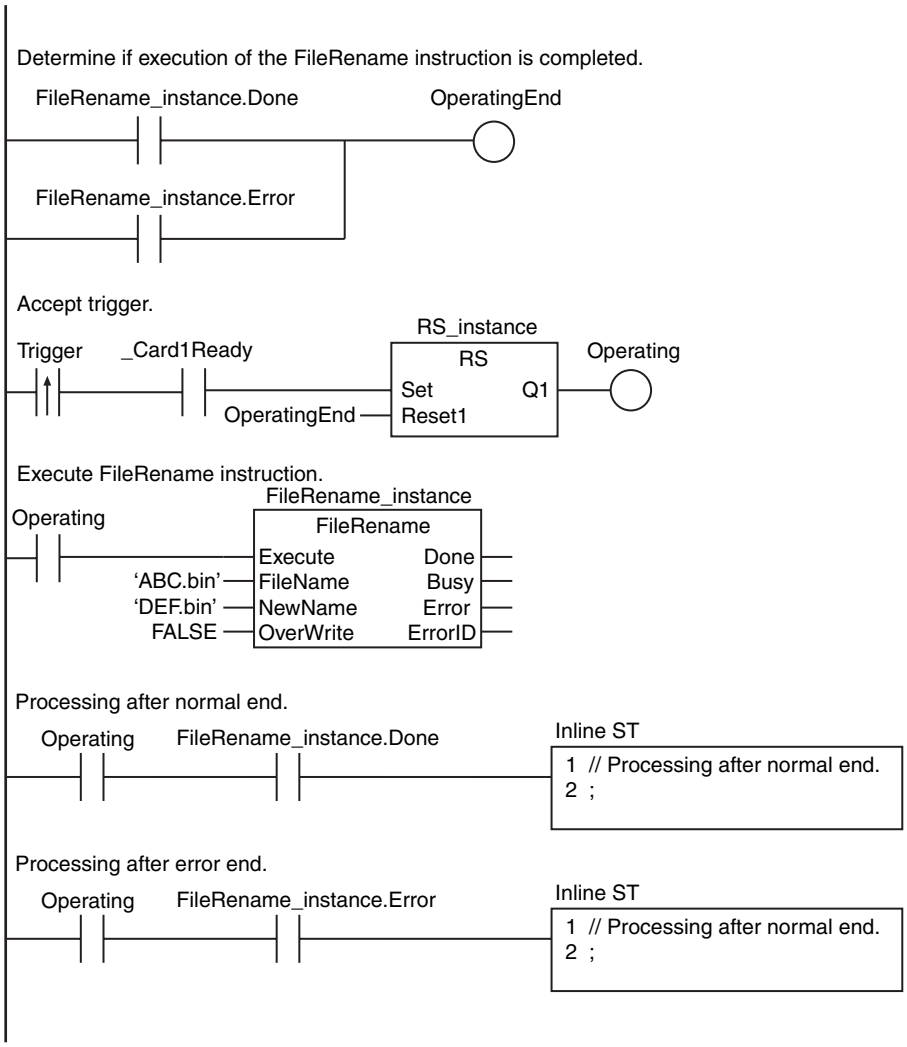
Sample Programming

In this sample, the name of the file 'ABC.bin' is changed to 'DEF.bin' on the SD Memory Card.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed.
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	RS_instance	RS		
	FileRename_instance	FileRename		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started.
	Operating	BOOL	False	Processing
	FileRename_instance	FileRename		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileRename_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

 // Execute FileRename instruction.
IF (Operating=TRUE) THEN
    FileRename_instance(
            Execute   :=TRUE,
            FileName :='ABC.bin',     // Original file name
            NewName:='DEF.bin',       // New file name
            OverWrite :=FALSE);       // Prohibit overwrite.

    IF (FileRename_instance.Done=TRUE) THEN
        Operating:=FALSE;             // Normal end
    END_IF;

    IF (FileRename_instance.Error=TRUE) THEN
        Operating:=FALSE;             // Error end
    END_IF;
END_IF;
```

DirCreate

The DirCreate instruction creates a directory with the specified name in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DirCreate	Create Directory	FB	DirCreate_instance DirCreate Execute Done DirName Busy Error ErrorID	DirCreate_instance(Execute, DirName, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DirName	Directory to create	Input	Name of directory to create	Depends on data type.	---	''

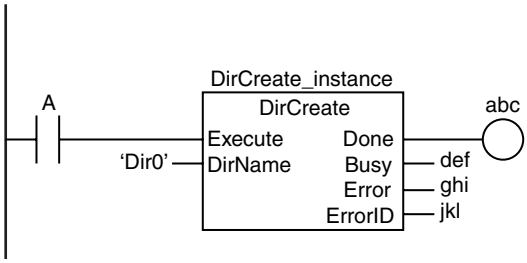
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				OK

Function

The DirCreate instruction creates a directory with the name specified by directory to create *Dir* in the SD Memory Card.

The following figure shows a programming example. Here, a directory named 'Dir0' is created.

LD



ST

```
DirCreate_instance(A, 'Dir0', abc,
                   def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- If a file is open when the power supply it stopped with the power switch, the file is not corrupted.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- If a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.

- There is insufficient space available on the SD Memory Card.
- The maximum number of directories is exceeded.
- The directory specified by *DirName* already exists.
- If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWrite-Var, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
- The value of *DirName* is not a valid directory name.
- The value of *DirName* exceeds the maximum number of characters allowed in a directory name.
- An error that prevents access occurs during SD Memory Card access.

Sample Programming

Refer to the sample programming that is provided for the FileCopy instruction (page 2-840).

DirRemove

The DirRemove instruction deletes the specified directory from the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DirRemove	Delete Directory	FB	DirRemove_instance DirRemove Execute Done DirName Busy All Error ErrorID	DirRemove_instance(Execute, DirName, All, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DirName	Directory to delete	Input	Directory to delete	Depends on data type.	---	"
All	All designation		Specifies whether to delete files and subdirectories inside specified directory TRUE: Delete files and subdirectories. FALSE: Do not delete.			FALSE

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				OK
All	OK																			

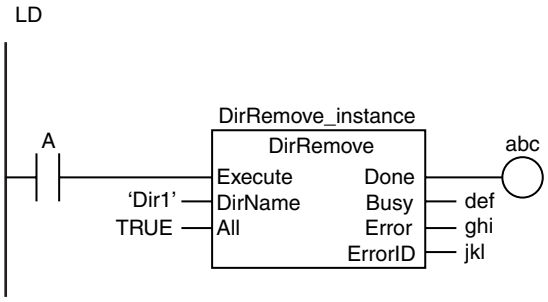
Function

The DirRemove instruction deletes the directory with the name specified by directory to delete *Dir* from the SD Memory Card.

If there are files or subdirectories in the specified directory, the following processing is performed according to the value of all designation *All*.

Value of *All*	Treatment
TRUE	All files and subdirectories are deleted along with the specified directory.
FALSE	The specified directory is not deleted and an error occurs.

The following figure shows a programming example. Here, a directory named 'Dir1' is deleted.



ST

```
DirRemove_instance(A, 'Dir1', TRUE, abc,
                   def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card is physically inserted and is mounted normally, i.e., if it can be accessed by instructions and communications commands. TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during SD Memory Card access. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-2 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- If a file is open when the power supply it stopped with the power switch, the file is not corrupted.
- If a file is open and the SD Memory Card is removed before the power switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- If a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- If the directory that is specified with *DirName* is write protected, an error occurs and the directory is not deleted. However, any files or directories that are not write-protected inside that directory are deleted.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - If the value of *All* is TRUE and the directory specified with *DirName* is being accessed by another instruction.
  - If the value of *All* is FALSE and the directory specified with *DirName* contains a file or directory.
  - The directory specified by *DirName* is write-protected.
  - The directory that is specified with *DirName* contains write-protected files or write-protected directories.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The directory specified by *DirName* does not exist.
  - The value of *DirName* exceeds the maximum number of characters allowed in a directory name.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

Refer to the sample programming that is provided for the FileCopy instruction (page 2-840).

Other Instructions

Instruction	Name	Page
ReadNbit_**	N-bit Read Group	2-864
WriteNbit_**	N-bit Write Group	2-866
ChkRange	Check Subrange Variable	2-868
GetMyTaskStatus	Read Current Task Status	2-870
Task_IsActive	Determine Task Status	2-873
Lock and Unlock	Lock Tasks/Unlock Tasks	2-875
Get**Clk	Get Clock Pulse Group	2-880
Get**Cnt	Get Incrementing Free-running Counter Group	2-881

ReadNbit_**

The ReadNbit_** instructions read zero or more bits from a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ReadNbit_**	N-bit Read Group	FUN	(@)ReadNbit_** EN ENO In Out Pos Size "**" must be a bit string data type.	Out:=ReadNbit_**(In, Pos, Size); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Read source	Input	Bit string to read	Depends on data type.	---	0
Pos	Read position		Bit position to read	0 to No. of bits in *In* –1		
Size	Read size		Number of bits to read	0 to No. of bits in *In*		1
Out	Read result	Output	Read result	Depends on data type.	---	---

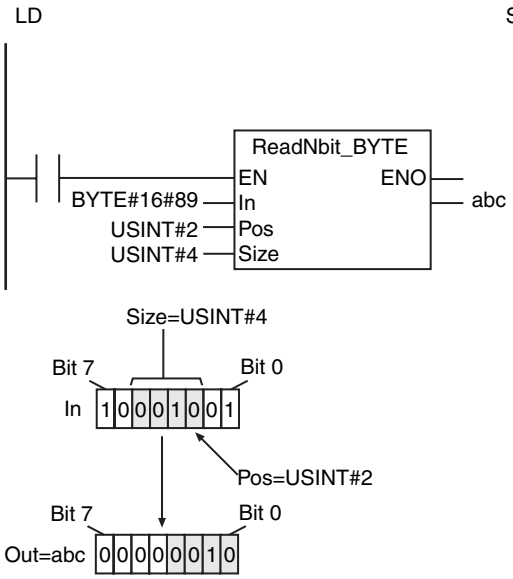
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Pos						OK														
Size						OK														
Out	Must be same data type as *In*																			

Function

A ReadNbit_** instruction reads the values of the upper *Size* bits from read position *Pos* in source bit string *In*. It assigns the values to read result *Out*.

The name of the instruction is determined by the data types of *In* and *Out*. For example, if *In* and *Out* are the WORD data type, the instruction is ReadNbit_WORD.

The following example shows the ReadNbit_BYTE instruction when *In* is BYTE#16#89, *Pos* is USINT#2 and *Size* is USINT#4.



Additional Information

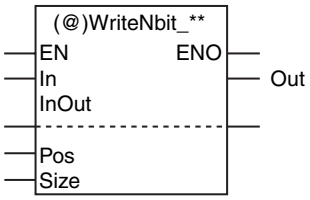
Use a WriteNbit_** instruction to write zero or more bits to a bit string.

Precautions for Correct Use

- The data types of *In* and *Out* must be the same.
- If the value of *Size* is 0, the value of *Out* is 16#0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The bit string in *In* does not have enough bits for the number of bits specified by *Size* from the position specified by *Pos*.

WriteNbit_**

The WriteNbit_** instructions write zero or more bits to a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
WriteNbit_**	N-bit Write Group	FUN	 "**" must be a bit string data type.	WriteNbit_**(In, Pos, Size, InOut); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Read source	Input	Bit string from which to read bits to write to *InOut*	Depends on data type.	---	0
Pos	Write position		Bit position to which to write	0 to No. of bits in *InOut* –1		
Size	Write size		Number of bits to write	0 to No. of bits in *In*		1
InOut	Write target	In-out	Write result	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

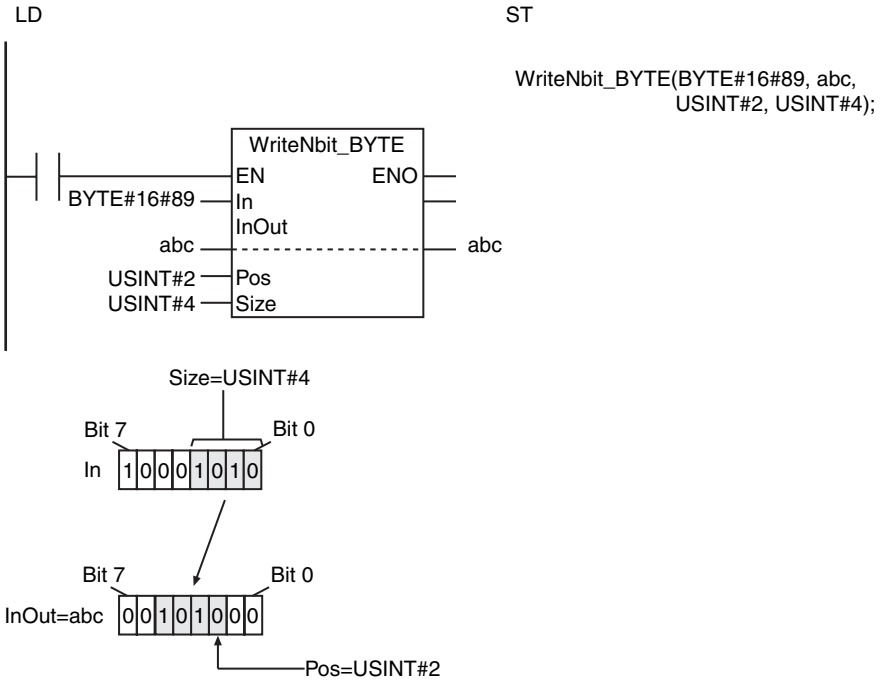
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Pos						OK														
Size						OK														
InOut	Must be same data type as *In*																			
Out	OK																			

Function

A WriteNbit_** instruction first reads the lower *Size* bits from read source *In*. Then it writes the values that it read to write position *Pos* in write target *InOut*.

The name of the instruction is determined by the data types of *In* and *Out*. For example, if *In* and *Out* are the WORD data type, the instruction is WriteNbit_WORD.

The following example shows the WriteNbit_BYTE instruction when *In* is BYTE#16#89, *Pos* is USINT#2 and *Size* is USINT#4.



Additional Information

Use a ReadNbit_** instruction to read zero or more bits from a bit string.

Precautions for Correct Use

- The data types of *In* and *InOut* must be the same.
- The value of *InOut* does not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The bit string in *InOut* does not have enough bits for the number of bits specified by *Size* from the position specified by *Pos*.

ChkRange

The ChkRange instruction determines if the value of a variable is within the valid range of the range type specification.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ChkRange	Check Subrange Variable	FUN		Out:=ChkRange(In, Val);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Variable to check	Input	Variable to check	Depends on data type.	---	*
Val	Range specification variable		Range specification variable	Depends on the range specification.		
Out	Check result	Output	Check result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Val	The basic data type that is the basis for the range specification must be the same as *In*.																			
Out	OK																			

Function

The ChkRange instruction determines if the value of variable to check *In* is within the valid range of the range specification variable *Val*. If the value is within the valid range, check result *Out* is TRUE. If the value is not within the valid range, check result *Out* is FALSE.

Additional Information

You can define the range type specification for integer variables (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, and LINT).

Precautions for Correct Use

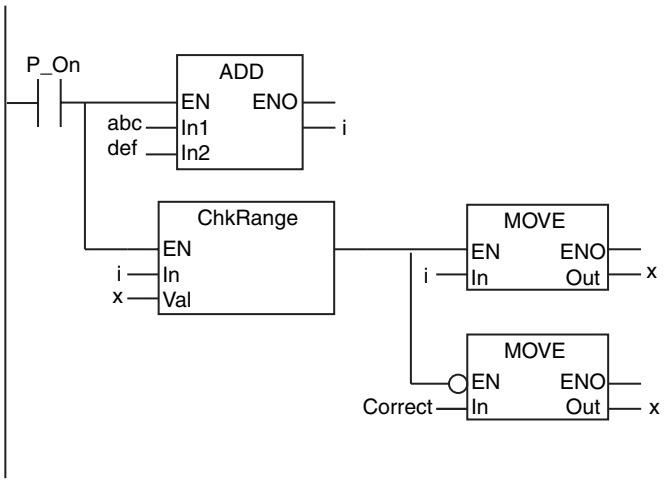
- If *In* is not a range specification variable, the value of *Out* changes to FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

Sample Programming

Here, the result of addition *i* is checked to see if it is within the valid range (10 to 99) of the range specification variable *x*. If it is not within the valid range, the value of variable *Correct* is assigned to variable *x*.

LD

Variable	Data type	Comment
i	INT	0
abc	INT	0
def	INT	0
x	INT(10..99)	10
Correct	INT	0



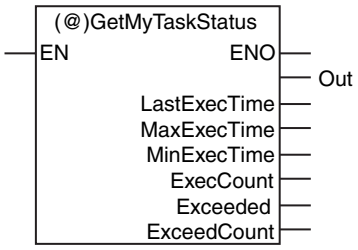
ST

Variable	Data type	Comment
i	INT	0
abc	INT	0
def	INT	0
Chk	BOOL	False
x	INT(10..99)	10
Correct	INT	0

```
i := abc+def;
Chk:=ChkRange(i, x);  // Check subrange variable.

IF (Chk=TRUE) THEN
    x := i;           // Assign i to x if value of i is in range.
ELSE
    x := Correct;     // Assign Correct to x if value of i is out of range.
END_IF;
```

GetMyTaskStatus

The GetMyTaskStatus reads the status of the current task.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetMyTaskStatus	Read Current Task Status	FUN		GetMyTaskStatus(LastExecTime, MaxExecTime, MinExecTime, ExecCount, Exceeded, ExceedCount);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Return value	Output	Always TRUE	TRUE only	---	---
LastExec Time	Last task execution time		Last task execution time of the current task	Depends on data type.*	ns	
MaxExec Time	Maximum task execution time		Maximum task execution time of the current task			
MinExec Time	Minimum task execution time		Minimum task execution time of the current task			
ExecCount	Task execution count		Number of task executions of the current task	Depends on data type.	---	
Exceeded	Task period exceeded flag		TRUE: The last execution of the current task was not completed within the task period. FALSE: The last execution of the current task was completed within the task period.			
Exceed-Count	Task period exceeded count		The number of times the current task has exceeded the task period.			

* Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out						OK														
LastExec Time																OK				
MaxExec Time																OK				
MinExec Time																OK				

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ExecCount								OK												
Exceeded	OK																			
Exceed-Count								OK												

Function

The GetMyTaskStatus reads the status of the current task. The task status includes the last task execution time *LastExecTime*, maximum task execution time *MaxExecTime*, minimum task execution time *MinExecTime*, task execution count *ExecCount*, task period exceeded flag *Exceeded*, and task period exceeded count *ExceedCount*.

Additional Information

MaxExecTime, *MinExecTime*, *ExecCount*, and *ExceedCount* are reset at the following times.

- When operation is started
- When a reset operation is executed from the Task Execution Time Monitoring Pane of the Sysmac Studio.

Precautions for Correct Use

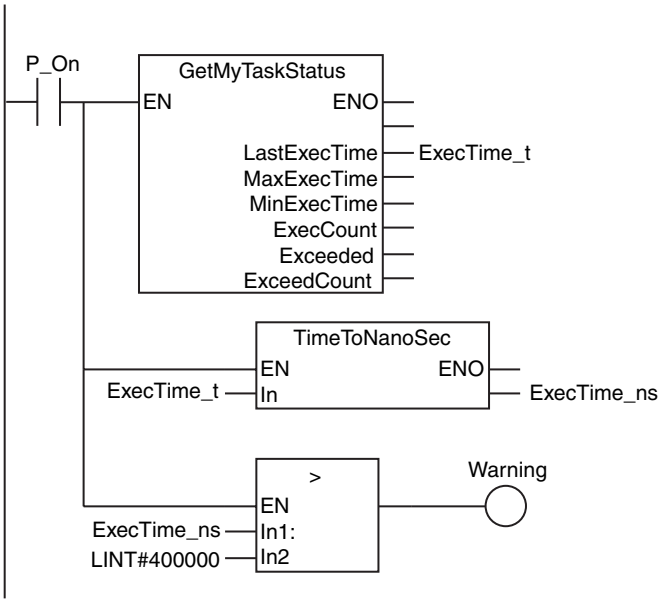
- When the value of *ExecCount* or *ExceedCount* exceeds the maximum value of UDINT data (4,294,967,295), it returns to 0.
- Return value *Out* is not used when the instruction is used in ST.

Sample Programming

In this sample, the GetMyTaskStatus reads the status of the current task. If the previous task execution time exceeds 400 μs (400000 ns), the value of the *Warning* variable changes to TRUE.

LD

Variable	Data type	Initial value	Comment
ExecTime_t	TIME	T#0s	Previous task execution time (TIME data)
ExecTime_ns	LINT	0	Previous task execution time (nanoseconds LINT data)
Warning	BOOL	False	Warning



ST

Variable	Data type	Initial value	Comment
ExecTime_t	TIME	T#0s	Previous task execution time (TIME data)
ExecTime_ns	LINT	0	Previous task execution time (nanoseconds LINT data)
Warning	BOOL	False	Warning

```
GetMyTaskStatus(LastExecTime=>ExecTime_t);  // Get previous task period.
ExecTime_ns:=TimeToNanoSec(ExecTime_t);     // Convert previous task period from TIME data to nanoseconds.
IF (ExecTime_ns>DINT#400000) THEN           // If previous task period exceeds 400,000 ns...
    Warning:=TRUE;                          //  Assign TRUE to Warning variable.
ELSE
    Warning:=FALSE;
END_IF;
```

Task_IsActive

The Task_IsActive instruction determines if the specified task is currently in execution.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Task_IsActive	Determine Task Status	FUN	(@)Task_IsActive EN TaskName Out	Out:=Task_IsActive(TaskName);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TaskName	Task name	Input	Task name	63 bytes max. (62 single-byte alphanumeric characters plus the final NULL character)	---	''
Out	Judgement	Output	TRUE: Task is in execution or on standby. FALSE: Not active	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TaskName																				OK
Out	OK																			

Function

The Task_IsActive instruction determines if the task specified with *TaskName* is currently in execution or on standby. "On standby" means that a high-priority task was started after this task was started, so processing has been interrupted.
If it is being executed or on standby, the value of judgment *Out* is TRUE. If it is not being executed, the value of *Out* is FALSE.

Precautions for Correct Use

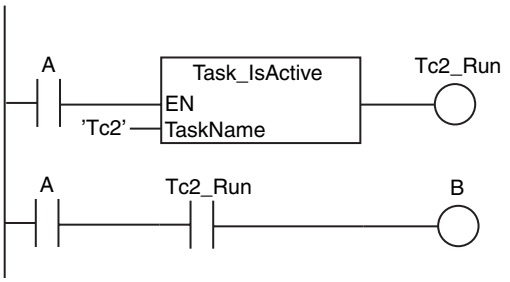
- You cannot use a variable to which a text string was assigned for *TaskName*. Directly specify a text string.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following case. The value of *Out* does not change.
  - The task specified with *TaskName* does not exist.

Sample Programming

In this sample, the instruction determines whether periodic task Tc2 is active when the value of variable *A* changes to TRUE. If it is active, the value of variable *B* changes to TRUE.

LD

Variable	Data type	Initial value	Comment
A	BOOL	False	
B	BOOL	False	
Tc2_Run	BOOL	False	Task Tc2 execution status



ST

Variable	Data type	Initial value	Comment
A	BOOL	False	
B	BOOL	False	
Tc2_Run	BOOL	False	Task Tc2 execution status

```
IF (A=TRUE) THEN
    // Determine task status.
    Tc2_Run:=Task_isActive('Tc2');
    // Make variable B TRUE if Tc2 is running.
    IF (Tc2_Run=TRUE) THEN
        B := TRUE;
    END_IF;

END_IF;
```

Lock and Unlock

Lock: Starts an exclusive lock between tasks. Execution of any other task with a lock region with the same lock number is disabled.

Unlock: Stops an exclusive lock between tasks.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Lock	Lock Tasks	FUN	(@)Lock EN ENO Index Out	Lock(Index);
Unlock	Unlock Tasks	FUN	(@)Unlock EN ENO Index Out	Unlock(Index);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Index	Lock number	Input	Lock number	Depends on data type.	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

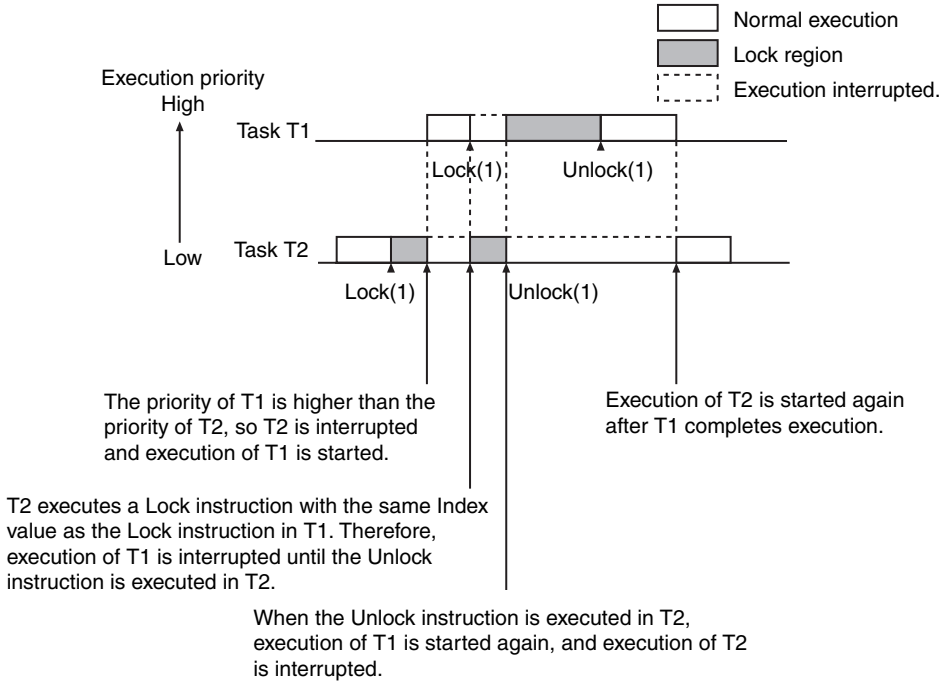
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Index						OK														
Out	OK																			

Function

The Lock and Unlock instructions create lock regions. If a lock region in one task is being executed, the lock regions with the same lock number in other tasks are not executed. Specify the lock number with *Index*.

The following figure shows a programming example.

Both task T1 and task T2 contain a lock region with *Index* set to 1. If the Lock instruction in T2 is executed first, the lock region in T1 is not executed until the Unlock instruction is executed in T2.



Lock regions with different values for *Index* do not affect each other.

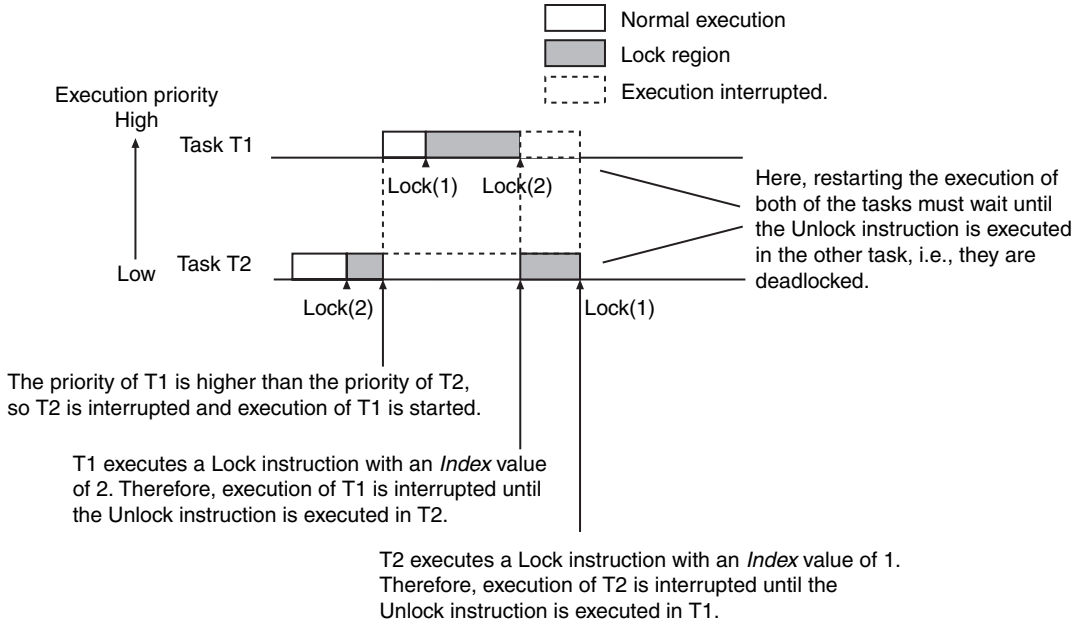
Additional Information

- The Lock and Unlock instructions are used when the same data is read/written from more than one task. They are used to prevent other tasks from reading/writing the data while a certain task is reading/writing the data.
- As long as the *Index* values are different, more than one pair of Lock and Unlock instructions can be placed in the same POU. The instruction pairs can also be nested.

Precautions for Correct Use

- Do not make lock regions any longer than necessary. If the lock region is too long, the task execution period may be exceeded.
- Always use the Lock and Unlock instructions together as a set in the same section of the same POU.
- You can set a maximum of 16,777,215 lock regions at the same time.

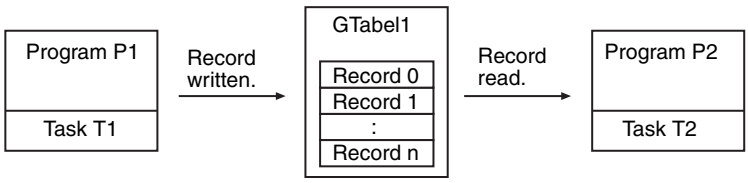
- If Lock instructions are used in more than one task, a deadlock may occur if they are positioned poorly. A Task Execution Timeout Error will occur if there is a deadlock and a total stop is performed.



- An error occurs in the following case. The value of *Out* does not change.
  - There are more than 16,777,215 lock region at the same time.

Sample Programming

Here, program P1 in task T1 and program P2 in task T2 both access the same global variable *GTable1*. When the value of write request *WriteReq* changes to TRUE, P1 writes one record to record array *GTable1.Record[]* and increments *GTable1.Index*. When read request *ReadReq* changes to TRUE, P2 decrements *GTable1.Index* and reads one record from *GTable1.Record[]*.
The Lock instruction is used so that reading and writing do not occur at the same time.



Definition of Global Variable *GTable*

Data type

Variable	Data type	Comment
USERTABLE	STRUCT	Record storage structure
Index	INT	Index
Record	ARRAY[0..99] OF LREAL	Record array

Global Variables

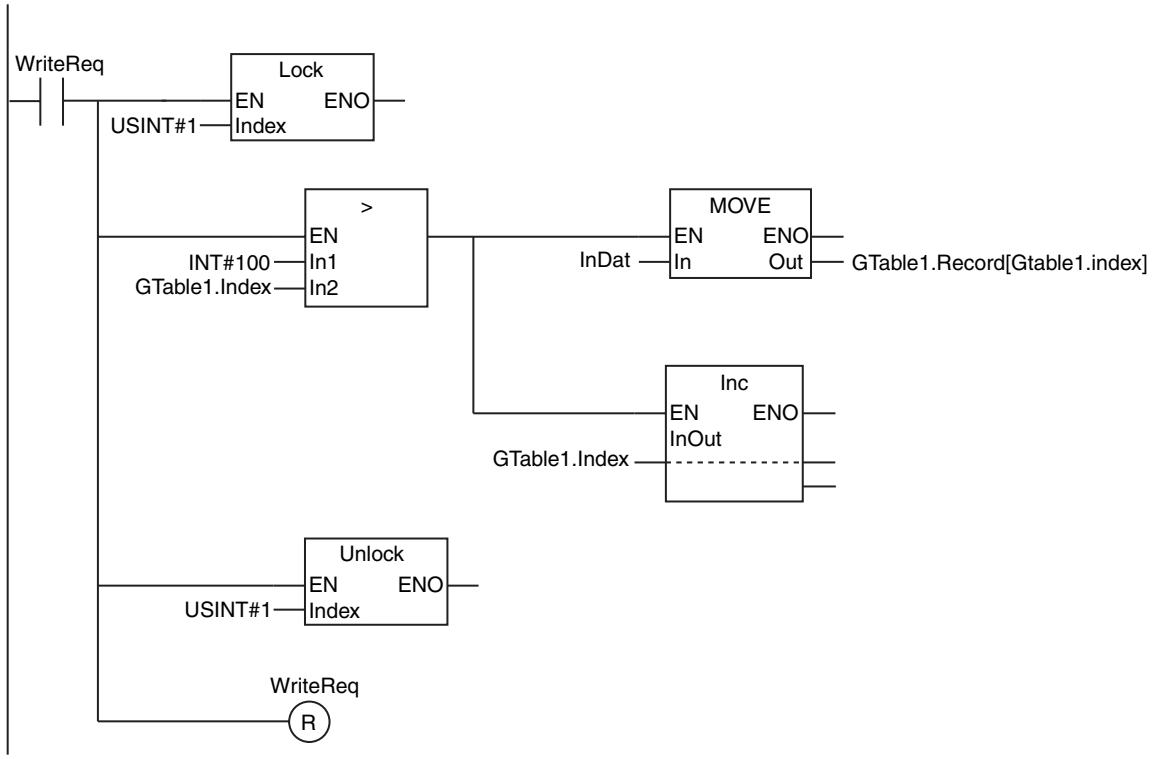
Variable	Data type	Initial value	Comment
GTable1	USERTABLE	(Index:=0,Record:=[100(0.0)])	Record storage structure

Program P1

LD

Internal Variables	Variable	Data type	Initial value	Comment
	WriteReq	BOOL	False	Write request
	InDat	LREAL	0.0	Write data

External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure



ST

Internal Variables	Variable	Data type	Initial value	Comment
	WriteReq	BOOL	False	Write request
	InDat	LREAL	0.0	Write data

External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure

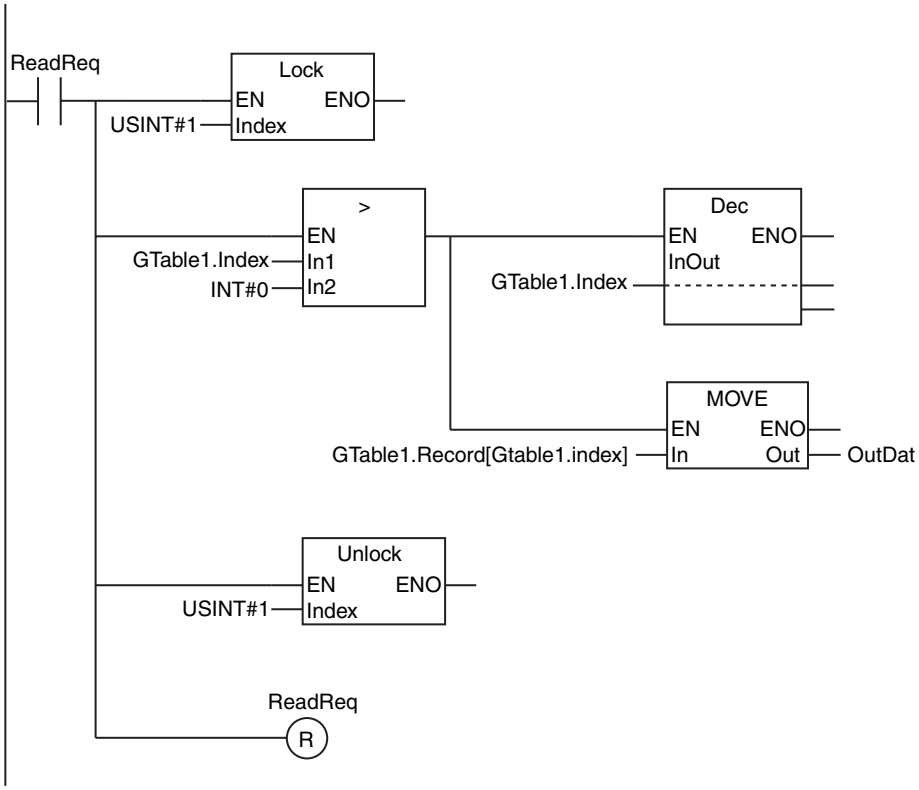
```
// Detect write request.
IF (WriteReq=TRUE) THEN

    // Execute Lock instruction.
    Lock(USINT#1);

    IF (INT#100>GTable1.Index) THEN
        GTable1.Record[GTable1.Index]:=InDat;
        GTable1.Index                 :=GTable1.Index+INT#1;
    END_IF;

    // Execute Unlock instruction.
    UnLock(USINT#1);
    WriteReq:=FALSE;

END_IF;
```

Program P2

LD

Internal Variables	Variable	Data type	Initial value	Comment
	ReadReq	BOOL	False	Read request
	OutDat	LREAL	0.0	Read data

External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure



ST

Internal Variables	Variable	Data type	Initial value	Comment
	ReadReq	BOOL	False	Read request
	OutDat	LREAL	0.0	Read data

External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure

```
// Detect read request.
IF (ReadReq=TRUE) THEN

    // Execute Lock instruction.
    Lock(USINT#1);

    IF (GTable1.Index>INT#0) THEN
        GTable1.Index:=GTable1.Index-INT#1;
        OutDat        :=GTable1.Record[GTable1.Index];
    END_IF;

    // Execute Unlock instruction.
    UnLock(USINT#1);
    ReadReq:=FALSE;

END_IF;
```

Get**Clk

The Get**Clk instruction outputs a clock pulse at the specified cycle.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Get**Clk	Get Clock Pulse Group	FUN	Get**Clk EN — Out "**" must be 100 us, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min.	Out:=Get**Clk(); "**" must be 100 us, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Clock pulse	Output	Clock pulse	Depends on data type.	---	---

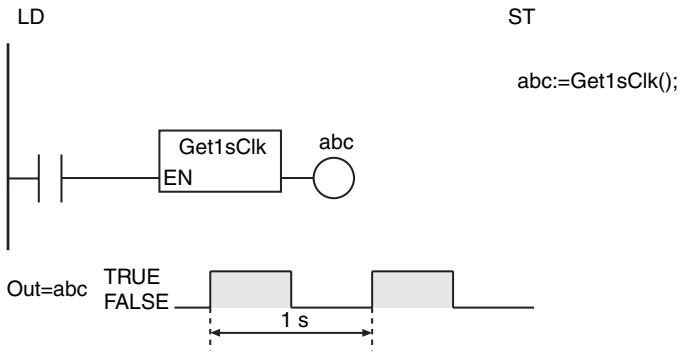
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			

Function

The Get**Clk instruction outputs a clock pulse at the specified cycle.
The clock pulse period is 100 us, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min.

The name of the instruction is determined by the period of the clock pulse. For example, if the period of the clock pulse is 10 ms, the instruction name is Get10msClk.

The following example is for the Get1sClk instruction.



Precautions for Correct Use

- When the instruction is executed, the first value of *Out* may be TRUE or it may be FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

Get**Cnt

The Get**Cnt instruction gets the values of free-running counters of the specified cycle.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Get**Cnt	Get Incrementing Free-running Counter Group	FUN	Get**Cnt EN ENO Out "**" must be 100 ns, 1us, 1 ms, 10 ms, 100 ms, or 1 s.	Out:=Get**Cnt(); "**" must be 100 ns, 1 us, 1 ms, 10 ms, 100 ms, or 1 s.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Count	Output	Value of free-running counter	Depends on data type.	---	---

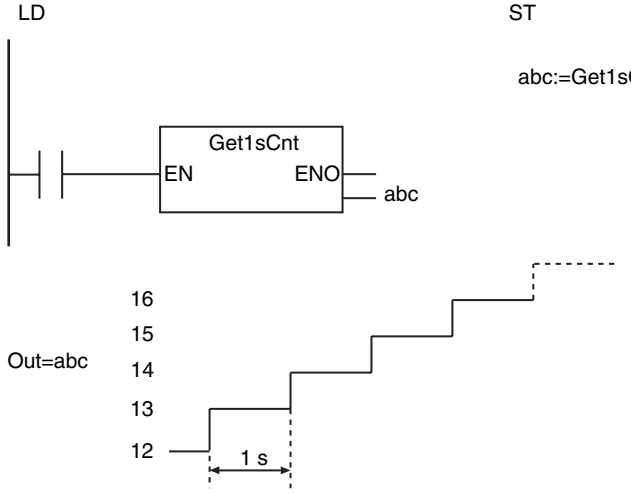
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out									OK											

Function

The Get**Cnt instruction gets the values of free-running counters of the specified cycle.
A free-running counter is a counter that is incremented at a specific period. *Out* is the current value of the count. The counter period is 100 ns, 1 us, 1 ms, 10 ms, 100 ms, or 1 s.

The name of the instruction is determined by counter period. For example, if the counter period is 10 ms, the instruction name is Get10msCnt.

The following example is for the Get1sCnt instruction.



Precautions for Correct Use

- Free-running counters start counting as soon as the power supply is turned ON. When the count exceeds the valid range of ULINT data (18,446,744,073,709,551,615), it returns to 0 and counting continues.
- This instruction only gets the current value of the free-running counter. It does not reset the counter to 0.
- The first value of *Out* cannot be predicted. It will not necessarily start from 0.

App

Appendices

A-1 Error Codes Related to Instructions A-2
A-2 Error Code Descriptions A-18
A-3 Error Code Details A-24
A-4 SDO Abort Codes A-47

A-1 Error Codes Related to Instructions

Error codes are assigned to the errors that can occur when instructions are executed. If an instruction has an *ErrorID* output variable, the value of the *ErrorID* gives you the error code. However, you cannot get the error codes for instructions that do not have an *ErrorID* output variable. The following table, however, gives all of the error codes that can occur for instruction execution. Use this table together with the information in *A-2 Error Code Descriptions* and *A-3 Error Code Details*.

Type	Instruction	Name	Error codes
Ladder Diagram Instructions	LD	Load	16#0406
	LDN	Load NOT	16#0406
	AND	AND	16#0406
	ANDN	AND NOT	16#0406
	OR	OR	16#0406
	ORN	OR NOT	16#0406
	Out	Output	16#0406
	OutNot	Output NOT	16#0406
ST Statement Instructions	IF	If	---
	CASE	Case	---
	WHILE	While	---
	REPEAT	Repeat	---
	RETURN	Return	---
	FOR	Repeat Start	---
	EXIT	Break Loop	---
Sequence Input Instructions	R_TRIG (Up)	Up Trigger	16#0406
	F_TRIG (Down)	Down Trigger	16#0406
	TestABit	Test A Bit	16#0405
	TestABitN	Test A Bit NOT	16#0405
Sequence Output Instructions	RS	Reset-Priority Keep	---
	SR	Set-Priority Keep	---
	Set	Set	---
	Reset	Reset	---
	SetBits	Set Bits	16#0405 16#0400 16#0406
	ResetBits	Reset Bits	16#0405 16#0400 16#0406
	SetABit	Set A Bit	16#0405
	ResetABit	Reset A Bit	16#0405
	OutABit	Output A Bit	16#0405
Sequence Control Instructions	End	End	---
	RETURN	Return	---
	MC	Master Control Start	---
	MCR	Master Control End	---
	JMP	Jump	---
	FOR	Repeat Start	---

Type	Instruction	Name	Error codes
Sequence Control Instructions	NEXT	Repeat End	---
	BREAK	Break Loop	---
Comparison Instructions	EQ (=)	Equal	---
	NE (<>)	Not Equal	---
	LT (<)	Less Than	---
	LE (<=)	Less Than Or Equal	---
	GT (>)	Greater Than	---
	GE (>=)	Greater Than Or Equal	---
	EQascii	Text String Comparison Equal	16#0410
	NEascii	Text String Comparison Not Equal	16#0410
	LTascii	Text String Comparison Less Than	16#0410
	LEascii	Text String Comparison Less Than or Equal	16#0410
	GTascii	Text String Comparison Greater Than	16#0410
	GEascii	Text String Comparison Greater Than or Equal	16#0410
	Cmp	Compare	---
	ZoneCmp	Zone Comparison	16#0401
	TableCmp	Table Comparison	16#0406
	AryCmpEQ	Array Comparison Equal	16#0400 16#0406
	AryCmpNE	Array Comparison Not Equal	16#0400 16#0406
	AryCmpLT	Array Comparison Less Than	16#0400 16#0406
	AryCmpLE	Array Comparison Less Than Or Equal	16#0400 16#0406
	AryCmpGT	Array Comparison Greater Than	16#0400 16#0406
	AryCmpGE	Array Comparison Greater Than Or Equal	16#0400 16#0406
	AryCmpEQV	Array Value Comparison Equal	16#0406
	AryCmpNEV	Array Value Comparison Not Equal	16#0406
	AryCmpLTV	Array Value Comparison Less Than	16#0406
	AryCmpLEV	Array Value Comparison Less Than Or Equal	16#0406
	AryCmpGTV	Array Value Comparison Greater Than	16#0406
	AryCmpGEV	Array Value Comparison Greater Than Or Equal	16#0406
Timer Instructions	TON	On-Delay Timer	---
	TOF	Off-Delay Timer	---
	TP	Timer Pulse	---
	AccumulationTimer	Accumulation Timer	---
	Timer	Hundred-ms Timer	---
Counter Instructions	CTD	Down-counter	---
	CTD_**	Down-counter Group	---
	CTU	Up-counter	---
	CTU_**	Up-counter Group	---

Type	Instruction	Name	Error codes
Counter Instructions	CTUD	Up-down Counter	---
	CTUD_**	Up-down Counter Group	---
Math Instructions	ADD (+)	Addition	16#0410
	AddOU (+OU)	Addition with Overflow/Underflow Check	---
	SUB (-)	Subtraction	---
	SubOU (-OU)	Subtraction with Overflow/Underflow Check	---
	MUL (*)	Multiplication	---
	MulOU (*OU)	Multiplication with Overflow/Underflow Check	---
	DIV (/)	Division	16#0400
	MOD	Modulo-division	16#0400
	ABS	Absolute Value	---
	RadToDeg	Radians to Degrees	---
	DegToRad	Degrees to Radians	---
	SIN	Sine in Radians	---
	COS	Cosine in Radians	---
	TAN	Tangent in Radians	---
	ASIN	Principal Arc Sine	---
	ACOS	Principal Arc Cosine	---
	ATAN	Principal Arc Tangent	---
	SQRT	Square Root	---
	LN	Natural Logarithm	---
	LOG	Logarithm Base 10	---
	EXP	Natural Exponential Operation	---
	EXPT (**)	Exponentiation	---
	Inc	Increment	---
	Dec	Decrement	---
	Rand	Random Number	---
	AryAdd	Array Addition	16#0400 16#1006
	AryAddV	Array Value Addition	16#0400 16#1006
	ArySub	Array Subtraction	16#0400 16#1006
	ArySubV	Array Value Subtraction	16#0400 16#1006
	AryMean	Array Mean	16#0406
	ArySD	Array Element Standard Deviation	16#0406
	ModReal	Real Number Modulo-division	---
	Fraction	Real Number Fraction	---
	CheckReal	Real Number Check	16#0402
BCD Conversion Instructions	**_BCD_TO_***	BCD-to-Unsigned Integer Conversion Group	16#0400 16#0403
	_TO_BCD_*	Unsigned Integer-to-BCD Conversion Group	16#0400

Type	Instruction	Name	Error codes
BCD Conversion Instructions	BCD_TO_**	BCD Data Type-to-Unsigned Integer Conversion Group	16#0400 16#0403
	BCDsToBin	Signed BCD-to-Signed Integer Conversion	16#0400 16#0403 16#0404
	BinToBCDs_**	Signed Integer-to-BCD Conversion Group	16#0400
	AryToBCD	Array BCD Conversion	16#0400 16#1006
	AryToBin	Array Unsigned Integer Conversion	16#0400 16#0403 16#0406
Data Type Conversion Instructions	**_TO_*** (Integer-to-Integer Conversion Group)	Integer-to-Integer Conversion Group	---
	_TO_* (Integer-to-Bit String Conversion Group)	Integer-to-Bit String Conversion Group	---
	_TO_* (Integer-to-Real Number Conversion Group)	Integer-to-Real Number Conversion Group	---
	_TO_* (Bit String-to-Integer Conversion Group)	Bit String-to-Integer Conversion Group	---
	_TO_* (Bit String-to-Bit String Conversion Group)	Bit String-to-Bit String Conversion Group	---
	_TO_* (Bit String-to-Real Number Conversion Group)	Bit String-to-Real Number Conversion Group	---
	_TO_* (Real Number-to-Integer Conversion Group)	Real Number-to-Integer Conversion Group	---
	_TO_* (Real Number-to-Bit String Conversion Group)	Real Number-to-Bit String Conversion Group	---
	_TO_* (Real Number-to-Real Number Conversion Group)	Real Number-to-Real Number Conversion Group	---
	**_TO_STRING (Integer-to-Text String Conversion Group)	Integer-to-Text String Conversion Group	---
	**_TO_STRING (Bit String-to-Text String Conversion Group)	Bit String-to-Text String Conversion Group	---
	**_TO_STRING (Real Number-to-Text String Conversion Group)	Real Number-to-Text String Conversion Group	---
	RealToFormatString	REAL-to-Formatted Text String	16#0400 16#0401
	LrealToFormatString	LREAL-to-Formatted Text String	16#0400 16#0401
	STRING_TO_** (Text String-to-Integer Conversion Group)	Text String-to-Integer Conversion Group	16#0407 16#0410
	STRING_TO_** (Text String-to-Bit String Conversion Group)	Text String-to-Bit String Conversion Group	16#0407 16#0410
	STRING_TO_** (Text String-to-Real Number Conversion Group)	Text String-to-Real Number Conversion Group	16#0400 16#0410
	TO_** (Integer Conversion Group)	Integer Conversion Group	16#0410

Type	Instruction	Name	Error codes
Data Type Conversion Instructions	TO_** (Bit String Conversion Group)	Bit String Conversion Group	16#0410
	TO_** (Real Number Conversion Group)	Real Number Conversion Group	16#0410
	TRUNC	Truncate	---
	Round	Round Off Real Number	---
	RoundUp	Round Up Real Number	---
Bit String Processing Instructions	AND (&)	Logical AND	---
	OR	Logical OR	---
	XOR	Logical Exclusive OR	---
	XORN	Logical Exclusive NOR	---
	NOT	Bit Reversal	---
	AryAnd	Array Logical AND	16#0400 16#1006
	AryOr	Array Logical OR	16#0400 16#1006
	AryXor	Array Logical Exclusive OR	16#0400 16#1006
	AryXorN	Array Logical Exclusive NOR	16#0400 16#1006
Selection Instructions	SEL	Binary Selection	16#0410
	MUX	Multiplexer	16#0400 16#0410
	LIMIT	Limiter	16#0401
	Band	Deadband Control	16#0401 16#0407
	Zone	Dead Zone Control	16#0401 16#0407
	MAX	Maximum	---
	MIN	Minimum	---
	AryMax	Array Maximum	16#0406
	AryMin	Array Minimum	16#0406
	ArySearch	Array Search	16#0406 16#0410 16#0419
Data Movement Instructions	MOVE	Move	16#0410
	MoveBit	Move Bit	16#0405
	MoveDigit	Move Digit	16#0406
	TransBits	Move Bits	16#0405 16#0406
	MemCopy	Memory Copy	16#0406
	SetBlock	Block Set	16#0406
	Exchange	Data Exchange	16#0407
	AryExchange	Array Data Exchange	16#0406 16#0407 16#0410
	AryMove	Array Move	16#0406

Type	Instruction	Name	Error codes
Data Movement Instructions	Clear	Initialize	---
	Copy**ToNum (Bit String to Signed Integer)	Bit Pattern Copy (Bit String to Signed Integer) Group	---
	Copy**To*** (Bit String to Real Number)	Bit Pattern Copy (Bit String to Real Number) Group	---
	CopyNumTo** (Signed Integer to Bit String)	Bit Pattern Copy (Signed Integer to Bit String) Group	---
	CopyNumTo** (Signed Integer to Real Number)	Bit Pattern Copy (Signed Integer to Real Number) Group	---
	Copy**To*** (Real Number to Bit String)	Bit Pattern Copy (Real Number to Bit String) Group	---
	Copy**ToNum (Real Number to Signed Integer)	Bit Pattern Copy (Real Number to Signed Integer) Group	---
Shift Instructions	AryShiftReg	Shift Register	16#0407
	AryShiftRegLR	Reversible Shift Register	16#0407
	ArySHL	Array N-element Left Shift	16#0407
	ArySHR	Array N-element Right Shift	16#0407
	SHL	N-bit Left Shift	---
	SHR	N-bit Right Shift	---
	NSHLC	Shift N-bits Left with Carry	16#0407
	NSHRC	Shift N-bits Right with Carry	16#0407
	ROL	Rotate N-bits Left	---
	ROR	Rotate N-bits Right	---
Conversion Instructions	Swap	Swap Bytes	---
	Neg	Reverse Sign	---
	Decoder	Bit Decoder	16#0406
	Encoder	Bit Encoder	16#0406
	BitCnt	Bit Counter	---
	ColmToLine_**	Column to Line Conversion Group	16#0405 16#0406
	LineToColm	Line to Column Conversion	16#0405 16#0406
	Gray	Gray Code Conversion	16#0400 16#0401
	PWLApprox	Broken Line Approximation	16#0401 16#0402 16#0406
	MovingAverage	Moving Average	16#0400 16#0406
	PIDAT	PID with Autotuning	16#0400 16#0401
	DispartReal	Separate Mantissa and Exponent	16#0402
	UniteReal	Combine Real Number Mantissa and Exponent	---
	NumToDecString	Fixed-length Decimal Text String Conversion	16#0400 16#0406
	NumToHexString	Fixed-length Hexadecimal Text String Conversion	16#0400 16#0406

Type	Instruction	Name	Error codes
Conversion Instructions	HexStringToNum_**	Hexadecimal Text String-to-Number Conversion Group	16#0410
	FixNumToString	Fixed-decimal Number-to-Text String Conversion	---
	StringToFixNum	Text String-to-Fixed-decimal Conversion	16#0407 16#0410
	DtToString	Date and Time-to-Text String Conversion	---
	DateToString	Date-to-Text String Conversion	---
	TodToString	Time of Day-to-Text String Conversion	---
	GrayToBin_**	Gray Code-to-Binary Code Conversion Group	---
	BinToGray_**	Binary Code-to-Gray Code Conversion	---
	StringToAry	Text String-to-Array Conversion	16#0407 16#0410
	AryToString	Array-to-Text String Conversion	16#0406
	DispartDigit	Four-bit Separation	16#0406
	UniteDigit_**	Four-bit Join Group	16#0406
	Dispart8Bit	Byte Data Separation	16#0406
	Unite8Bit_**	Byte Data Join Group	16#0406
	ToAryByte	Conversion to Byte Array	16#0400 16#0407
	AryByteTo	Conversion from Byte Array	16#0400 16#0406
	SizeOfAry	Get Number of Array Elements	---
Stack and Table Instructions	StackPush	Push onto Stack	16#0400 16#0401 16#0406 16#0407 16#0410
	StackFIFO	First In First Out	16#0400 16#0401 16#0406 16#0407 16#0410
	StackLIFO	Last In First Out	16#0400 16#0401 16#0406 16#0407 16#0410
	StackIns	Insert into Stack	16#0400 16#0401 16#0406 16#0407 16#0410
	StackDel	Delete from Stack	16#0401 16#0407

Type	Instruction	Name	Error codes
Stack and Table Instructions	RecSearch	Record Search	16#0400 16#0406 16#0410
	RecRangeSearch	Range Record Search	16#0400 16#0401 16#0406
	RecSort	Record Sort	16#0400 16#0406
	RecNum	Get Number of Records	16#0410 16#0406
	RecMax	Maximum Record Search	16#0406
	RecMin	Minimum Record Search	16#0406
FCS Instructions	StringSum	Checksum Calculation	16#0400 16#0410
	StringLRC	Calculate Text String LRC	16#0400 16#0410
	StringCRCCCITT	Calculate Text String CRC-CCITT	16#0400 16#0410
	StringCRC16	Calculate Text String CRC-16	16#0400 16#0410
	AryLRC_**	Calculate Array LRC Group	16#0406
	AryCRCCCITT	Calculate Array CRC-CCITT	16#0400 16#0406
	AryCRC16	Calculate Array CRC-16	16#0400 16#0406
Text String Instructions	CONCAT	Concatenate String	16#0410
	LEFT	Get String Left	16#0410
	RIGHT	Get String Right	16#0410
	MID	Get String Any	16#0406 16#0410
	FIND	Find String	16#0410
	LEN	String Length	16#0410
	REPLACE	Replace String	16#0406 16#0410
	DELETE	Delete String	16#0406 16#0410
	INSERT	Insert String	16#0406 16#0410
	GetByteLen	Get Byte Length	16#0410
	ClearString	Clear String	---
	ToUCase	Convert to Uppercase	16#0410
	ToLCase	Convert to Lowercase	16#0410
	TrimL	Trim String Left	16#0410
	TrimR	Trim String Right	16#0410
Time and Time of Day Instructions	ADD_TIME	Add Time	---
	ADD_TOD_TIME	Add Time to Time of Day	---
	ADD_DT_TIME	Add Time to Date and Time	---

Type	Instruction	Name	Error codes
Time and Time of Day Instructions	SUB_TIME	Subtract Time	---
	SUB_TOD_TIME	Subtract Time from Time of Day	---
	SUB_TOD_TOD	Subtract Time of Day	---
	SUB_DATE_DATE	Subtract Date	---
	SUB_DT_DT	Subtract Date and Time	---
	SUB_DT_TIME	Subtract Time from Date and Time	---
	MULTIME	Multiply Time	---
	DIVTIME	Divide Time	16#0400
	CONCAT_DATE_TOD	Concatenate Date and Time of Day	16#0407
	DT_TO_TOD	Extract Time of Day from Date and Time	---
	DT_TO_DATE	Extract Date from Date and Time	---
	SetTime	Set Time	16#0400
	GetTime	Get Time of Day	---
	DtToSec	Convert Date and Time to Seconds	---
	DateToSec	Convert Date to Seconds	---
	TodToSec	Convert Time of Day to Seconds	---
	SecToDt	Convert Seconds to Date and Time	16#0400
	SecToDate	Convert Seconds to Date	16#0400
	SecToTod	Convert Seconds to Time of Day	16#0400
	TimeToNanoSec	Convert Time to Nanoseconds	---
	TimeToSec	Convert Time to Seconds	---
	NanoSecToTime	Convert Nanoseconds to Time	16#0400
	SecToTime	Convert Seconds to Time	16#0400
	ChkLeapYear	Check for Leap Year	---
	GetDaysOfMonth	Get Days in Month	16#0400
	DaysToMonth	Convert Days to Month	16#0400
	GetDayOfWeek	Get Day of Week	---
	GetWeekOfYear	Get Week Number	---
	DtToDateStruct	Break Down Date and Time	---
	DateStructToDt	Join Time	16#0400 16#0407
System Control Instructions	TraceSamp	Data Trace Sampling	---
	TraceTrig	Data Trace Trigger	---
	GetTraceStatus	Read Data Trace Status	16#0400
	SetAlarm	Create User-defined Error	16#0400 16#040C
	ResetAlarm	Reset User-defined Error	16#0400
	GetAlarm	Get User-defined Error Status	---
	ResetPLCError	Reset PLC Controller Error	---
	GetPLCError	Get PLC Controller Error Status	---
	ResetCJBError	Reset CJ Bus Controller Error	16#0400 16#040D
	GetCJBError	Get I/O Bus Error Status	---
	GetEIPError	Get EtherNet/IP Error Status	---
	ResetMCError	Reset Motion Control Error	---
	GetMCError	Get Motion Control Error Status	---

Type	Instruction	Name	Error codes
System Control Instructions	ResetECError	Reset EtherCAT Error	16#041A
	GetECError	Get EtherCAT Error Status	---
	SetInfo	Create User-defined Information	16#0400
	ResetUnit	Restart Unit	16#0400 16#040D 16#040F
	GetNTPStatus	Read NTP Status	---
Communications Instructions	ExecPMCR	Protocol Macro	16#0400 16#0406 16#0407 16#040D 16#0413 16#0C00 16#0800 16#0801
	SerialSend	SCU Send Serial	16#0400 16#0406 16#040D 16#0C00 16#0800 16#0801
	SerialRcv	SCU Receive Serial	16#0400 16#0407 16#040D 16#0C00 16#0800 16#0801
	SendCmd	Send Command	16#0400 16#0406 16#0407 16#0800 16#0801
	CIPOpen	Open CIP Class 3 Connection	16#0400 16#1C00 16#1C01 16#1C03 16#1C04 16#2000 16#2003 16#2004
	CIPRead	Read Variable Class 3 Explicit	16#0400 16#0407 16#1C00 16#1C02 16#1C03 16#1C04

Type	Instruction	Name	Error codes
Communications Instructions	CIPWrite	Write Variable Class 3 Explicit	16#0400 16#0407 16#1C00 16#1C02 16#1C03 16#1C04
	CIPSend	Send Explicit Message Class 3	16#0407 16#1C00 16#1C02 16#1C03 16#1C04
	CIPClose	Close CIP Class 3 Connection	16#1C02 16#1C03
	CIPUCMMRead	Read Variable UCMM Explicit	16#0400 16#0407 16#2000 16#2004 16#1C00 16#1C01 16#1C03 16#1C04
	CIPUCMMWrite	Write Variable UCMM Explicit	16#0400 16#0407 16#2000 16#2004 16#1C00 16#1C01 16#1C03 16#1C04
	CIPUCMMSend	Send Explicit Message UCMM	16#0400 16#0407 16#2000 16#2004 16#1C00 16#1C01 16#1C03 16#1C04
	EC_CoESDOWrite	Write EtherCAT CoE SDO	16#0400 16#1800 16#1801 16#1802 16#1804 16#1808

Type	Instruction	Name	Error codes
Communications Instructions	EC_CoESDORead	Read EtherCAT CoE SDO	16#0400 16#1800 16#1801 16#1802 16#1803 16#1804 16#1808
	EC_StartMon	Start EtherCAT Packet Monitor	16#1805 16#1807 16#1808
	EC_StopMon	Stop EtherCAT Packet Monitor	16#1806 16#1808
	EC_SaveMon	Save EtherCAT Packets	16#1805 16#1807 16#1808
	EC_CopyMon	Transfer EtherCAT Packets	16#0400 16#1400 16#1401 16#1402 16#1403 16#1404 16#1405 16#140A 16#140B 16#140D 16#140E 16#1808
	EC_DisconnectSlave	Disconnect EtherCAT Slave	16#1801 16#1808
	EC_ConnectSlave	Connect EtherCAT Slave	16#1801 16#1808
	SktUDPCreate	Create UDP Socket	16#0400 16#2000 16#2001 16#2002 16#2003 16#2004 16#2008
	SktUDPRcv	UDP Socket Receive	16#0400 16#0407 16#2003 16#2006 16#2007 16#2008

Type	Instruction	Name	Error codes
Communications Instructions	SktUDPSend	UDP Socket Send	16#0400 16#0407 16#2002 16#2003 16#2007 16#2008
	SktTCPAccept	Accept TCP Socket	16#0400 16#2000 16#2002 16#2003 16#2004 16#2006 16#2008
	SktTCPConnect	Connect TCP Socket	16#0400 16#2000 16#2002 16#2003 16#2008
	SktTCPRcv	TCP Socket Receive	16#0400 16#0407 16#2003 16#2006 16#2007 16#2008
	SktTCPSend	TCP Socket Send	16#0400 16#0407 16#2003 16#2007 16#2008
	SktGetTCPStatus	Read TCP Socket Status	16#2007 16#2008
	SktClose	Close TCP/UDP Socket	16#2007 16#2008
	SktClearBuf	Clear TCP/UDP Socket Receive Buffer	16#2007 16#2008
SD Memory Card Instructions	FileWriteVar	Write Variable to File	16#0400 16#1400 16#1401 16#1402 16#1403 16#1404 16#1405 16#1409 16#140A 16#140B 16#140D 16#140E

Type	Instruction	Name	Error codes
SD Memory Card Instructions	FileReadVar	Read Variable from File	16#0400 16#1400 16#1403 16#1405 16#140B 16#140D 16#140E
	FileOpen	Open File	16#0400 16#1400 16#1401 16#1403 16#1404 16#1405 16#140A 16#140B 16#140D 16#140E
	FileClose	Close File	16#1403 16#1405 16#140E
	FileSeek	Seek File	16#0400 16#1400 16#1403 16#1405 16#1407 16#140E
	FileRead	Read File	16#0406 16#1400 16#1403 16#1405 16#1406 16#140E
	FileWrite	Write File	16#0406 16#1400 16#1401 16#1402 16#1403 16#1405 16#1406 16#140E
	FileGets	Get Text String	16#1400 16#1403 16#1405 16#1406 16#140E

Type	Instruction	Name	Error codes
SD Memory Card Instructions	FilePuts	Put Text String	16#1400 16#1401 16#1402 16#1403 16#1405 16#1406 16#140E
	FileCopy	Copy File	16#0400 16#1400 16#1401 16#1402 16#1403 16#1404 16#1405 16#1409 16#140A 16#140B 16#140D 16#140E
	FileRemove	Delete File	16#0400 16#1400 16#1401 16#1403 16#1405 16#140A 16#140B 16#140D 16#140E
	FileRename	Change File Name	16#0400 16#1400 16#1401 16#1403 16#1405 16#1408 16#1409 16#140A 16#140B 16#140D 16#140E
	DirCreate	Create Directory	16#0400 16#1400 16#1401 16#1402 16#1404 16#1409 16#140B 16#140C 16#140D 16#140E

Type	Instruction	Name	Error codes
SD Memory Card Instructions	DirRemove	Delete Directory	16#0400 16#1400 16#1401 16#1405 16#1408 16#140A 16#140B 16#140C 16#140D 16#140E
Other Instructions	ReadNbit_**	N-bit Read Group	16#1405 16#1406
	WriteNbit_**	N-bit Write Group	16#1405 16#1406
	ChkRange	Check Subrange Variable	---
	GetMyTaskStatus	Read Current Task Status	---
	Task_IsActive	Determine Task Status	16#1032
	Lock	Lock Tasks	16#0400
	Unlock	Unlock Tasks	---
	Get**Clk	Get Clock Pulse Group	---
	Get**Cnt	Get Incrementing Free-running Counter Group	---

A-2 Error Code Descriptions

The following table gives the error name, meaning, and assumed cause for each error code. Refer to *A-3 Error Code Details* for details.

Error code	Name	Meaning	Assumed cause	Reference
16#0400	Input Value Out of Range	An input parameter for an instruction exceeded the valid range for an input variable. Or, division by an integer of 0 occurred in division or remainder calculations.	• An input parameter for an instruction exceeded the valid range for an input variable. Or, division by an integer of 0 occurred in division or remainder calculations.	page A-25
16#0401	Input Mismatch	The relationship for the instruction input parameters did not meet required conditions. Or, a numeric value during or after instruction execution did not meet conditions.	• The relationship for an input parameter did not meet required conditions. • A value when processing an instruction or in the result does not meet the conditions.	page A-25
16#0402	Floating-point Error	Non-numeric data was input for a floating-point number input parameter to an instruction.	• Non-numeric data was input for a floating-point number input parameter to an instruction.	page A-25
16#0403	BCD Error	A value that was not BCD was input for a BCD input parameter to an instruction.	• A hexadecimal digit of A, B, C, D, E, or F was input for a BCD input parameter to an instruction.	page A-26

16#0404	Signed BCD Error	An illegal value was input for the most significant digit for a signed BCD input parameter to an instruction.	• An illegal value was input for the most significant digit for a signed BCD input parameter to an instruction.
			  • The most-significant digit was 2 to F when *_BCD0* was specified as the BCD format.
			  • The most-significant digit was A, B, C, D, or E when *_BCD2* was specified as the BCD format.
			  • The most-significant digit was B, C, D, or E when *_BCD3* was specified as the BCD format.
				page A-26

16#0405	Illegal Bit Position Specified	The bit position specified for an instruction was illegal.	• The bit position specified for an instruction exceeds the data range.	page A-26
16#0406	Illegal Data Position Specified	The data position specified for an instruction exceeded the data area range.	• The data position or data size specified for an instruction exceeded the data area range.	page A-27
16#0407	Data Range Exceeded	The results of instruction processing exceeded the data area range of the output parameter.	• The results of instruction processing, such as the number of array elements, exceeded the data area range of the output parameter.	page A-27
16#0409	No Errors to Clear	An instruction to clear a Controller error was executed when there was no error in the Controller.	• An instruction to clear a Controller error was executed when there was no error in the Controller.	page A-27
16#040B	No User Errors to Clear	An instruction to clear user-defined errors was executed when there was no user-defined error.	• An instruction to clear user-defined errors was executed when there was no user-defined error.	page A-28
16#040C	Limit Exceeded for User-defined Errors	An attempt was made to use the Create User-defined Error instruction to create more than the maximum number of user-defined errors.	• An attempt was made to use the Create User-defined Error instruction to create more than the maximum number of user-defined errors.	page A-28

Error code	Name	Meaning	Assumed cause	Reference
16#040D	Illegal Unit Specified	The Unit specified for an instruction does not exist.	• A Unit that does not exist in the Unit configuration information was specified. • A Unit that is in the Unit configuration information was specified, but the Units does not actually exist in the Controller.	page A-28
16#040F	Unit Restart Failed	Restarting a Special I/O Unit or CPU Bus Unit failed.	• The Special I/O Unit or CPU Bus Unit is processing data.	page A-29
16#0410	Text String Format Error	The text string input to an instruction is not correct.	• The text string that is input to the instruction for conversion to a number does not represent a number or it does not represent a positive number. • The input text string does not end in NULL.	page A-29
16#0411	Illegal Program Specified	The program specified for an instruction does not exist.	• The program specified by the function does not exist (e.g., it was deleted).	page A-29
16#0413	Undefined CJ-series Memory Address	The required specification is missing for a variable for which CJ-series Unit memory must be specified.	• The required AT specification is missing for a variable for which CJ-series Unit memory must be specified.	page A-30
16#0414	Stack Underflow	There is no data in a stack.	• An attempt was made to read data from a stack that contains no data.	page A-30
16#0416	Illegal Number of Array Elements or Dimensions	The valid range was exceeded for the number of array elements or dimensions in an array I/O parameter for an instruction.	• The valid range was exceeded for the number of array elements or dimensions in an array I/O parameter for an instruction.	page A-30
16#0417	Specified Task Does Not Exist	The task specified for the instruction does not exist.	• The specified task does not exist.	page A-30
16#0418	Unallowed Task Specification	An unallowed task was specified for an instruction.	• The local task, the primary periodic task, or a periodic task was specified.	page A-31
16#0419	Incorrect Data Type	A data type that cannot be used for an instruction is specified for an input or in-out variable.	• A data type that cannot be used for an instruction is specified for an input or in-out variable.	page A-31
16#041A	Multi-execution of Instructions	Multi-execution was specified for an instruction that does not support it.	• Execution of an instruction that does not support multi-execution of instructions was specified more than once.	page A-31
16#0800	FINS Error	An error occurred when a FINS command was sent or received.	• An error occurred when a FINS command was sent or received.	page A-32
16#0801	FINS Port Already in Use	The FINS port is being used.	• The FINS port is being used.	page A-32
16#0C00	Illegal Serial Communications Mode	The Serial Communications Unit is not in the serial communications mode required to execute an instruction.	• The serial communications port for the Serial Communications Unit is not set to the mode expected by the instruction.	page A-32
16#0C02	Port Setup Already Busy	A Change Port Setup instruction was executed during execution of another Change Port Setup instruction.	• A Change Port Setup instruction was executed during execution of another Change Port Setup instruction.	page A-33
16#1400	SD Memory Card Access Failure	SD Memory Card access failed when an instruction was executed.	• An SD Memory Card is either not inserted or is not inserted properly. • The SD Memory Card is broken. • The SD Memory Card slot is broken.	page A-33
16#1401	SD Memory Card Write-protected	An attempt was made to write to a write-protected SD Memory Card when an instruction was executed.	• An attempt was made to write to a write-protected SD Memory Card.	page A-33

Error code	Name	Meaning	Assumed cause	Reference
16#1402	SD Memory Card Insufficient Capacity	The capacity of the SD Memory Card was insufficient when writing to the SD Memory Card for an instruction.	• The SD Memory Card has run out of free space.	page A-34
16#1403	File Does Not Exist	The file specified for an instruction does not exist.	• The specified file does not exist.	page A-34
16#1404	Too Many Files/ Directories	The maximum number of files/directories was exceeded when creating a file/directory for an instruction.	• The number of files or directories exceeded the maximum number.	page A-34
16#1405	File Already in Use	A file specified for an instruction cannot be accessed because it is already being used.	• An instruction attempted to read or write a file already being accessed by another instruction.	page A-35
16#1406	Open Mode Mismatch	A file operation for an instruction was inconsistent with the open mode of the file.	• The file open mode specified by the Open File instruction does not match the file operation attempted by a subsequent SD Memory Card instruction.	page A-35
16#1407	Offset Out of Range	Access to the address is not possible for the offset specified for an instruction.	• An attempt was made to access beyond the size of the file.	page A-35
16#1408	Directory Not Empty	A directory was not empty when the Delete Directory instruction was executed or when an attempt was made to change the directory name.	• A directory was not empty when the Delete Directory instruction was executed. • A directory contained another directory when an attempt was made to change the directory name.	page A-36
16#1409	That File Name Already Exists	An instruction could not be executed because the file name specified for the instruction already exists.	• A file already exists with the same name as the name specified for the instruction to create.	page A-36
16#140A	Write Access Denied	An attempt was made to write to a write-protected file or directory when an instruction was executed.	• The file or directory specified for the instruction to write is write-protected.	page A-36
16#140B	Too Many Files Open	The maximum number of open files was exceeded when opening a file for an instruction.	• The maximum number of open files was exceeded when opening a file for an instruction.	page A-37
16#140C	Directory Does Not Exist	The directory specified for an instruction does not exist.	• The directory specified for an instruction does not exist.	page A-37
16#140D	File or Directory Name Is Too Long	The file name or directory name that was specified for an instruction is too long.	• The file name or directory name that was specified for the instruction to create is too long.	page A-37
16#140E	SD Memory Card Access Failed	SD Memory Card access failed.	• The SD Memory Card is broken. • The SD Memory Card slot is broken.	page A-38
16#1800	EtherCAT Communications Error	Accessing the EtherCAT network failed when an instruction was executed.	• The EtherCAT network is not in a usable status.	page A-38
16#1801	EtherCAT Slave Does Not Respond	Accessing the target slave failed when an instruction was executed.	• The target slave does not exist. • The target slave is not in an operating condition.	page A-38
16#1802	EtherCAT Timeout	A timeout occurred while trying to access an EtherCAT slave when an instruction was executed.	• Communications with the target slave timed out.	page A-39

Error code	Name	Meaning	Assumed cause	Reference
16#1803	Reception Buffer Overflow	The receive data from an EtherCAT slave overflowed the receive buffer when an instruction was executed.	• The receive data from the slave overflowed the receive buffer.	page A-39
16#1804	SDO Abort Error	An SDO abort error was received from an EtherCAT slave when an instruction was executed.	• Depends on the specifications of the slave.	page A-39
16#1805	Saving Packet Monitor File	An instruction for packet monitoring was executed while saving an EtherCAT packet monitor file.	• An instruction for packet monitoring was executed while saving an EtherCAT packet monitor file.	page A-39
16#1806	Packet Monitoring Function Not Started	A Stop EtherCAT Packet Monitor instruction was executed when EtherCAT packet monitoring was stopped.	• A Stop EtherCAT Packet Monitor instruction was executed when EtherCAT packet monitoring was stopped.	page A-40
16#1807	Packet Monitoring Function in Operation	A Start EtherCAT Packet Monitor instruction was executed when EtherCAT packet monitoring was already being executed.	• The Start EtherCAT Packet Monitor instruction was executed again while the EtherCAT packet monitoring function was already in operation.	page A-40
16#1808	Communications Resource Overflow	More than 32 EtherCAT communications instructions were executed at the same time.	• More than 32 EtherCAT communications instructions were executed at the same time. The EtherCAT communications instructions are listed below. • EC_CoESDOWrite instruction • EC_CoESDORead instruction • EC_ConnectSlave instruction • EC_DisconnectSlave instruction • EC_StartMon instruction • EC_SaveMon instruction • EC_StopMon instruction • EC_CopyMon instruction	page A-41
16#1C00	Explicit Message Error	An error response code was returned for an explicit message that was sent with a CIP communications instruction.	• Depends on the nature of the error.	page A-41
16#1C01	Incorrect Route Path	The format of the route path that is specified for a CIP communications instruction is not correct.	• The format of the route path that is specified for a CIP communications instruction is not correct.	page A-41
16#1C02	CIP Handle Out of Range	The handle that is specified for the CIP communications instruction is not correct.	• The handle that is specified for the CIP communications instruction is not correct.	page A-42
16#1C03	CIP Communications Resource Overflow	The maximum resources that you can use for CIP communications instructions at the same time was exceeded.	• More than 32 CIP communications instructions were executed at the same time. • An attempt was made to use more than 32 handles at the same time.	page A-42

Error code	Name	Meaning	Assumed cause	Reference
16#1C04	CIP Timeout	A CIP timeout occurred during execution of a CIP communications instruction.	• A device does not exist for the specified IP address. • The CIP connection for the specified handle timed out and was closed. • Power to the remote device is OFF. • Communications are stopped at the remote device. • The Ethernet cable connector for EtherNet/IP is disconnected. • The Ethernet cable for EtherNet/IP is disconnected. • Noise	page A-42
16#2000	Local IP Address Setting Error	An instruction was executed when there was a setting error in the local IP address.	• An instruction was executed when there was a setting error in the local IP address.	page A-43
16#2001	TCP/UDP Port Already in Use	The UDP or TCP port was already in use when the instruction was executed.	• The UDP or TCP port is already in use.	page A-43
16#2002	Address Resolution Failed	Address resolution failed for a remote node with the domain name that was specified in the instruction.	• The domain name specified for the instruction is not correct. • The hosts and DNS settings in the Controller are incorrect. • The DNS server settings are incorrect.	page A-43

16#2003	Status Error	The status was not suitable for execution of the instruction.	• SktUDPRcv Instruction
			  • The socket is receiving data.
			  • The socket is not open.
			• SktUDPSend Instruction
			  • The socket is sending data.
			  • The socket is not open.
			• SktTCPAccept Instruction
			The specified TCP port is in one of the following states.
			  • The port is being opened.
			  • The port is being closed.
			  • A connection is already established for this instruction for the same IP address and TCP port.
			• SktTCPConnect Instruction
			  • The TCP port that is specified with the *SrcTcpPort* input variable is already open.
			  • The remote node that is specified with *DstAdr* input variable does not exist.
			  • The remote node that is specified with *DstAdr* and *DstTcpPort* input variables is not waiting for a connection.
			• SktTCPRcv Instruction
			  • The specified socket is receiving data.
			  • The specified socket is not connected.
			• SktTCPSend Instruction
			  • The specified socket is sending data.
			  • The specified socket is not connected.
				page A-44

16#2004	Local IP Address Not Set	The local IP address was not set when a socket service instruction was executed.	• There is a BOOTP server setting error. • The BOOTP server does not exist. • The local IP address is not set because operation just started.	page A-45

Error code	Name	Meaning	Assumed cause	Reference
16#2006	Socket Timeout	A timeout occurred for a socket service instruction.	• SktTCPAccept instruction: There was no request for a connection from the remote node during the user-set timeout time. • SktTCPRcv or SktUDPRcv instruction: Data was not received from the remote node during the user-set timeout time.	page A-45
16#2007	Socket Handle Out of Range	The handle that is specified for the socket service instruction is not correct.	• The handle that is specified for the socket service instruction is not correct.	page A-45
16#2008	Socket Communications Resource Overflow	The maximum resources that you can use for socket service instructions at the same time was exceeded.	• More than 17 socket service communications instructions were executed at the same time. • An attempt was made to use more than 16 socket handles at the same time.	page A-46

A-3 Error Code Details

This appendix provides detailed information on error codes.

Error Descriptions

The items that are used to describe individual errors are described in the following copy of an error table.

<table>
<tr><td>Name</td><td colspan="2">Gives the name of the error.</td><td>Error code</td><td>Gives the code of the error.</td></tr>
<tr><td>Meaning</td><td colspan="4">Gives a short description of the error.</td></tr>
<tr><td>Effects</td><td>User program</td><td>Tells what will happen to execution of the user program.*</td><td>Operation</td><td>Provides special information on the operation that results from the error.</td></tr>
<tr><td rowspan="2">System-defined variables</td><td>Variable</td><td colspan="2">Data type</td><td>Name</td></tr>
<tr><td colspan="4">Lists the variable names, data types, and meanings for system-defined variables that provide direct error notification, that are directly affected by the error, or that contain settings that cause the error.</td></tr>
<tr><td rowspan="2">Cause and correction</td><td>Assumed cause</td><td colspan="2">Correction</td><td>Prevention</td></tr>
<tr><td colspan="4">Lists the possible causes, corrections, and preventive measures for the error.</td></tr>
<tr><td>Precautions/ Remarks</td><td colspan="4">Provides precautions, restrictions, and supplemental information.</td></tr>
</table>

* One of the following:
Continues: Execution of the user program will continue.
Stops: Execution of the user program stops.
Starts: Execution of the user program starts.

Name	Input Value Out of Range		Error code	16#0400
Meaning	An input parameter for an instruction exceeded the valid range for an input variable. Or, division by an integer of 0 occurred in division or remainder calculations.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	An input parameter for an instruction exceeded the valid range for an input variable. Or, division by an integer of 0 occurred in division or remainder calculations.		Check the valid range for the input variables of the instruction. Make sure the input parameters are within the valid range and that no division by 0 or remainder calculation for 0 is performed.	Set the value of the input parameter to the instruction so that the input range is not exceeded.
Precautions/ Remarks	None			

Name	Input Mismatch		Error code	16#0401
Meaning	The relationship for the instruction input parameters did not meet required conditions. Or, a numeric value during or after instruction execution did not meet conditions.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The relationship for an input parameter did not meet required conditions.		Check the meaning and the relationship of the input variables of the instruction. Correct them so that the relationships for the input parameters meet the required conditions.	Set the input parameter to the instruction so that the value meets the conditions of the relationship for the input variables.
	A value when processing an instruction or in the result does not meet the conditions.		Check the execution process of the instruction. Set the value of the input parameter so that it does not cause inappropriate processing results.	Check the execution process of the instruction. Set the input parameter so that it does not cause this error during processing.
Precautions/ Remarks	None			

Name	Floating-point Error		Error code	16#0402
Meaning	Non-numeric data was input for a floating-point number input parameter to an instruction.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	Non-numeric data was input for a floating-point number input parameter to an instruction.		Correct the instruction so that a numeric value is input for the floating-point number input parameter.	Use numeric values for the floating-point number input parameters.
Precautions/ Remarks	None			

Name	BCD Error			Error code	16#0403
Meaning	A value that was not BCD was input for a BCD input parameter to an instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	A hexadecimal digit of A, B, C, D, E, or F was input for a BCD input parameter to an instruction.		Correct the instruction so that BCD data is input for the BCD input parameter.		Change the BCD input parameter for the instruction to BCD data.
Precautions/ Remarks	None				

Name	Signed BCD Error			Error code	16#0404
Meaning	An illegal value was input for the most significant digit for a signed BCD input parameter to an instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	An illegal value was input for the most significant digit for a signed BCD input parameter to an instruction. • The most-significant digit was 2 to F when *_BCD0* was specified as the BCD format. • The most-significant digit was A, B, C, D, or E when *_BCD2* was specified as the BCD format. • The most-significant digit was B, C, D, or E when *_BCD3* was specified as the BCD format.		Correct the instruction so that proper signed BCD data is input for the BCD input parameter.		Set the most-significant digit of the signed BCD data input parameter for the instruction to the correct value.
Precautions/ Remarks	None				

Name	Illegal Bit Position Specified			Error code	16#0405
Meaning	The bit position specified for an instruction was illegal.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The bit position specified for an instruction exceeds the data range.		Correct the instruction so that the bit position specified for an instruction does not exceed the data range.		Use the instruction so that the bit position specified for an instruction does not exceed the data range.
Precautions/ Remarks	None				

Name	Illegal Data Position Specified		Error code	16#0406
Meaning	The data position specified for an instruction exceeded the data area range.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The data position or data size specified for an instruction exceeded the data area range.	Correct the instruction so that the data position or data size specified for an instruction does not exceed the range of the data area.	Use the instruction so that the data position or data size specified for an instruction does not exceed the data range.	
Precautions/ Remarks	None			

Name	Data Range Exceeded		Error code	16#0407
Meaning	The results of instruction processing exceeded the data area range of the output parameter.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The results of instruction processing, such as the number of array elements, exceeded the data area range of the output parameter.	Correct the input parameters so that the processing result of the instruction does not exceed the range of the data area of the output parameter.	Set the input parameter so that the processing result of the instruction does not exceed the range of the data area of the output parameter.	
Precautions/ Remarks	None			

Name	No Errors to Clear		Error code	16#0409
Meaning	An instruction to clear a Controller error was executed when there was no error in the Controller.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The output or Unit operation is not affected.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	An instruction to clear a Controller error was executed when there was no error in the Controller.	Correct the program so that the instruction is executed when there is a Controller error.	Write the program so that the instruction is executed when there is a Controller error.	
Precautions/ Remarks	None			

Name	No User Errors to Clear		Error code	16#040B
Meaning	An instruction to clear user-defined errors was executed when there was no user-defined error.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The output or Unit operation is not affected.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	An instruction to clear user-defined errors was executed when there was no user-defined error.		Correct the program so that the instruction is executed when there is a user-defined error.	Write the program so that the instruction is executed when there is a user-defined error.
Precautions/ Remarks	None			

Name	Limit Exceeded for User-defined Errors		Error code	16#040C
Meaning	An attempt was made to use the Create User-defined Error instruction to create more than the maximum number of user-defined errors.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The output or Unit operation is not affected.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	An attempt was made to use the Create User-defined Error instruction to create more than the maximum number of user-defined errors.		Execute the Reset User-defined Error instruction. Monitor the number of user-defined errors in the system-defined variable to check the number of user-defined errors.	Write the program so that it checks the number of user-defined errors as a condition to execute the user-defined error instruction.
Precautions/ Remarks	None			

Name	Illegal Unit Specified		Error code	16#040D
Meaning	The Unit specified for an instruction does not exist.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The output or Unit operation is not affected.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	A Unit that does not exist in the Unit configuration information was specified.		Correct the unit number in the instruction so that it specifies a Unit in the Unit configuration and make sure that the actual Unit exists.	Make sure that unit numbers in instructions specify Units in the Unit configuration and make sure that the actual Units exist.
	A Unit that is in the Unit configuration information was specified, but the Units does not actually exist in the Controller.			
Precautions/ Remarks	None			

Name	Unit Restart Failed			Error code	16#040F
Meaning	Restarting a Special I/O Unit or CPU Bus Unit failed.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The output or Unit operation is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The Special I/O Unit or CPU Bus Unit is processing data.		Wait a few moments and then restart the Special I/O Unit or CPU Bus Unit.		Check to be sure that Special I/O Units and CPU Bus Units are not processing data before restarting them from the user program.
Precautions/ Remarks	None				

Name	Text String Format Error			Error code	16#0410
Meaning	The text string input to an instruction is not correct.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The text string that is input to the instruction for conversion to a number does not represent a number or it does not represent a positive number.		Correct the text string so that it is properly formatted for the instruction.		When converting a text string to a number, make sure that the text string that is input to the instruction represents a number. If the number must be positive, make sure the text string represents a positive number.
	The input text string does not end in NULL.		Correct the text string that is input to the instruction so that it ends in NULL.		When converting a text string to a number, make sure that the text string that is input to the instruction ends in NULL.
Precautions/ Remarks	None				

Name	Illegal Program Specified			Error code	16#0411
Meaning	The program specified for an instruction does not exist.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The program specified by the function does not exist (e.g., it was deleted).		Make sure that the program that is specified by the instruction exists. Or, add the program that is specified for the instruction.		Make sure that the programs that are specified by instructions exist. Be careful not to delete any programs that are used by instructions.
Precautions/ Remarks	None				

Name	Undefined CJ-series Memory Address			Error code	16#0413
Meaning	The required specification is missing for a variable for which CJ-series Unit memory must be specified.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The required AT specification is missing for a variable for which CJ-series Unit memory must be specified.		Correct the program so that it uses the AT specification to specify CJ-series Unit memory when doing so is required by the variable.		Write the program so that it uses an AT designation to specify CJ-series Unit memory when doing so is required by the variable.
Precautions/ Remarks	None				

Name	Stack Underflow			Error code	16#0414
Meaning	There is no data in a stack.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	An attempt was made to read data from a stack that contains no data.		Correct the program so that the data is read only after it is stored in the stack.		Correct the program so that the data is read only after it is stored in the stack.
Precautions/ Remarks	None				

Name	Illegal Number of Array Elements or Dimensions			Error code	16#0416
Meaning	The valid range was exceeded for the number of array elements or dimensions in an array I/O parameter for an instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The valid range was exceeded for the number of array elements or dimensions in an array I/O parameter for an instruction.		Correct the instruction so that the valid range for the number of array elements or dimensions in an array I/O parameter is not exceeded.		Correct the instruction so that the valid range for the number of array elements or dimensions in an array I/O parameter is not exceeded.
Precautions/ Remarks	None				

Name	Specified Task Does Not Exist			Error code	16#0417
Meaning	The task specified for the instruction does not exist.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The specified task does not exist.		Correct the user program so that it specifies an existing task.		Write the user program so that it specifies only existing tasks.
Precautions/ Remarks	None				

Name	Unallowed Task Specification		Error code	16#0418
Meaning	An unallowed task was specified for an instruction.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The local task, the primary periodic task, or a periodic task was specified.		Correct the user program so that it specifies an event task that is not the local task.	Write the user program so that it specifies event tasks that are not the local task.
Precautions/ Remarks	None			

Name	Incorrect Data Type		Error code	16#0419
Meaning	A data type that cannot be used for an instruction is specified for an input or in-out variable.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	A data type that cannot be used for an instruction is specified for an input or in-out variable.		Check the data types of the input and in-out variables of the instruction and correct them to correct data types.	Check the allowed data types for input and in-out variables for the instruction and use correct data types.
Precautions/ Remarks	None			

Name	Multi-execution of Instructions		Error code	16#041A
Meaning	Multi-execution was specified for an instruction that does not support it.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	Execution of an instruction that does not support multi-execution of instructions was specified more than once.		Correct the program so that any instance of an instruction that does not support multi-execution is completed before another instance is executed.	Write the user program so that any instance of an instruction that does not support multi-execution is completed before another instance is executed.
Precautions/ Remarks	None			

Name	FINS Error			Error code	16#0800
Meaning	An error occurred when a FINS command was sent or received.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	An error occurred when a FINS command was sent or received.		Check the value of the *ErrorIDEx* output variable from the instruction and refer to the description in this manual for the communications response code (ErrorIDEx) with the same value for the instruction.		Read the description of *ErrorIDEx* in advance for the instruction and program correctly.
Precautions/ Remarks	None				

Name	FINS Port Already in Use			Error code	16#0801
Meaning	The FINS port is being used.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The communications output or Unit operation is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The FINS port is being used.		Correct the program by inserting *_Port.isAvailable* in a N.O. input condition.		Insert *_Port.isAvailable* in a N.O. input condition when you create the program.
Precautions/ Remarks	None				

Name	Illegal Serial Communications Mode			Error code	16#0C00
Meaning	The Serial Communications Unit is not in the serial communications mode required to execute an instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The communications output will follow the specifications of the instruction. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The serial communications port for the Serial Communications Unit is not set to the mode expected by the instruction.		Change to the serial communications mode required to execute the instruction. Or, correct the program so that it only uses instructions that can be executed in the current mode.		Set the Serial Communications Unit to the serial communications mode required to execute the instruction. Or, correct the program so that it only uses instructions that can be executed in the currently set mode.
Precautions/ Remarks	None				

Name	Port Setup Already Busy			Error code	16#0C02
Meaning	A Change Port Setup instruction was executed during execution of another Change Port Setup instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. Communications output will follow the specifications of the instruction. The operation of the Unit will follow the changes made to the port settings with the first instruction.	
System-defined variables	Variable		Data type		Name
	_CJB_SCU##1ChgSta		BOOL		Serial Communications Unit ## Port 1 Settings Changing Flag
	_CJB_SCU##2ChgSta		BOOL		Serial Communications Unit ## Port 2 Settings Changing Flag
Cause and correction	Assumed cause		Correction		Prevention
	A Change Port Setup instruction was executed during execution of another Change Port Setup instruction.		Correct the program so that the instruction is not executed while changing port settings.		Write the program so that the instruction is not executed while changing port settings.
Precautions/ Remarks	None				

Name	SD Memory Card Access Failure			Error code	16#1400
Meaning	SD Memory Card access failed when an instruction was executed.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	An SD Memory Card is either not inserted or is not inserted properly.		Insert the SD Memory Card correctly.		Make sure that the SD Memory Card is inserted properly.
	The SD Memory Card is broken.		Replace the SD Memory Card with one that operates normally.		None
	The SD Memory Card slot is broken.		If this error persists even after making the above two corrections, replace the CPU Unit.		None
Precautions/ Remarks	None				

Name	SD Memory Card Write-protected			Error code	16#1401
Meaning	An attempt was made to write to a write-protected SD Memory Card when an instruction was executed.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	An attempt was made to write to a write-protected SD Memory Card.		Remove write protection from the SD Memory Card. Slide the small switch on the side of the SD Memory Card from the LOCK position to the writable position.		Use an SD Memory Card that is not write-protected when writing to the SD Memory Card.
Precautions/ Remarks	None				

Name	SD Memory Card Insufficient Capacity			Error code	16#1402
Meaning	The capacity of the SD Memory Card was insufficient when writing to the SD Memory Card for an instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The SD Memory Card has run out of free space.		Replace the SD Memory Card for one with sufficient available capacity.		Use an SD Memory Card with sufficient available space when you add files to it.
Precautions/ Remarks	Do not remove the SD Memory Card during Card access. That may damage the SD Memory Card or corrupt the data on it.				

Name	File Does Not Exist			Error code	16#1403
Meaning	The file specified for an instruction does not exist.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The specified file does not exist.		Make sure that the filename that is specified for the instruction exists. Or, modify the filename so that it matches the filename specified for the instruction.		Make sure that the filename that is specified for the instruction exists.
Precautions/ Remarks	None				

Name	Too Many Files/ Directories			Error code	16#1404
Meaning	The maximum number of files/directories was exceeded when creating a file/directory for an instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The number of files or directories exceeded the maximum number.		Delete any unnecessary files and/or directories. Or, replace the SD Memory Card with one that has fewer files and directories compared to the maximum number of files and directories for FAT16 or FAT32.		Delete unnecessary files and directories so that there are not too many files and directories on the SD Memory Card. Regularly replace the SD Memory Card when the number of files grows constantly.
Precautions/ Remarks	None				

Name	File Already in Use			Error code	16#1405
Meaning	A file specified for an instruction cannot be accessed because it is already being used.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	An instruction attempted to read or write a file already being accessed by another instruction.		Correct the program so that the relevant instruction is only executed when the *Busy* output variable for all other instructions for the same file are FALSE.		When you execute multiple instructions that access the same file, write the program so that the instructions are not executed simultaneously. Make sure that the *Busy* output variable for all other instructions for the same file is FALSE.
Precautions/ Remarks	None				

Name	Open Mode Mismatch			Error code	16#1406
Meaning	A file operation for an instruction was inconsistent with the open mode of the file.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The file open mode specified by the Open File instruction does not match the file operation attempted by a subsequent SD Memory Card instruction.		Correct the Open File instruction to open the file in an open mode that is suitable for the file operation.		Change the Open File instruction to open the file in an open mode that is suitable for the file operation.
Precautions/ Remarks	None				

Name	Offset Out of Range			Error code	16#1407
Meaning	Access to the address is not possible for the offset specified for an instruction.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	An attempt was made to access beyond the size of the file.		Decrease the offset specified for the instruction.		Include information in the file so that the file format can be identified, and modify the program to check that information in order to perform appropriate file seeking.
Precautions/ Remarks	None				

Name	Directory Not Empty			Error code	16#1408
Meaning	A directory was not empty when the Delete Directory instruction was executed or when an attempt was made to change the directory name.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	A directory was not empty when the Delete Directory instruction was executed.		Delete all files in the relevant directory.		Check the contents of a directory before you delete the directory using the Delete Directory instruction or before you change the directory name.
	A directory contained another directory when an attempt was made to change the directory name.		Delete all directories from the relevant directory.		
Precautions/ Remarks	None				

Name	That File Name Already Exists			Error code	16#1409
Meaning	An instruction could not be executed because the file name specified for the instruction already exists.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	A file already exists with the same name as the name specified for the instruction to create.		Correct the program so that the filename specified for the instruction does not already exist. Or, delete the existing file.		Make sure that the file specified does not already exist when you create a file with an instruction.
Precautions/ Remarks	When you delete an existing file, check to make sure that you no longer need the file.				

Name	Write Access Denied			Error code	16#140A
Meaning	An attempt was made to write to a write-protected file or directory when an instruction was executed.				
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.	
System-defined variables	Variable		Data type		Name
	None		---		---
Cause and correction	Assumed cause		Correction		Prevention
	The file or directory specified for the instruction to write is write-protected.		Remove write protection from the file or directory specified for the instruction. Or, change the filename of the file to write.		Do not write-protect any files that need to be written to.
Precautions/ Remarks	Before you remove write protection from a file, be sure it is OK to overwrite the file.				

Name	Too Many Files Open		Error code	16#140B
Meaning	The maximum number of open files was exceeded when opening a file for an instruction.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The maximum number of open files was exceeded when opening a file for an instruction.		Correct the program to decrease the number of open files.	Decrease the number of files. Or, write the program so that files that no longer need to be open are closed in order to prevent too many files from being open at once.
Precautions/ Remarks	None			

Name	Directory Does Not Exist		Error code	16#140C
Meaning	The directory specified for an instruction does not exist.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The directory specified for an instruction does not exist.		Correct the program so that the directory specified for the instruction exists. Or, create the relevant directory in advance.	Make sure that the directory specified for the instruction directory actually exists when using an instruction that accesses a directory.
Precautions/ Remarks	None			

Name	File or Directory Name Is Too Long		Error code	16#140D
Meaning	The file name or directory name that was specified for an instruction is too long.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The file name or directory name that was specified for the instruction to create is too long.		Correct the program so that the file name or directory name specified for the instruction is within FAT16 or FAT32 restrictions.	Write the program so that the specified file names and directory names are within FAT16 or FAT32 restrictions.
Precautions/ Remarks	None			

Name	SD Memory Card Access Failed		Error code	16#140E
Meaning	SD Memory Card access failed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. The operation of the Unit is not affected.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The SD Memory Card is broken.	Replace the SD Memory Card.	None	
	The SD Memory Card slot is broken.	If this error occurs even after making the above correction, replace the CPU Unit.	None	
Precautions/ Remarks	None			

Name	EtherCAT Communications Error		Error code	16#1800
Meaning	Accessing the EtherCAT network failed when an instruction was executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The EtherCAT network is not in a usable status.	Check the operation status of the EtherCAT network by checking the status of the EtherCAT master. Use this information to correct the cause of the problem.	Depends on the nature of the error.	
Precautions/ Remarks	None			

Name	EtherCAT Slave Does Not Respond		Error code	16#1801
Meaning	Accessing the target slave failed when an instruction was executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The target slave does not exist.	Specify an existing node address.	Specify an existing node address for the target slave.	
	The target slave is not in an operating condition.	Check the status of the target EtherCAT slave. Make sure that the target slave is in a usable status.	Make sure that the target slave is in a usable status.	
Precautions/ Remarks	None			

Name	EtherCAT Timeout		Error code	16#1802
Meaning	A timeout occurred while trying to access an EtherCAT slave when an instruction was executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	Communications with the target slave timed out.	Check the operating status of the target slave and correct the cause of the problem.	Depends on the nature of the error.	
Precautions/ Remarks	None			

Name	Reception Buffer Overflow		Error code	16#1803
Meaning	The receive data from an EtherCAT slave overflowed the receive buffer when an instruction was executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications. It will not be possible to receive data from the slave.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The receive data from the slave overflowed the receive buffer.	Set the size of the reception buffer to a value larger than the size of the receive data from the slave.	Set the size of the receive buffer to a value larger than the size of the receive data from the slave.	
Precautions/ Remarks	None			

Name	SDO Abort Error		Error code	16#1804
Meaning	An SDO abort error was received from an EtherCAT slave when an instruction was executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	Depends on the specifications of the slave.	Refer to the manual for the slave and correct the problem.	Refer to the manual for the slave and take the necessary steps to prevent the problem.	
Precautions/ Remarks	None			

Name	Saving Packet Monitor File		Error code	16#1805
Meaning	An instruction for packet monitoring was executed while saving an EtherCAT packet monitor file.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	An instruction for packet monitoring was executed while saving an EtherCAT packet monitor file.	Execute the instruction for packet monitoring after saving the EtherCAT packet monitor file is completed. You can check packet monitor file save status to see if saving a packet monitor file is completed.	Execute packet monitoring instructions only after the packet monitor file is saved. You can check packet monitor file save status to see if saving a packet monitor file is completed.	
Precautions/ Remarks	None			

Name	Packet Monitoring Function Not Started		Error code	16#1806
Meaning	A Stop EtherCAT Packet Monitor instruction was executed when EtherCAT packet monitoring was stopped.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	A Stop EtherCAT Packet Monitor instruction was executed when EtherCAT packet monitoring was stopped.	Execute the Stop EtherCAT Packt Monitor instruction after starting the packet monitoring function. You can check packet monitoring function operation status to see if the packet monitoring function is currently in operation.	Execute the Stop EtherCAT Packet Monitor instruction after starting the packet monitoring function. You can check packet monitoring function operation status to see if the packet monitoring function is currently in operation.	
Precautions/ Remarks	None			

Name	Packet Monitoring Function in Operation		Error code	16#1807
Meaning	A Start EtherCAT Packet Monitor instruction was executed when EtherCAT packet monitoring was already being executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The Start EtherCAT Packet Monitor instruction was executed again while the EtherCAT packet monitoring function was already in operation.	Execute the Start EtherCAT Packet Monitor instruction after the packet monitoring function was stopped. You can check packet monitoring function operation status to see if the packet monitoring function is stopped.	Execute the Start EtherCAT Packet Monitor instruction after the packet monitoring function is stopped. You can check packet monitoring function operation status to see if the packet monitoring function is stopped.	
Precautions/ Remarks	None			

Name	Communications Resource Overflow		Error code	16#1808
Meaning	More than 32 EtherCAT communications instructions were executed at the same time.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	More than 32 EtherCAT communications instructions were executed at the same time. The EtherCAT communications instructions are listed below. • EC_CoESDOWrite instruction • EC_CoESDORead instruction • EC_ConnectSlave instruction • EC_DisconnectSlave instruction • EC_StartMon instruction • EC_SaveMon instruction • EC_StopMon instruction • EC_CopyMon instruction		Correct the user program so that no more than 32 EtherCAT communications instructions are executed at the same time.	Write the user program so that no more than 32 EtherCAT communications instructions are executed at the same time.
Precautions/ Remarks	None			

Name	Explicit Message Error		Error code	16#1C00
Meaning	An error response code was returned for an explicit message that was sent with a CIP communications instruction.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	Depends on the nature of the error.		Check the value of the *ErrorIDEx* output variable from the instruction and refer to the description in this manual of the CIP message error code.	Depends on the nature of the error. Refer to the description in this manual of the CIP message error code.
Precautions/ Remarks	None			

Name	Incorrect Route Path		Error code	16#1C01
Meaning	The format of the route path that is specified for a CIP communications instruction is not correct.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The format of the route path that is specified for a CIP communications instruction is not correct.		Correct the route path that is specified by the instruction.	Make sure that the instructions specify correct route paths.
Precautions/ Remarks	None			

Name	CIP Handle Out of Range		Error code	16#1C02
Meaning	The handle that is specified for the CIP communications instruction is not correct.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	The handle that is specified for the CIP communications instruction is not correct.	Correct the handle for the instruction to the handle that was obtained with the CIPOpen instruction.	Specify handles that were obtained with the CIPOpen instruction.	
Precautions/ Remarks	None			

Name	CIP Communications Resource Overflow		Error code	16#1C03
Meaning	The maximum resources that you can use for CIP communications instructions at the same time was exceeded.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	More than 16 CIP communications instructions were executed at the same time.	Correct the user program so that no more than 16 CIP communications instructions are executed at the same time.	Write the user program so that no more than 16 CIP communications instructions are executed at the same time.	
	An attempt was made to use more than 32 handles at the same time.	Correct the user program so that no more than 32 handles are used at the same time.	Write the user program so that no more than 32 handles are used at the same time.	
Precautions/ Remarks	None			

Name	CIP Timeout		Error code	16#1C04
Meaning	A CIP timeout occurred during execution of a CIP communications instruction.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable	Data type	Name	
	None	---	---	
Cause and correction	Assumed cause	Correction	Prevention	
	A device does not exist for the specified IP address.	Correct the specified IP address to the IP address of the remote device.	Specify the correct IP address of the remote device.	
	The CIP connection for the specified handle timed out and was closed.	Execute the instruction before the connection times out. Or, increase the timeout time of the connection.	Execute the instruction before the connection times out.	
	Power to the remote device is OFF.	Check the status of the remote device and start it normally.	Check the status of the remote device and start it normally.	
	Communications are stopped at the remote device.			
	The Ethernet cable connector for EtherNet/IP is disconnected.	Reconnect the connector and make sure it is mated correctly.	Connect the connector securely.	
	The Ethernet cable for EtherNet/IP is disconnected.	Replace the Ethernet cable.	None	
	Noise	Implement noise countermeasures if there is excessive noise.	Implement noise countermeasures if there is excessive noise.	
Precautions/ Remarks	None			

Name	Local IP Address Setting Error		Error code	16#2000
Meaning	An instruction was executed when there was a setting error in the local IP address.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	An instruction was executed when there was a setting error in the local IP address.		There was a TCP/IP Basic Setting Error (IP Address Setting Error) when the instruction was executed. Remove the cause of the TCP/IP Basic Setting Error.	Set the IP addresses correctly so that a TCP/IP Basic Setting Error does not occur.
Precautions/ Remarks	None			

Name	TCP/UDP Port Already in Use		Error code	16#2001
Meaning	The UDP or TCP port was already in use when the instruction was executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The UDP or TCP port is already in use.		Correct the user program so that an unused port is specified for the instruction.	Write the user program so that used ports are not specified for instructions.
Precautions/ Remarks	None			

Name	Address Resolution Failed		Error code	16#2002
Meaning	Address resolution failed for a remote node with the domain name that was specified in the instruction.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The domain name specified for the instruction is not correct.		Correct the domain name that is specified in the instruction.	Specify correct domain names in instructions.
	The hosts and DNS settings in the Controller are incorrect.		Correct the hosts and DNS settings in the Controller.	Check the hosts and DNS settings in the Controller and make sure they are correct.
	The DNS server settings are incorrect.		Correct the DNS server settings.	Check that there are no mistakes in the DNS server settings.
Precautions/ Remarks	None			

<table>
<tr><td>Name</td><td colspan="2">Status Error</td><td>Error code</td><td>16#2003</td></tr>
<tr><td>Meaning</td><td colspan="4">The status was not suitable for execution of the instruction.</td></tr>
<tr><td>Effects</td><td>User program</td><td>Continues.</td><td>Operation</td><td>The relevant instruction will end according to specifications.</td></tr>
<tr><td rowspan="2">System-defined variables</td><td colspan="2">Variable</td><td>Data type</td><td>Name</td></tr>
<tr><td colspan="2">None</td><td>---</td><td>---</td></tr>
<tr><td>Cause and correction</td><td colspan="2">Assumed cause</td><td>Correction</td><td>Prevention</td></tr>
<tr><td colspan="2">• SktUDPRcv Instruction
  • The socket is receiving data.
  • The socket is not open.
• SktUDPSend Instruction
  • The socket is sending data.
  • The socket is not open.
• SktTCPAccept Instruction
The specified TCP port is in one of the following states.
  • The port is being opened.
  • The port is being closed.
  • A connection is already established for this instruction for the same IP address and TCP port.
• SktTCPConnect Instruction
  • The TCP port that is specified with the SrcTcpPort input variable is already open.
  • The remote node that is specified with DstAdr input variable does not exist.
  • The remote node that is specified with DstAdr and DstTcpPort input variables is not waiting for a connection.
• SktTCPRcv Instruction
  • The specified socket is receiving data.
  • The specified socket is not connected.
• SktTCPSend Instruction
  • The specified socket is sending data.
  • The specified socket is not connected.</td><td>Remove the cause of the error for the instruction.</td><td>Do not execute the instruction when it will cause an error.</td></tr>
<tr><td>Precautions/ Remarks</td><td colspan="4">None</td></tr>
</table>

Name	Local IP Address Not Set		Error code	16#2004
Meaning	The local IP address was not set when a socket service instruction was executed.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	There is a BOOTP server setting error.		Correct any errors in the BOOTP server settings.	Check that there are no mistakes in the BOOTP server settings.
	The BOOTP server does not exist.		Make sure that the BOOTP server has started normally and is normally connected to the network.	Make sure that the BOOTP server has started normally and is normally connected to the network.
	The local IP address is not set because operation just started.		Wait until the local IP address is set before executing socket service instructions.	Wait until the local IP address is set before executing socket service instructions.
Precautions/ Remarks	None			

Name	Socket Timeout		Error code	16#2006
Meaning	A timeout occurred for a socket service instruction.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	SktTCPAccept instruction: There was no request for a connection from the remote node during the user-set timeout time.		Correct the system and user program so that there is a connection request from the remote node within the timeout time after the instruction is executed. Or, increase the timeout time.	Set up the system and user program so that there is a connection request from the remote node within the timeout time after the instruction is executed.
	SktTCPRcv or SktUDPRcv instruction: Data was not received from the remote node during the user-set timeout time.		Correct the system and user program so that data is received from the remote node within the timeout time after the instruction is executed. Or, increase the timeout time.	Set up the system and user program so that data is received from the remote node within the timeout time after the instruction is executed.
Precautions/ Remarks	None			

Name	Socket Handle Out of Range		Error code	16#2007
Meaning	The handle that is specified for the socket service instruction is not correct.			
Effects	User program	Continues.	Operation	The relevant instruction will end according to specifications.
System-defined variables	Variable		Data type	Name
	None		---	---
Cause and correction	Assumed cause		Correction	Prevention
	The handle that is specified for the socket service instruction is not correct.		Correct the socket handle for the instruction to the handle that was obtained with one of the following instructions. • SktUDPCreate instruction • SktTCPConnect instruction • SktTCPAccept instruction	Specify handles that are obtained with the following instructions. • SktUDPCreate instruction • SktTCPConnect instruction • SktTCPAccept instruction
Precautions/ Remarks	None			

<table>
<tr><td>Name</td><td colspan="2">Socket Communications Resource Overflow</td><td>Error code</td><td>16#2008</td></tr>
<tr><td>Meaning</td><td colspan="4">The maximum resources that you can use for socket service instructions at the same time was exceeded.</td></tr>
<tr><td>Effects</td><td>User program</td><td>Continues.</td><td>Operation</td><td>The relevant instruction will end according to specifications.</td></tr>
<tr><td rowspan="2">System-defined variables</td><td colspan="2">Variable</td><td colspan="2">Data type</td><td>Name</td></tr>
<tr><td colspan="2">None</td><td colspan="2">---</td><td>---</td></tr>
<tr><td rowspan="3">Cause and correction</td><td colspan="2">Assumed cause</td><td colspan="2">Correction</td><td>Prevention</td></tr>
<tr><td colspan="2">More than 17 socket service communications instructions were executed at the same time.</td><td colspan="2">Correct the user program so that no more than 17 socket service instructions are executed at the same time.</td><td>Write the user program so that no more than 17 socket service instructions are executed at the same time.</td></tr>
<tr><td colspan="2">An attempt was made to use more than 16 socket handles at the same time.</td><td colspan="2">Correct the user program so that no more than 16 socket handles are used at the same time.</td><td>Write the user program so that no more than 16 socket handles are used at the same time.</td></tr>
<tr><td>Precautions/ Remarks</td><td colspan="4">None</td></tr>
</table>

A-4 SDO Abort Codes

As reference information, the following table lists the SDO abort codes for EtherCAT communications. The abort codes that are used in actual communications are specified by the slaves. Refer to the slave manuals when programming communications.

Value	Meaning
16#05030000	Toggle bit not changed
16#05040000	SDO protocol timeout
16#05040001	Client/Server command specifier not valid or unknown
16#05040005	Out of memory
16#06010000	Unsupported access to an object
16#06010001	Attempt to read to a write only object
16#06010002	Attempt to write to a read only object
16#06020000	The object does not exist in the object directory
16#06040041	The object cannot be mapped into the PDO
16#06040042	The number and length of the objects to be mapped would exceed the PDO length
16#06040043	General parameter incompatibility reason
16#06040047	General internal incompatibility in the device
16#06060000	Access failed due to a hardware error
16#06070010	Data type does not match, length of service parameter does not match
16#06070012	Data type does not match, length of service parameter too high
16#06070013	Data type does not match, length of service parameter too low
16#06090011	Subindex does not exist
16#06090030	Value range of parameter exceeded (only for write access)
16#06090031	Value of parameter written too high
16#06090032	Value of parameter written too low
16#06090036	Maximum value is less than minimum value
16#08000000	General error
16#08000020	Data cannot be transferred or stored to the application
16#08000021	Data cannot be transferred or stored to the application because of local control
16#08000022	Data cannot be transferred or stored to the application because of the present device state
16#08000023	Object dictionary dynamic generation failed or no object dictionary is present

Source: EtherCAT Specification Part 6 Application Layer Protocol Specification.

Document No.: ETG.1000.6 S (R) V1.0.2

I

Index

Index

Symbols

& (Logical AND) 2-286
* (Multiplication) 2-161
** (Exponentiation) 2-187
_BCD_TO_*
(BCD-to-Unsigned Integer Conversion Group) 2-212
_TO_*
(Bit String-to-Bit String Conversion Group) 2-242
_TO_*
(Bit String-to-Integer Conversion Group) 2-239
_TO_*
(Bit String-to-Real Number Conversion Group) 2-244
_TO_*
(Integer-to-Bit String Conversion Group) 2-235
_TO_*
(Integer-to-Integer Conversion Group) 2-232
_TO_*
(Integer-to-Real Number Conversion Group) 2-237
_TO_*
(Real Number-to-Bit String Conversion Group) 2-249
_TO_*
(Real Number-to-Integer Conversion Group) 2-246
_TO_* (Real Number-to-Real Number
Conversion Group) 2-251
_TO_BCD_*
(Unsigned Integer-to-BCD Conversion Group) 2-215
**_TO_STRING
(Bit String-to-Text String Conversion Group) 2-255
**_TO_STRING
(Integer-to-Text String Conversion Group) 2-253
**_TO_STRING
(Real Number-to-Text String Conversion Group) 2-257
+ (Addition) 2-152
+OU (Addition with Overflow/Underflow Check) 2-154
- (Subtraction) 2-156
-OU (Subtraction with Overflow/Underflow Check) ... 2-158
/ (Division) 2-166
< (Less Than) 2-88
<= (Less Than Or Equal) 2-88
<> (Not Equal) 2-86
= (Equal) 2-84
> (Greater Than) 2-88
>= (Greater Than Or Equal) 2-88

A

ABS (Absolute Value) 2-170
Absolute Value 2-170
Accept TCP Socket 2-767
Accumulation Timer 2-126
AccumulationTimer (Accumulation Timer) 2-126
ACOS (Principal Arc Cosine) 2-177
ADD (Addition) 2-152
Add Time 2-544
Add Time to Date and Time 2-548
Add Time to Time of Day 2-546
ADD_DT_TIME (Add Time to Date and Time) 2-548
Addition 2-152
Addition with Overflow/Underflow Check 2-154
AddOU (Addition with Overflow/Underflow Check) 2-154
ADD_TIME (Add Time) 2-544
ADD_TOD_TIME (Add Time to Time of Day) 2-546
AND 2-16
AND (AND) 2-16
AND (Logical AND) 2-286
AND NOT 2-16
ANDN (AND NOT) 2-16
Array Addition 2-193
Array BCD Conversion 2-227
Array Comparison Equal 2-105
Array Comparison Greater Than 2-107
Array Comparison Greater Than Or Equal 2-107
Array Comparison Less Than 2-107
Array Comparison Less Than Or Equal 2-107
Array Comparison Not Equal 2-105
Array Data Exchange 2-333
Array Element Standard Deviation 2-203
Array Logical AND 2-293
Array Logical Exclusive NOR 2-293
Array Logical Exclusive OR 2-293
Array Logical OR 2-293
Array Maximum 2-312
Array Mean 2-201
Array Minimum 2-312
Array Move 2-335
Array N-element Left Shift 2-357
Array N-element Right Shift 2-357
Array Search 2-314
Array Subtraction 2-197
Array Unsigned Integer Conversion 2-229
Array Value Addition 2-195
Array Value Comparison Equal 2-110
Array Value Comparison Greater Than 2-112
Array Value Comparison Greater Than Or Equal 2-112
Array Value Comparison Less Than 2-112
Array Value Comparison Less Than Or Equal 2-112
Array Value Comparison Not Equal 2-110
Array Value Subtraction 2-199
Array-to-Text String Conversion 2-443
AryAdd (Array Addition) 2-193
AryAddV (Array Value Addition) 2-195
AryAnd (Array Logical AND) 2-293
AryByteTo (Conversion from Byte Array) 2-458
AryCmpEQ (Array Comparison Equal) 2-105
AryCmpEQV (Array Value Comparison Equal) 2-110
AryCmpGE
(Array Comparison Greater Than Or Equal) 2-107

AryCmpGEV
(Array Value Comparison Greater Than Or Equal) 2-112
AryCmpGT
(Array Comparison Greater Than) 2-107
AryCmpGTV
(Array Value Comparison Greater Than) 2-112
AryCmpLE
(Array Comparison Less Than Or Equal) 2-107
AryCmpLEV
(Array Value Comparison Less Than Or Equal) 2-112
AryCmpLT (Array Comparison Less Than) 2-107
AryCmpLTV (Array Value Comparison Less Than) ... 2-112
AryCmpNE (Array Comparison Not Equal) 2-105
AryCmpNEV (Array Value Comparison Not Equal) .. 2-110
AryCRC16 (Calculate Array CRC-16) 2-516
AryCRCCCITT (Calculate Array CRC-CCITT) 2-514
AryExchange (Array Data Exchange) 2-333
AryLRC_** (Calculate Array LRC Group) 2-512
AryMax (Array Maximum) 2-312
AryMean (Array Mean) 2-201
AryMin (Array Minimum) 2-312
AryMove (Array Move) 2-335
AryOr (Array Logical OR) 2-293
arySD (Array Element Standard Deviation) 2-203
arySearch (Array Search) 2-314
aryShiftReg (Shift Register) 2-352
aryShiftRegLR (Reversible Shift Register) 2-354
arySHL (Array N-element Left Shift) 2-357
arySHR (Array N-element Right Shift) 2-357
arySub (Array Subtraction) 2-197
arySubV (Array Value Subtraction) 2-199
AryToBCD (Array BCD Conversion) 2-227
AryToBin (Array Unsigned Integer Conversion) 2-229
AryToString (Array-to-Text String Conversion) 2-443
AryXor (Array Logical Exclusive OR) 2-293
AryXorN (Array Logical Exclusive NOR) 2-293
ASIN (Principal Arc Sine) 2-177
ATAN (Principal Arc Tangent) 2-177

B

Band (Deadband Control) 2-304
BCD Data Type-to-Unsigned Integer
Conversion Group 2-218
BCDsToBin
(Signed BCD-to-Signed Integer Conversion) 2-221
BCD_TO_** (BCD Data Type-to-Unsigned
Integer Conversion Group) 2-218
BCD-to-Unsigned Integer Conversion Group 2-212
Binary Code-to-Gray Code Conversion 2-438
Binary Selection 2-298
BinToBCDs_**
(Signed Integer-to-BCD Conversion Group) 2-224
BinToGray_**
(Binary Code-to-Gray Code Conversion) 2-438
Bit Counter 2-376
Bit Decoder 2-371
Bit Encoder 2-374
Bit Pattern Copy
(Bit String to Real Number) Group 2-341
Bit Pattern Copy
(Bit String to Signed Integer) Group 2-339
Bit Pattern Copy
(Real Number to Bit String) Group 2-347
Bit Pattern Copy
(Real Number to Signed Integer) Group 2-349
Bit Pattern Copy
(Signed Integer to Bit String) Group 2-343
Bit Pattern Copy
(Signed Integer to Real Number) Group 2-345
Bit Reversal 2-291
Bit String Conversion Group 2-279
Bit String-to-Bit String Conversion Group 2-242
Bit String-to-Integer Conversion Group 2-239
Bit String-to-Real Number Conversion Group 2-244
Bit String-to-Text String Conversion Group 2-255
BitCnt (Bit Counter) 2-376
Block Set 2-329
BREAK (Break Loop) 2-81
Break Down Date and Time 2-597
Break Loop 2-81
Broken Line Approximation 2-384
Byte Data Join Group 2-451
Byte Data Separation 2-449

C

Calculate Array CRC-16 2-516
Calculate Array CRC-CCITT 2-514
Calculate Array LRC Group 2-512
Calculate Text String CRC-16 2-510
Calculate Text String CRC-CCITT 2-508
Calculate Text String LRC 2-506
Case 2-28
CASE (Case) 2-28
Change File Name 2-852
Check for Leap Year 2-588
Check Subrange Variable 2-868
CheckReal (Real Number Check) 2-209
Checksum Calculation 2-504
ChkLeapYear (Check for Leap Year) 2-588
ChkRange (Check Subrange Variable) 2-868
CIPClose (Close CIP Class 3 Connection) 2-704
CIPOpen (Open CIP Class 3 Connection) 2-684
CIPRead (Read Variable Class 3 Explicit) 2-692
CIPSend (Send Explicit Message Class 3) 2-701
CIPUCMM Read (Read Variable UCMM Explicit) 2-706
CIPUCMM Send (Send Explicit Message UCMM) ... 2-716
CIPUCMM Write (Write Variable UCMM Explicit) 2-710
CIPWrite (Write Variable Class 3 Explicit) 2-696
Clear (Initialize) 2-337
Clear String 2-537
Clear TCP/UDP Socket Receive Buffer 2-789
ClearString (Clear String) 2-537
Close CIP Class 3 Connection 2-704
Close File 2-806
Close TCP/UDP Socket 2-786

Cmp (Compare) 2-98
ColmToLine_**
(Column to Line Conversion Group) 2-377
Column to Line Conversion Group 2-377
Combine Real Number Mantissa and Exponent 2-421
Compare 2-98
CONCAT (Concatenate String) 2-520
CONCAT_DATE _TOD
(Concatenate Date and Time of Day) 2-564
Concatenate Date and Time of Day 2-564
Concatenate String 2-520
Connect EtherCAT Slave 2-752
Connect TCP Socket 2-770
Conversion from Byte Array 2-458
Conversion to Byte Array 2-453
Convert Date and Time to Seconds 2-574
Convert Date to Seconds 2-576
Convert Days to Month 2-591
Convert Nanoseconds to Time 2-585
Convert Seconds to Date 2-580
Convert Seconds to Date and Time 2-578
Convert Seconds to Time 2-586
Convert Seconds to Time of Day 2-582
Convert Time of Day to Seconds 2-577
Convert Time to Nanoseconds 2-583
Convert Time to Seconds 2-584
Convert to Lowercase 2-538
Convert to Uppercase 2-538
Copy File 2-840
Copy**To*** (Bit Pattern Copy
(Bit String to Real Number) Group) 2-341
Copy**To*** (Bit Pattern Copy
(Real Number to Bit String) Group) 2-347
Copy**ToNum (Bit Pattern Copy
(Bit String to Signed Integer) Group) 2-339
Copy**ToNum (Bit Pattern Copy
(Real Number to Signed Integer) Group) 2-349
CopyNumTo** (Bit Pattern Copy
(Signed Integer to Bit String) Group) 2-343
CopyNumTo** (Bit Pattern Copy
(Signed Integer to Real Number) Group) 2-345
COS (Cosine in Radians) 2-174
Cosine in Radians 2-174
Create Directory 2-857
Create UDP Socket 2-754
Create User-defined Error 2-610
Create User-defined Information 2-639
CTD (Down-counter) 2-134
CTD_** (Down-counter Group) 2-136
CTU (Up-counter) 2-138
CTU_** (Up-counter Group) 2-140
CTUD (Up-down Counter) 2-142
CTUD_** (Up-down Counter Group) 2-146

D

Data Exchange 2-331
Data Trace Sampling 2-602
Data Trace Trigger 2-605
Date and Time-to-Text String Conversion 2-433
DateStructToDt (Join Time) 2-599
DateToSec (Convert Date to Seconds) 2-576
DateToString (Date-to-Text String Conversion) 2-435
Date-to-Text String Conversion 2-435
DaysToMonth (Convert Days to Month) 2-591
Dead Zone Control 2-307
Deadband Control 2-304
Dec (Decrement) 2-189
Decoder (Bit Decoder) 2-371
Decrement 2-189
Degrees to Radians 2-172
DegToRad (Degrees to Radians) 2-172
DELETE (Delete String) 2-531
Delete Directory 2-860
Delete File 2-848
Delete from Stack 2-480
Delete String 2-531
Determine Task Status 2-873
DirCreate (Create Directory) 2-857
DirRemove (Delete Directory) 2-860
Disconnect EtherCAT Slave 2-746
Dispart8Bit (Byte Data Separation) 2-449
DispartDigit (Four-bit Separation) 2-445
DispartReal (Separate Mantissa and Exponent) 2-418
DIV (Division) 2-166
Divide Time 2-562
Division 2-166
DIVTIME (Divide Time) 2-562
Down (Down Trigger) 2-40
Down Trigger 2-40
Down-counter 2-134
Down-counter Group 2-136
DT_TO_DATE (Extract Date from Date and Time) 2-568
DtToDateStruct (Break Down Date and Time) 2-597
DtToSec (Convert Date and Time to Seconds) 2-574
DtToString
(Date and Time-to-Text String Conversion) 2-433
DT_TO_TOD
(Extract Time of Day from Date and Time) 2-566

E

EC_CoESDORead (Read EtherCAT CoE SDO) 2-729
EC_CoESDOWrite (Write EtherCAT CoE SDO) 2-726
EC_ConnectSlave (Connect EtherCAT Slave) 2-752
EC_CopyMon (Transfer EtherCAT Packets) 2-744
EC_DisconnectSlave (Disconnect EtherCAT Slave) . 2-746
EC_SaveMon (Save EtherCAT Packets) 2-742
EC_StartMon (Start EtherCAT Packet Monitor) 2-734
EC_StopMon (Stop EtherCAT Packet Monitor) 2-740
Encoder (Bit Encoder) 2-374
End 2-60
End (End) 2-60
EQ (Equal) 2-84
EQascii (Text String Comparison Equal) 2-91
Equal 2-84
Exchange (Data Exchange) 2-331
ExecPMCR (Protocol Macro) 2-648

EXP (Natural Exponential Operation) 2-185
Exponentiation 2-187
EXPT (Exponentiation) 2-187
Extract Date from Date and Time 2-568
Extract Time of Day from Date and Time 2-566

F

FileClose (Close File) 2-806
FileCopy (Copy File) 2-840
FileGets (Get Text String) 2-826
FileOpen (Open File) 2-803
FilePuts (Put Text String) 2-833
FileRead (Read File) 2-812
FileReadVar (Read Variable from File) 2-799
FileRemove (Delete File) 2-848
FileRename (Change File Name) 2-852
FileSeek (Seek File) 2-809
FileWrite (Write File) 2-819
FileWriteVar (Write Variable to File) 2-794
FIND (Find String) 2-526
Find String 2-526
First In First Out 2-475
Fixed-decimal Number-to-Text String Conversion 2-428
Fixed-length Decimal Text String Conversion 2-423
Fixed-length Hexadecimal Text String Conversion 2-423
FixNumToString (Fixed-decimal Number-to-Text String Conversion) 2-428
FOR (Repeat Start) 2-76
Four-bit Join Group 2-447
Four-bit Separation 2-445
Fraction (Real Number Fraction) 2-207
F_TRIG (Down Trigger) 2-40

G

GE (Greater Than Or Equal) 2-88
GEascii (Text String Comparison Greater Than or Equal) 2-95
Get Byte Length 2-535
Get Clock Pulse Group 2-880
Get Days in Month 2-589
Get EtherCAT Error Status 2-637
Get EtherNet/IP Error Status 2-628
Get I/O Bus Error Status 2-626
Get Incrementing Free-running Counter Group 2-881
Get Motion Control Error Status 2-634
Get Number of Array Elements 2-463
Get Number of Records 2-497
Get PLC Controller Error Status 2-622
Get String Any 2-524
Get String Left 2-522
Get String Right 2-522
Get Text String 2-826
Get Time of Day 2-572
Get User-defined Error Status 2-617
Get**Clk (Get Clock Pulse Group) 2-880
Get**Cnt (Get Incrementing Free-running Counter Group) ... 2-881
GetAlarm (Get User-defined Error Status) 2-617
GetByteLen (Get Byte Length) 2-535
GetCJBError (Get I/O Bus Error Status) 2-626
GetDayOfWeek (Get Day of Week) 2-593
GetDaysOfMonth (Get Days in Month) 2-589
GetECError (Get EtherCAT Error Status) 2-637
GetEIPError (Get EtherNet/IP Error Status) 2-628
GetMCError (Get Motion Control Error Status) 2-634
GetMyTaskStatus (Read Current Task Status) 2-870
GetNTPStatus (Read NTP Status) 2-645
GetPLCError (Get PLC Controller Error Status) 2-622
GetTime (Get Time of Day) 2-572
GetTraceStatus (Read Data Trace Status) 2-607
GetWeekOfYear (Get Week Number) 2-595
Gray (Gray Code Conversion) 2-381
Gray Code Conversion 2-381
Gray Code-to-Binary Code Conversion Group 2-438
GrayToBin_** (Gray Code-to-Binary Code Conversion Group) 2-438
Greater Than 2-88
Greater Than Or Equal 2-88
GT (Greater Than) 2-88
GTascii (Text String Comparison Greater Than) 2-95

H

Hexadecimal Text String-to-Number Conversion Group 2-426
HexStringToNum_** (Hexadecimal Text String-to-Number Conversion Group) 2-426
Hundred-ms Timer 2-129

I

If 2-24
IF (If) 2-24
Inc (Increment) 2-189
Increment 2-189
Initialize 2-337
INSERT (Insert String) 2-533
Insert into Stack 2-478
Insert String 2-533
Integer Conversion Group 2-277
Integer-to-Bit String Conversion Group 2-235
Integer-to-Integer Conversion Group 2-232
Integer-to-Real Number Conversion Group 2-237
Integer-to-Text String Conversion Group 2-253

J

JMP (Jump) 2-74
Join Time 2-599
Jump 2-74

L

Last In First Out 2-475
LD (Load) 2-14
LDN (Load NOT) 2-14

LE (Less Than Or Equal) 2-88
LEascii (Text String Comparison Less Than or Equal) 2-95
LEFT (Get String Left) 2-522
LEN (String Length) 2-528
Less Than 2-88
Less Than Or Equal 2-88
LIMIT (Limiter) 2-302
Limiter 2-302
Line to Column Conversion 2-379
LineToColm (Line to Column Conversion) 2-379
LN (Natural Logarithm) 2-182
Load 2-14
Load NOT 2-14
Lock (Lock Tasks) 2-875
Lock Tasks 2-875
LOG (Logarithm Base 10) 2-182
Logarithm Base 10 2-182
Logical AND 2-286
Logical Exclusive OR 2-286
Logical OR 2-286
LrealToFormatString (LREAL-to-Formatted Text String) 2-264
LREAL-to-Formatted Text String 2-264
LT (Less Than) 2-88
LTascii (Text String Comparison Less Than) 2-95

M

Master Control End 2-62
Master Control Start 2-62
MAX (Maximum) 2-310
Maximum 2-310
Maximum Record Search 2-499
MC (Master Control Start) 2-62
MCR (Master Control End) 2-62
MemCopy (Memory Copy) 2-327
Memory Copy 2-327
MID (Get String Any) 2-524
MIN (Minimum) 2-310
Minimum 2-310
Minimum Record Search 2-499
MOD (Modulo-division) 2-168
ModReal (Real Number Modulo-division) 2-205
Modulo-division 2-168
Move 2-318
MOVE (Move) 2-318
Move Bit 2-321
Move Bits 2-325
Move Digit 2-323
MoveBit (Move Bit) 2-321
MoveDigit (Move Digit) 2-323
Moving Average 2-387
MovingAverage (Moving Average) 2-387
MUL (Multiplication) 2-161
MulOU (Multiplication with Overflow/Underflow Check) 2-163
MULTIME (Multiply Time) 2-560
Multiplexer 2-300
Multiplication 2-161
Multiplication with Overflow/Underflow Check 2-163
Multiply Time 2-560
MUX (Multiplexer) 2-300

N

NanoSecToTime (Convert Nanoseconds to Time) 2-585
Natural Exponential Operation 2-185
Natural Logarithm 2-182
N-bit Left Shift 2-360
N-bit Right Shift 2-360
NE (Not Equal) 2-86
NEascii (Text String Comparison Not Equal) 2-93
Neg (Reverse Sign) 2-369
NEXT (Repeat End) 2-76
NOT (Bit Reversal) 2-291
Not Equal 2-86
NSHLC (Shift N-bits Left with Carry) 2-362
NSHRC (Shift N-bits Right with Carry) 2-362
NumToDecString (Fixed-length Decimal Text String Conversion) 2-423
NumToHexString (Fixed-length Hexadecimal Text String Conversion) 2-423

O

Off-Delay Timer 2-120
On-Delay Timer 2-116
Open CIP Class 3 Connection 2-684
Open File 2-803
OR 2-18
OR (Logical OR) 2-286
OR (OR) 2-18
OR NOT 2-18
ORN (OR NOT) 2-18
Out (Output) 2-20
OutABit (Output A Bit) 2-57
OutNot (Output NOT) 2-20
Output 2-20
Output A Bit 2-57
Output NOT 2-20

P

PID Control with Autotuning 2-393
PIDAT (PID Control with Autotuning) 2-393
Principal Arc Cosine 2-177
Principal Arc Sine 2-177
Principal Arc Tangent 2-177
Protocol Macro 2-648
Push onto Stack 2-466
Put Text String 2-833
PWLApprox (Broken Line Approximation) 2-384

R

Radians to Degrees 2-172
RadToDeg (Radians to Degrees) 2-172

Rand (Random Number) 2-191
Random Number 2-191
Range Record Search 2-487
Read Current Task Status 2-870
Read Data Trace Status 2-607
Read EtherCAT CoE SDO 2-729
Read File 2-812
Read NTP Status 2-645
Read TCP Socket Status 2-783
Read Variable Class 3 Explicit 2-692
Read Variable from File 2-799
Read Variable UCMM Explicit 2-706
ReadNbit_** (N-bit Read Group) 2-864
Real Number Check 2-209
Real Number Conversion Group 2-281
Real Number Fraction 2-207
Real Number Modulo-division 2-205
Real Number-to-Bit String Conversion Group 2-249
Real Number-to-Integer Conversion Group 2-246
Real Number-to-Real Number Conversion Group 2-251
Real Number-to-Text String Conversion Group 2-257
RealToFormatString
(REAL-to-Formatted Text String) 2-259
REAL-to-Formatted Text String 2-259
RecMax (Maximum Record Search) 2-499
RecMin (Minimum Record Search) 2-499
RecNum (Get Number of Records) 2-497
Record Search 2-482
Record Sort 2-492
RecRangeSearch (Range Record Search) 2-487
RecSearch (Record Search) 2-482
RecSort (Record Sort) 2-492
Repeat 2-34
REPEAT (Repeat) 2-34
Repeat End 2-76
Repeat Start 2-76
REPLACE (Replace String) 2-529
Replace String 2-529
Reset 2-50
Reset (Reset) 2-50
Reset A Bit 2-55
Reset Bits 2-53
Reset EtherCAT Controller Error 2-636
Reset I/O Bus Error 2-624
Reset Motion Control Error 2-630
Reset PLC Controller Error 2-619
Reset User-defined Error 2-615
ResetABit (Reset A Bit) 2-55
ResetAlarm (Reset User-defined Error) 2-615
ResetBits (Reset Bits) 2-53
ResetCJBError (Reset I/O Bus Error) 2-624
ResetECError (Reset EtherCAT Controller Error) 2-636
ResetMCError (Reset Motion Control Error) 2-630
ResetPLCError (Reset PLC Controller Error) 2-619
Reset-Priority Keep 2-46
ResetUnit (Restart Unit) 2-641
Restart Unit 2-641
Return 2-61
RETURN (Return) 2-61
Reverse Sign 2-369
Reversible Shift Register 2-354
RIGHT (Get String Right) 2-522
ROL (Rotate N-bits Left) 2-364
ROR (Rotate N-bits Right) 2-364
Rotate N-bits Left 2-364
Rotate N-bits Right 2-364
Round (Round Off Real Number) 2-283
Round Off Real Number 2-283
Round Up Real Number 2-283
RoundUp (Round Up Real Number) 2-283
RS (Reset-Priority Keep) 2-46
R_TRIG (Up Trigger) 2-40

S

Save EtherCAT Packets 2-742
SCU Receive Serial 2-665
SCU Send Serial 2-658
SecToDate (Convert Seconds to Date) 2-580
SecToDt (Convert Seconds to Date and Time) 2-578
SecToTime (Convert Seconds to Time) 2-586
SecToTod (Convert Seconds to Time of Day) 2-582
Seek File 2-809
SEL (Binary Selection) 2-298
Send Command 2-674
Send Explicit Message Class 3 2-701
Send Explicit Message UCMM 2-716
SendCmd (Send Command) 2-674
Separate Mantissa and Exponent 2-418
SerialRcv (SCU Receive Serial) 2-665
SerialSend (SCU Send Serial) 2-658
Set 2-50
Set (Set) 2-50
Set A Bit 2-55
Set Bits 2-53
Set Time 2-570
SetABit (Set A Bit) 2-55
SetAlarm (Create User-defined Error) 2-610
SetBits (Set Bits) 2-53
SetBlock (Block Set) 2-329
SetInfo (Create User-defined Information) 2-639
Set-Priority Keep 2-48
SetTime (Set Time) 2-570
Shift N-bits Left with Carry 2-362
Shift N-bits Right with Carry 2-362
Shift Register 2-352
SHL (N-bit Left Shift) 2-360
SHR (N-bit Right Shift) 2-360
Signed BCD-to-Signed Integer Conversion 2-221
Signed Integer-to-BCD Conversion Group 2-224
SIN (Sine in Radians) 2-174
Sine in Radians 2-174
SizeOfAry (Get Number of Array Elements) 2-463
SktClearBuf
(Clear TCP/UDP Socket Receive Buffer) 2-789
SktClose (Close TCP/UDP Socket) 2-786
SktGetTCP Status (Read TCP Socket Status) 2-783
SktTCP Connect (Connect TCP Socket) 2-770

SktTCPAccept (Accept TCP Socket) 2-767
SktTCPRcv (TCP Socket Receive) 2-777
SktTCPSend (TCP Socket Send) 2-780
SktUDP Create (Create UDP Socket) 2-754
SktUDPRcv (UDP Socket Receive) 2-761
SktUDPSend (UDP Socket Send) 2-764
SQRT (Square Root) 2-180
Square Root 2-180
SR (Set-Priority Keep) 2-48
StackDel (Delete from Stack) 2-480
StackFIFO (First In First Out) 2-475
StackIns (Insert into Stack) 2-478
StackLIFO (Last In First Out) 2-475
StackPush (Push onto Stack) 2-466
Start EtherCAT Packet Monitor 2-734
Stop EtherCAT Packet Monitor 2-740
String Length 2-528
StringCRC16 (Calculate Text String CRC-16) 2-510
StringCRCCCITT
(Calculate Text String CRC-CCITT) 2-508
StringLRC (Calculate Text String LRC) 2-506
StringSum (Checksum Calculation) 2-504
STRING_TO_** (Text String-to-Bit String
Conversion Group) 2-272
STRING_TO_**
(Text String-to-Integer Conversion Group) 2-270
STRING_TO_**
(Text String-to-Real Number Conversion Group) 2-274
StringToAry (Text String-to-Array Conversion) 2-441
StringToFixNum (Text String-to-Fixed-decimal 2-430
SUB (Subtraction) 2-156
SUB_DATE_DATE (Subtract Date) 2-555
SUB_DT_DT (Subtract Date and Time) 2-556
SUB_DT_TIME
(Subtract Time from Date and Time) 2-558
SubOU
(Subtraction with Overflow/Underflow Check) 2-158
SUB_TIME (Subtract Time) 2-550
SUB_TOD_TIME
(Subtract Time from Time of Day) 2-552
SUB_TOD_TOD (Subtract Time of Day) 2-554
Subtract Date 2-555
Subtract Date and Time 2-556
Subtract Time 2-550
Subtract Time from Date and Time 2-558
Subtract Time from Time of Day 2-552
Subtract Time of Day 2-554
Subtraction 2-156
Subtraction with Overflow/Underflow Check 2-158
Swap (Swap Bytes) 2-368
Swap Bytes 2-368

T

Table Comparison 2-102
TableCmp (Table Comparison) 2-102
TAN (Tangent in Radians) 2-174
Tangent in Radians 2-174
Task_IsActive (Determine Task Status) 2-873
TCP Socket Receive 2-777
TCP Socket Send 2-780
Test A Bit 2-43
Test A Bit NOT 2-43
TestABit (Test A Bit) 2-43
TestABitN (Test A Bit NOT) 2-43
Text String Comparison Equal 2-91
Text String Comparison Greater Than 2-95
Text String Comparison Greater Than or Equal 2-95
Text String Comparison Less Than 2-95
Text String Comparison Less Than or Equal 2-95
Text String Comparison Not Equal 2-93
Text String-to-Array Conversion 2-441
Text String-to-Bit String Conversion Group 2-272
Text String-to-Fixed-decimal Conversion 2-430
Text String-to-Integer Conversion Group 2-270
Text String-to-Real Number Conversion Group 2-274
Time of Day-to-Text String Conversion 2-436
Timer (Hundred-ms Timer) 2-129
Timer Pulse 2-123
TimeToNanoSec (Convert Time to Nanoseconds) 2-583
TimeToSec (Convert Time to Seconds) 2-584
TO_** (Bit String Conversion Group) 2-279
TO_** (Integer Conversion Group) 2-277
TO_** (Real Number Conversion Group) 2-281
ToAryByte (Conversion to Byte Array) 2-453
TodToSec (Convert Time of Day to Seconds) 2-577
TodToString
(Time of Day-to-Text String Conversion) 2-436
TOF (Off-Delay Timer) 2-120
ToLCase (Convert to Lowercase) 2-538
TON (On-Delay Timer) 2-116
ToUCase (Convert to Uppercase) 2-538
TP (Timer Pulse) 2-123
TraceSamp (Data Trace Sampling) 2-602
TraceTrig (Data Trace Trigger) 2-605
TransBits (Move Bits) 2-325
Transfer EtherCAT Packets 2-744
Trim String Left 2-540
Trim String Right 2-540
TrimL (Trim String Left) 2-540
TrimR (Trim String Right) 2-540
TRUNC (Truncate) 2-283
Truncate 2-283

U

UDP Socket Receive 2-761
UDP Socket Send 2-764
Unite8Bit_** (Byte Data Join Group) 2-451
UniteDigit_** (Four-bit Join Group) 2-447
UniteReal
(Combine Real Number Mantissa and Exponent) .. 2-421
Unlock (Unlock Tasks) 2-875
Unlock Tasks 2-875
Unsigned Integer-to-BCD Conversion Group 2-215
Up (Up Trigger) 2-40
Up Trigger 2-40
Up-counter 2-138

Up-counter Group 2-140
Up-down Counter 2-142
Up-down Counter Group 2-146

W

While 2-32
WHILE (While) 2-32
Write EtherCAT CoE SDO 2-726
Write File 2-819
Write Variable Class 3 Explicit 2-696
Write Variable to File 2-794
Write Variable UCMM Explicit 2-710
WriteNbit_** (N-bit Write Group) 2-866

X

XOR (Logical Exclusive OR) 2-286
XORN (Logical Exclusive NOR) 2-289

Z

Zone (Dead Zone Control) 2-307
Zone Comparison 2-100
ZoneCmp (Zone Comparison) 2-100

OMRON Corporation **Industrial Automation Company**
Tokyo, JAPAN
Contact: www.ia.omron.com

Regional Headquarters
OMRON EUROPE B.V.
Wegalaan 67-69-2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/Fax: (31)2356-81-388

OMRON ELECTRONICS LLC
One Commerce Drive Schaumburg,
IL 60173-5302 U.S.A.
Tel: (1) 847-843-7900/Fax: (1) 847-843-7787

OMRON ASIA PACIFIC PTE. LTD.
No. 438A Alexandra Road # 05-05/08 (Lobby 2),
Alexandra Technopark,
Singapore 119967
Tel: (65) 6835-3011/Fax: (65) 6835-2711

OMRON (CHINA) CO., LTD.
Room 2211, Bank of China Tower,
200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-5037-2222/Fax: (86) 21-5037-2200

Authorized Distributor:

Cat. No. W502-E1-01 1107