KEYENCE

C60GB

Programable Controller KV-8000 Series

ST Language Programming Manual

Please read this Manual before use.
Keep this Manual in a safe place for later reference.

Chapter 1 OVERVIEW
Chapter 2 Creating ST
Chapter 3 Data Processing
Chapter 4 ST Statements and Precautions
Chapter 5 Basic Functions
Chapter 6 Application Functions
Chapter 7 Arithmetic Functions
Chapter 8 Extension Functions
· List of Control Relays/ Control Memories
· Index

Applicable CPU Unit
·KV-8000
·KV-8000A

Frequently Asked Questions
FAQ
www.keyence.com/controlfaq

Preface

This manual describes the specifications and setting methods of the functions required to write the ST Language program for "KV-8000 Series". Please read this manual carefully and fully understand it before writing programs.
When programming, refer to the user's manual of each unit such as "KV-8000 Series User's Manual", "KV-8000/7000/5000/3000/1000 Series & KV Nano Series Instruction Reference Manual", and "ST Language User's Manual" for CPU built-in function commands and expansion unit dedicated commands.
Please store this manual in a safe place where you can retrieve it whenever you need it,
and hand over it to the end user.

■KV-8000 Series related manuals

All of the following PDF manuals can be accessed through the ST Language help file. The latest version of PDF manuals can be downloaded from the Keyence web site.
To request a printed manual, please call the consultation telephone number on the back cover of this manual.

Series	Description	Bookbinding
KV-8000 Series User's Manual	Describes how to connect, and maintain the KV-8000 Series. It also describes the CPU built-in functions and how to write Ladder programs.	None (PDF only)
KV-8000/7000/5000/3000/1000 Series & KV Nano Series Instruction Reference Manual	Describes various instructions available for Ladder programs.	None (PDF only)
KV-8000 Series ST Language Programming Manual	This manual. Describes how to write ST Language programs, and available Operators, Control statements and Functions that can be used.	None (PDF only)
KV-8000/7000/5000/3000/1000 Series & KV Nano Series Script Programming Manual	Describes how to write Script programs, and available Operators, Control statements and Functions that can be used.	None (PDF only)
KV STUDIO User's Manual	Describes how to operate "KV STUDIO".	None (PDF only)

Safety Precautions

■Symbols

In this manual, in order to prevent personal injury and equipment damage, various items that need to be observed are classified according to the following symbols.

DANGER	**Failure to observe this item will result in death or serious injury.**
WARNING	**Failure to observe this item may result in death or serious injury.**
CAUTION	**Failure to observe this item may result in moderate or mild injury.**
NOTICE	**Failure to observe this item may cause product self-damage, and also other property damage.**

Important	**Notes on operations that must be performed.**
Point	**Notes on error-prone operations.**
Reference	**Useful information to help you understand the content of the text, etc.**

Pages need to be referred to, or reference pages in another manual.

■General Precautions

DANGER	• **Do not use this product for the purpose to protect the human body or parts of the human body.** • **This product is not suitable for explosion-proof areas. Do not use it in such a location.** • **Do not use this product in applications where the operating conditions of this product may cause serious injury to personnel and property (such as power plants, aviation, railways, ships, vehicles, medical equipment, entertainment equipment, etc.).**
WARNING	• **When using “ST Language”, ensure that the functions and performance of "KV Series" are operating normally before use.** • **Before starting work or operation, ensure that the functions and performance of this product are operating normally.** • **Establish a safety circuit that bypasses the PLC to ensure the safe operation of the entire system even if the PLC itself fails.** • **Output circuit or internal circuit malfunctions sometimes prevent normal control operation. Be sure to provide a safety circuit in control systems where its circuit malfunction may lead to fire or other serious accidents.** • **When installing this product, perform a risk assessment of the entire system to ensure that appropriate risk reduction is achieved. Please provide appropriate protective measures that bypass this product so that the entire system can operate safely even in abnormal situations.**
CAUTION	**Using this product in a manner not specified in this instruction manual may damage the protection attached to this product.**
NOTICE	• **Please note that KEYENCE cannot guarantee the functions and performance of the product in case of the product is not used in accordance with the specifications or if the product is modified.** • **When using this product in combination with other equipment, the functions and performance of the product may not satisfy you depending on the conditions of use and environment. Please consider carefully before using.**

KV-8000 CPU Function Version

The functions of the KV-8000 Series differ depending on the CPU function version.

The CPU function version can be checked from "Tools" → "Check CPU Function Version" → "PLC" in the ST Language menu. (*ST Language Ver.10.0 or later is required.)

The CPU function version can be updated from "Tools" → "Check CPU Function Version" → "PLC" →"Update System Program" in the ST Language menu. (*ST Language Ver.10.0 or later is required.)

■CPU function difference depending on CPU function version

Released		2021/09	2021/07	2021/04	2020/04	2019/04
CPU function version		2.3	2.2	2.1	2.0	1.0
ST Language compatible version		Ver.11.3 or later	Ver.11.2 or later	Ver.11.1 or later	Ver.11.0 or later	Ver.10.0 or later
CPU function	Device comment	Up to 128 characters	Up to 128 characters	Up to 128 characters	Up to 32 characters	Up to 32 characters
	Variable	○	○	○	○	-
	Array	Compatible with function block type	Compatible with STRING type	-	-	-
	Structure	○	○	○	○	-
	Function block	Compatible with automatic data type conversion	○	○	○ (supports EN/ ENO and Instance)	○
	ST Language	○	○	○	-	-
Function/ Script	Function block	○	○	○	2 functions added (FBCALL, FBSTRT)	No addition
	Data Size Get Function	○	○	○	2 functions added (WSIZE, ACNT)	-

Terminologies

This manual uses the following terminologies, with some exceptions.

Term	Description
Programmable Controller (PLC)	An electronic equipment allows to freely control the system by altering programs, and is also called PLC (Programmable Logic Controller).
"KV-8000"	PLC KV-8000/8000A made by Keyence.
KV-8000 Series	A generic term for expansion units that can be connected to "KV-8000" CPU unit.
CPU unit	"KV-8000".
KV-8000 Series Expansion unit	Expansion I/O units and special expansion units other than CPU units that can be used with the KV-8000 Series.
KV-8000/7000 Series Expansion unit	Expansion I/O units and special expansion units other than CPU units that can be used with the KV-8000/7000 Series.
KV-5000/3000/1000 Series Expansion unit	Expansion I/O units and special expansion units other than CPU units that can be used with the KV-5000/3000/1000 Series.
"ST Language"	Software that supports KV Series programming.
Ladder program	A program written with "ST Language".
Instruction Reference Manual	KV-8000/7000/5000/3000/1000 & KV Nano Series Instruction Reference Manual.

Symbols

In the Description of this manual, menus and buttons are represented by the following symbols.

Term	Description
16#	Values prefixed by "16#" indicate hexadecimal.
[DM0·DM1]	When calculating 32-bit numbers, two Devices are used. In the program, only the Device No. that stores lower 16-bit (DM0 in the example on the left) will be written. To prevent confusion, please write as shown in the example on the left to show that you are calculating 32-bit data.
↑	Indicates "the rising edge of a signal (when turned on) ".
⎍	Indicates "while the signal turns on".
↓	Indicates "the falling edge of a signal (when turned off) ".

Contents

Preface

Safety Precautions **1**

Terminologies 3
Symbols 3

Contents **4**

Chapter 1 Overview

ST Types **1-2**

Box ST 1-2
Area ST 1-3

Functions of ST **1-4**

About Assignment statement 1-4
About Operator 1-4
About Control statement 1-5

Chapter 2 Creating ST

Creating ST **2-2**

ST program 2-2

Box ST **2-4**

Features of Box ST 2-4
Box ST creating 2-4
Precautions on using Box ST 2-9

Area ST **2-11**

Features of Area ST 2-11
Area ST creating 2-11

Variable Programming and Arrays/Structures **2-15**

Variable programming 2-15
Global/Local variables 2-15
Variable/Data type list 2-15
Array 2-16
Structure 2-18
Variable setting 2-20
Precautions on using Variables 2-22

Precautions on ST Writing **2-23**

About Parallel connection of Area ST 2-23
Precautions on Writing Operation expression in ST 2-23

ST Conversion Messages **2-24**

Warning message 2-24
Error message 2-24

Function Block **2-27**

Calling Function block 2-27

Debug function of ST **2-28**

Monitoring function 2-28

Input Auxiliary Function and Others **2-30**

About Input auxiliary function 2-30
About Comment function 2-34
Precautions on Edit window and Customization 2-34
About ST/Script work memory (Work device) 2-35
XYM notation 2-35

Chapter 3 Data Processing

Devices **3-2**
Device list 3-2
Device notation 3-5
Bit device 3-7
Word device 3-13
Processing Bit device in word units 3-19
Processing Word device as Bit device 3-20
Direct I/O Relay 3-21
Variable 3-21
Constant 3-22
Index modify and Indirect designation **3-25**
Index modify 3-25
Indirect designation 3-27
Combination of Index modify and Indirect designation 3-29
Data Types (Suffixes) **3-31**
Suffix 3-31
About Operation type 3-36

Chapter 4 ST Statements and Precautions

Assignment statements **4-2**
How to use Assignment statements 4-2
Operators **4-4**
Operator list 4-4
Control Statements **4-6**
Control Statement List 4-6
IF ~ statement (Condition branch ①) 4-7
IF ~ ELSE statement (Condition branch ②) 4-8
IF ~ ELS IF ~ statement (Condition branch ③) 4-9
CASE ~ statement (Multi-branch control) 4-11
FOR ~ END_FOR statement (Frequency Specified Loop Control) 4-12
WHILE statement (Pre-judge Loop Control) 4-13
REPEAT statement (Post-judge Loop Control) 4-14
EXIT Processing 4-15
RETURN Processing 4-15
Precautions on Using Control Statement 4-16
Precautions on ST Programming **4-18**

Chapter 5 Basic Functions

How to Read Functions **5-2**
BMOV 5-2
Contact Function **5-3**
LDP 5-3
LDF 5-3
LDPB 5-4
LDFB 5-5
BLD 5-5
Bit Contact Function **5-7**
BLDB 5-7
Output Function **5-8**
SET 5-8
RES 5-8

Bit Output Function **5-10**
BOUT 5-10
BOUB 5-10
BSET 5-11
BRES 5-12
Timer/Counter Function **5-13**
TMR 5-13
TMH 5-14
TMS 5-15
TMU 5-15
CNT 5-16
CURVAL 5-17
Direct I/O Function **5-18**
RFSX 5-18
RFSY 5-18

Chapter 6 Application Functions

Flow Function **6-2**
CALL 6-2
ECALL 6-2
Function Block Function **6-4**
FBSTRT 6-4
FEND 6-5
MDSTRT 6-5
MDSTOP 6-6
Index Register Function **6-7**
ZPUSH 6-7
ZPOP 6-8
Indirect Designation Function **6-9**
ADRSET 6-9
ADRINC 6-9
ADRDEC 6-10
ADRADD 6-11
ADRSUB 6-12
File Register Function **6-13**
FRSET 6-13
FRSTM 6-13
FRLDM 6-14
Unit Program Function **6-15**
UPSTRT 6-15
UPSTOP 6-16
UFSUS 6-17
UFRSM 6-18
UMALLOC 6-20
Buffer Memory Function **6-21**
UREAD 6-21
UWRIT 6-21
UFILL 6-22
Unit Type Identification Function **6-23**
IS_ □□□ 6-23
Array Element Amount Get Function **6-24**
ACNT 6-24
Word Size Get Function **6-25**
WSIZE 6-25

Chapter 7 Arithmetic Functions

Data Transfer Function **7-2**

BMOV 7-2
FMOV 7-2
PMOV 7-3
BYBMOV 7-4
BYLMOV 7-5

Arithmetic Function 7-6

INC 7-6
DEC 7-7
ROOT 7-7
SQRT 7-8

Logical Operation Function 7-9

ANDA 7-9
ORA 7-9
EORA 7-10
ENRA 7-11
COM 7-12
NEG 7-12
ABS 7-13

Data Shift Function 7-14

SHR 7-14
SHL 7-14
ASRA 7-15
ASLA 7-16
RRA 7-16
RLA 7-17
ROR 7-17
ROL 7-18
WSR 7-19
WSL 7-19
BSR 7-20
BSL 7-21

Data Control Function 7-22

LIMIT 7-22
BANDC 7-22
ZONE 7-23
APR 7-24
RAMP 7-25
TPOUT 7-26
LLFLT 7-27

Data Convert Function 7-29

TBCD 7-29
TBIN 7-29
MPX 7-30
DMX 7-30
GRY 7-31
RGRY 7-32
DISN 7-32
UNIN 7-33
DISB 7-34
UNIB 7-34
SWAP 7-35
BSWAP 7-35
DECO 7-36
ENCO 7-37
CPMSET 7-37
CPMGET 7-38
*_TO_** 7-39
TO_** 7-40
*_TRUNC_** 7-40
TRUNC_** 7-41
*_TO_BCD_** 7-42
TO_BCD_** 7-42
*_BCD_TO_** 7-43
BCD_TO_** 7-43

Floating Point Function 7-44

FLOAT 7-44

DFLOAT 7-44
INTG 7-45
INT 7-46
DISF 7-46
UNIF 7-47
EXP 7-48
EXPT 7-48
LN 7-49
LOG 7-50
RAD 7-50
DEG 7-51
SIN 7-52
COS 7-53
TAN 7-54
ASIN 7-55
ACOS 7-55
ATAN 7-56
ATAN2 7-57

Character Process Function 7-59

ASC 7-59
RASC 7-59
DASC 7-60
STR 7-61
RDASC 7-62
HASC 7-63
RHASC 7-64
FASC 7-65
RFASC 7-66
VAL 7-67
LEN 7-67
SRGHT 7-68
RIGHT 7-69
SLEFT 7-69
LEFT 7-70
SMID 7-71
MID 7-72
SRPLC 7-72
REPLACE 7-73
SINS 7-74
INSERT 7-75
SDEL 7-75
DELETE 7-76
TRIM 7-77
LTRIM 7-78
RTRIM 7-78
STRIM 7-79
SPLIT 7-80
SFIND 7-81
SFINDN 7-82
INSTR 7-82
FIND 7-83
DISS 7-84
UNIS 7-85
CHR 7-85
CPSASC 7-86
RCPSASC 7-86
CONCAT 7-87

Chapter 8 Extension Functions

Data Process Function 8-2

SEG 8-2
BCNT 8-3
DCNT 8-3
SER 8-4
DSER 8-5
MAX 8-5
MAX_RANGE 8-6
MIN 8-7

MIN_RANGE 8-7
SEL 8-8
MUX 8-8
AVG 8-9
WSUM 8-10
BSUM 8-10
CRC 8-11
ZRES 8-12
BCMP 8-13
BCMPI 8-14
RND 8-14
SORT 8-15
SORTN 8-16

Table Process Function 8-17

FIFOW 8-17
FIFOR 8-17
LIFOW 8-18
LIFOR 8-19
FWRIT 8-19
FINS 8-20
FDEL 8-21

Time Process Function 8-22

WTIME 8-22
SEC 8-22
RSEC 8-23
AJST 8-24

High-speed Process Function 8-25

DI 8-25
EI 8-25
DIC 8-26
IEDGE 8-26

PID Function 8-28

PIDAT 8-28

Log Function 8-29

LOGE 8-29
LOGD 8-29
TRGD 8-30
TRGR 8-30

Storage Function 8-32

MWRIT 8-32
MREAD 8-33
MFREE 8-34
MFREEK 8-36
MMKDIR 8-37
MRMDIR 8-38
MDEL 8-39
MPRINT 8-40
MREADL 8-41
MCOPY 8-42
MMOV 8-43
MREN 8-44
MSTAT 8-44

Access Window Function 8-46

AWSHOW 8-46
AWHIDE 8-46

FREE RUN Counter Function 8-48

RFSFRC 8-48

Sensor Setting Function 8-49

SPRD 8-49
SPWR 8-50
SSVC 8-51

Cyclic Communication Refresh Function **8-53**

RFSCI 8-53
RFSCO 8-53

Appendix

CR/CM List **A-2**

Control relay (CR) A-2
Control memory CM A-4

ASCII Code Table **A-10**

List of Characters That Cannot Be Used **A-11**

Unusable Function List **A-13**

List of Functions supported as Standard Functions **A-14**

Index **A-15**

Functions Index **A-17**

1

Overview

ST Types 1-2
Functions of ST 1-4

ST Types

Two types of ST with different programming styles are available in ST Language.

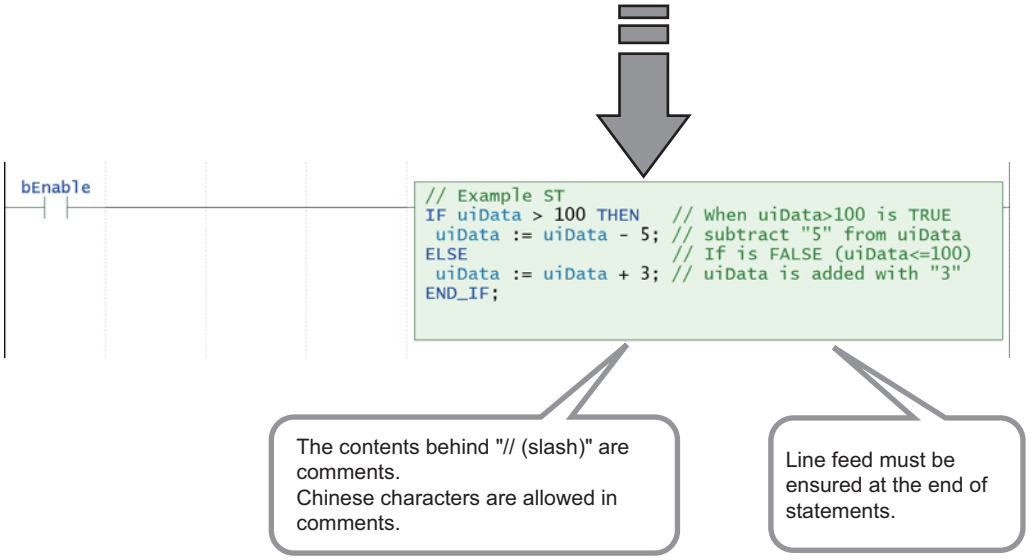
Box ST

Box ST refers to a ST to program execution conditions with Ladder.
The programming area on Edition screen is within 6 cells from the 5th cell to the 10th cell in one row. Characters for programming can be written without conversion.

Program ST

```
// Example ST
IF uiData > 100 THEN          // When uiData>100 is TRUE
    uiData := uiData - 5;     // subtract "5" from uiData
ELSE                          // If is FALSE (uiData<=100)
    uiData := uiData + 3;     // uiData is added with "3"
END_IF;
```

Box ST



Point **When execution condition is not met, the program inside Box ST is not executed and will be skipped. (please pay attention to partial Functions such as differential instructions, etc.).**

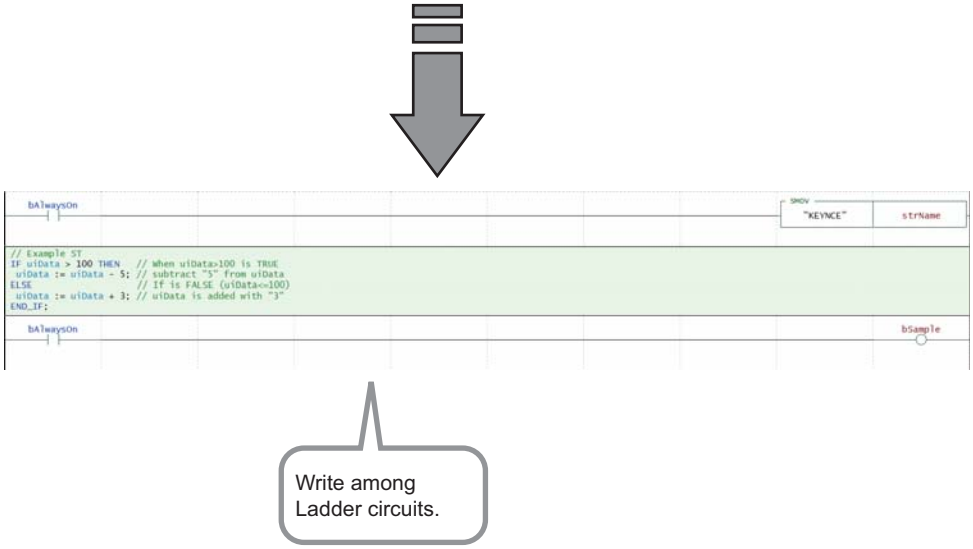
Area ST

Area type ST refers to a scan execution type ST that does not have Ladder execution conditions.
The programming area on Edition screen occupies the entire line.

Program ST

```
// Example ST
IF uiData > 100 THEN          // When uiData>100 is TRUE
    uiData := uiData - 5;     // subtract "5" from uiData
ELSE                          // If is FALSE (uiData<=100)
    uiData := uiData + 3;     // uiData is added with "3"
END_IF;
```

Area ST



Functions of ST

By ST, the process for string values and operation results in Devices and Variables can be written using Assignment statements.

About Assignment statement

By ST, the process for string values and operation results in Devices and Variables can be written using Assignment statements.

■Assignment statement

uiValue1 := uiValue2 + 10;

The result (value) on the right side is stored on the left side of ":=".

●Numerical value assignment

To store constant 1000 into uiValue.

Program ST

```
uiValue := 1000;
```

●String assignment

To store string "KEYENCE" into strName (NUL (00H) indicating the end of string will be automatically added).

Program ST

```
strName := 'KEYENCE';
```

●Bit assignment

To map the state of BOOL variable (bValue1) to bValue2.

Program ST

```
bValue2 := bValue1;
```

About Operator

By ST, more intuitive programming is possible because it is not necessary to convert Arithmetic operation and Logical operation to dedicated instruction such as Ladder program, and Operators can be programed directly.

●Operation example

To multiply uiValue1 value by 10, and double the result of adding uiValue2. To assign (store) operation result to uiValue3.

Program ST

```
uiValue3 := 2 * (uiValue1 * 10 + uiValue2);
```

● Operator list

Sign		Processing content	Programming example
Arithmetic operator	**	Calculates the power	uiValue1 := uiValue2 ** 2;
	*	Calculates product of 2 values (multiplication)	uiValue1 := uiValue2 * 3;
	/	Calculates quotient of 2 values (division)	uiValue1 := uiValue2/4;
	MOD	Divides 2 values, returns a remainder	uiValue1 := uiValue2 MOD 5;
	+	Calculates the sum of 2 values (addition)	uiValue1 := uiValue2 + 6;
	-	Calculates the difference of 2 values (subtraction)	uiValue1 := uiValue2 - 7;
Comparison operator	<	Less than	bFlag := uiValue < 10;
	<=	Less than or equal to	bFlag := uiValue <= 20;
	>	Larger than	bFlag := uiValue > 30;
	>=	Larger than or equal to	bFlag := uiValue >= 40;
	=	Equal to (equivalent)	bFlag := uiValue = 50;
	<>	Unequal to (unequal)	bFlag := uiValue <> 60;
Logic operator	NOT	Calculates Logical negation of a value	bFlag1 := NOT bFlag2;
	AND, &	Calculates Logical conjunction of 2 values (for Bit data only, or Word data only)	bFlag1 := bFlag2 AND bFlag3;
	OR	Calculates Logical disjunction of 2 values (for Bit data only, or Word data only)	bFlag1 := bFlag2 OR bFlag3;
	XOR	Calculates Exclusive disjunction of 2 values (for Bit data only, or Word data only)	bFlag1 := bFlag2 XOR bFlag3;

For details on Operator, refer to "Operators" (page 4-4)

About Control statement

You can write various Control statements by ST. By using Control statements, you can easily change (control) the procedure of the program. Two types of Control statements are available: the Condition branch statement that branches processing according to conditions, and the Loop control statement that performs repetitive processing.

■Condition branch statement

Selects the process to be executed according to the conditions.

● Condition branch statement types

Control statement	Basic expression	Description
IF statement	IF ␣<Condition expression>␣THEN <Process statement >; END_IF;	Branch Control statement that determines whether to execute processing according to the result of <Condition expression>
	IF ␣<Condition expression>␣THEN <Process statement 1>; ELSE ␣ <Process statement 2>; END_IF;	Two-Branch Control statement that executes separate processing according to the result of <Condition expression>

	IF ␣<Condition expression1>␣THEN
	  <Process statement 1>;
	ELSIF ␣<Condition expression 2>␣THEN
	  <Process statement 2>;
	  .
	  .
	ELSE
	  <Process statement n>;
	END_IF;
		Multi-branch Control statement that executes different processing according to the results of multiple condition expressions (<Condition expression 1>, <Condition expression 2>...<Condition expression n>)

CASE statement	CASE ␣<Word data>␣OF <Comparison value 1>, <Comparison value 2>:<Process statement 1>; <Comparison value 3>:<Process statement 2>; ELSE <Process statement 3>; END_CASE;	Multi-branch control statement that selects and executes the corresponding process according to the value of <Word data>

* ␣indicates inserting 1-byte space.

■Loop control statement

Repeats the process under the specified conditions.

● Loop control statement types

Control statement	Basic expression	Description
FOR statement	FOR ␣<Word data>:=<default> ␣TO ␣<final value> ␣STEP ␣<increment> ␣DO <Process statement >; END_FOR;	Control statement that repeats processing until <Word data>= <final value>
WHILE statement	WHILE ␣<Condition expression> ␣DO <Process statement >; END ␣WHILE;	Control statement that repeats processing (unlimited times) while the Condition expression is satisfied (true)
REPEAT statement	REPEAT <Process statement >; UNTIL ␣<Condition expression> END_REPEAT;	Control statement that repeats processing (unlimited times) while the Condition expression is not satisfied (false)
EXIT statement	<Loop condition statement starts>; EXIT; <Loop condition statement ends>;	Interruptible Loop control statement regardless of the condition (only for FOR statement, WHILE statement and REPEAT statement)

* ␣indicates inserting 1-byte space.
* For details on Control statement, refer to "Control Statements" (page 4-6)

■Program execution end statement

Ends the execution of the written program.

●Program execution end statement types

Control statement	Basic expression	Description
RETURN statement	<Process statement>; RETURN; <Process statement >;	Ends the execution of the written program

About Suffix

In the Ladder language, the type of instruction operation is determined by adding a suffix to each instruction, whereas in the ST language, the type of function operation and the value stored in the device are determined by adding a suffix to the device. For more information on data types (suffixes), refer to 「Data Types (Suffixes)」(page 3-31).

●Omitting suffix

When writing a Device without adding a suffix, the type depends on the Device type.

Device type	Type processed when suffix is omitted (default type)
R(DR),B,MR,LR,T,C,CTC,CR,UR	.B (Bit type)
DM,W,EM,ZF,FM,TM,CM,V,P,UM,UV, Devices specified indirectly(*)	.U (unsigned 16-bit data type)
Z	.L (signed 32-bit data)

●Variable suffix

When writing Global variables or Local variables, the type processed will be determined according to the "data type" selected when registering the Variable.

Variable data type		Corresponding data type
BOOL	Bit	.B
INT	1-word signed integer	.S
DINT	2-word signed integer	.L
UINT	1-word unsigned integer	.U
UDINT	2-word unsigned integer	.D
REAL	Single precision floating point real number	.F
LREAL	Double precision floating point real number	.DF
STRING	String	.T
Timer	Timer	.B
COUNTER	COUNTER	.B
ARRAY		Default data type specified when registering Variables
Structure		Default data type of each member set in the Structure definition
Function block		Default data type of each Argument set in the Argument setting

About ST Function

The ST Function allows the Ladder instructions of KV-8000 Series available via ST.
To discriminate from Ladder instructions, instructions used for ST are called "ST Function".

● ST Function types

ST Functions can be broadly divided into the following types.

Type	Description
Operation Process Function	Function group for operating Word data and Bit data, including ANDA/ORA that performs Logical operation, BMOV/FMOV that performs data transfer, and SIN/COS that performs floating-point type operation.
String Process Function	Function group for string processing, including LEN for testing string length and SMID/SLEFT for extracting string, etc.
Data Process Function	Function group for data processing, including MAX to find the maximum value of data, MIN to find the minimum value, and BCNT to count the number of ON bits of the specified Word data.
Other Functions	Function group for processing those other than Operation Process/String Process/Data Process, including Functions such as TMR to operate Timers, ADRSET to indirectly specify, and MWRIT/MREAD to execute the memory card control.

■About the operation flags (CR2009 to CR2012)

Each ST will be converted into a Ladder program, and its operation flag will change accordingly when the Function is executed. Therefore, before and after the execution of the ST program, the state of the operation flag will change.
In addition, calculation flags that are changed with ST programs are basically impossible to use because Ladder programs generated by ST conversion can be written in various ways such as combination functions.

■Precautions on Indirect designation and Index modify

When using Indirect designation, Index modify, etc., be careful not to specify Devices or work areas outside the range.
If you try to use Indirect designation or Index modify to access an out-of-range Device or work area, the following situation will occur respectively.

● Function operation list when invalid address is specified

Function operation	Function name
0 is stored in the Return value	LDP, LDF, BLD, BLDB, LDPB, LDFB, UNIF, RDASC, RHASC, RFASC, VAL, LEN, SFIND, INSTR, FIND, SEG, MAX_RANGE, MIN_RANGE, AVG, WSUM, BSUM, BCMP, BCMPI, SEC
NUL (00H) is stored in the Return value	DASC, STR, HASC, FASC, SRGHT, RIGHT, SLEFT, LEFT, SMID, MID, SRPLC, REPLACE, SINS, INSERT, SDEL, DELETE
Value of 1st Argument is stored in the Return value *1	ROOT, ANDA, ORA, EORA, ENRA, COM, NEG, SHR、SHL, RRA, RLA, ROR, ROL, TBCD, TBIN, MPX, DMX, GRY, RGRY, SWAP, DECO, ENCO, TO,USDLF, FLOAT, INTG, INT, EXP, LOG, RAD, DEG, SIN, COS, TAN, ASIN, ACOS, ATAN, ASC, RASC, DASC, CHR, BCNT, DCNT, SER, CRC, WTIME, AWNUM, AWMSG, WTIME, SQRT, ABS, RND
The Function is not executed	SET, RES, BOUT, BOUB, BSET, BRES, ZPUSH, ZPOP, ADRSET, ADRINC, ADRDEC, ADRADD, ADRSUB, BMOV, FMOV, PMOV, INC, DEC, WSR, WSL, BSR, BSL, DISF, DISS, UNIS, ZRES, FIFOW, FIFOR, LIFOW, LIFOR, FWRIT, FINS, FDEL, RSEC, DIC, LOGE, LOGD, MWRIT, MREAD, MFREE, MMKDIR, RFSX, RFSY, ECALL, MDSTRT, MDSTOP, FRSET, FRSTM, FRLDM, UREAD, UWRIT, UFILL, LIMIT, BANDC, ZONE, APR, DISN, DISB, UNIN, UNIB, DSER, RFSCTH, TRGD, MRMDIR, MDEL, RFSFRC、ACNT

*1 However, if invalid indirect reference is performed in the 1st Argument, the correctness of Return value cannot be guaranteed.

2

Creating ST

Creating ST 2-2
Box ST 2-4
Area ST. 2-11
Variable Programming and Arrays/Structures 2-15
Precautions on ST Writing 2-23
ST Conversion Messages. 2-24
Function Block 2-27
Debug function of ST 2-28
Input Auxiliary Function and Others 2-30

Creating ST

This chapter describes how to operate Box ST/Area ST as well as Menu and Edition screens.

ST program

How to write ST

Box ST/Area ST can be written by combining with the Ladder program on the Ladder Editor window.
ST can be written after checking ST programming area on the Ladder Editor window.

① Call from Menu

Select “ST/Script (S)" → "Insert Area ST (R)" or "Insert Box ST (T)".

② Call from the right-click menu

When the cursor locates on the Ladder Editor window, select "Edit ST/Script (X)" from the right-click menu, and then select "Insert Area ST (R)" or "Insert Box ST (T)".

③ Call from icons

Click Icon on the Tool bar.

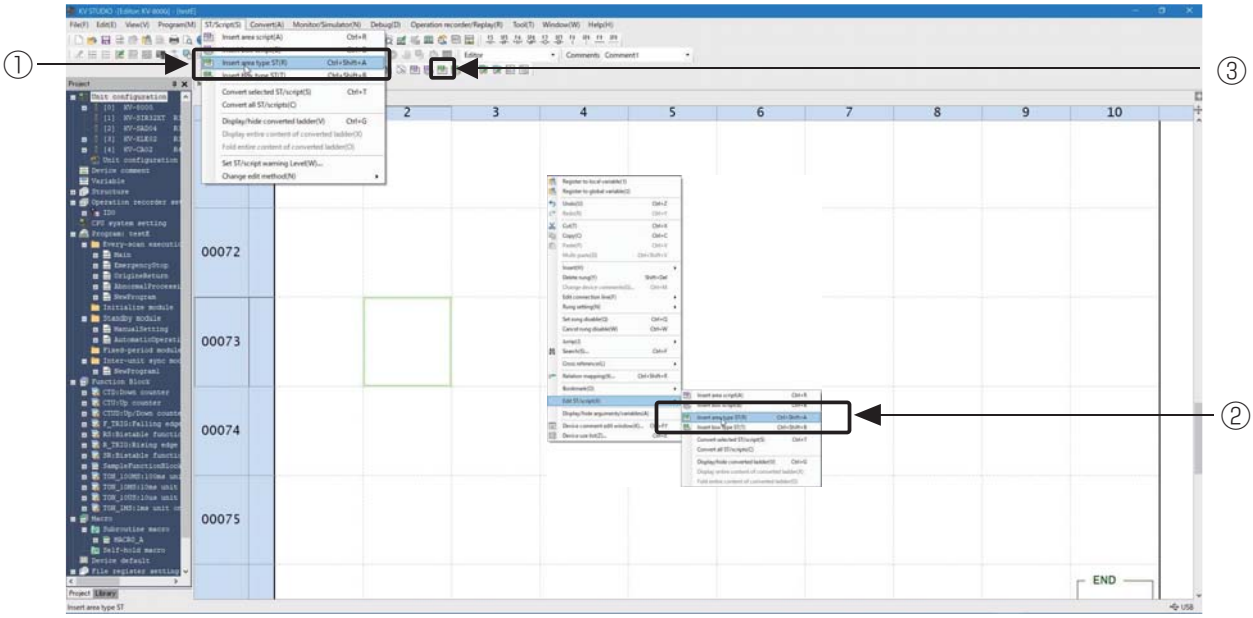
·Area ST:

·Box ST:

④ Call using the shortcut key

·Area ST: Ctrl Shift key + A key

·Box ST: Ctrl Shift Key + B key

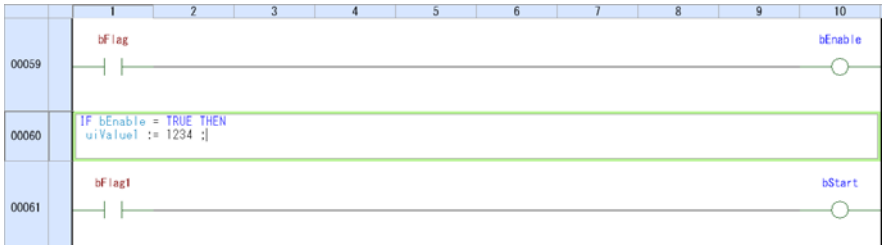


How to input ST

There are two input methods for ST. But the function will not vary with different input methods. So, please select as you like.

■Direct input (default)

Program directly on the Ladder Editor in the same way as the Ladder program instruction.

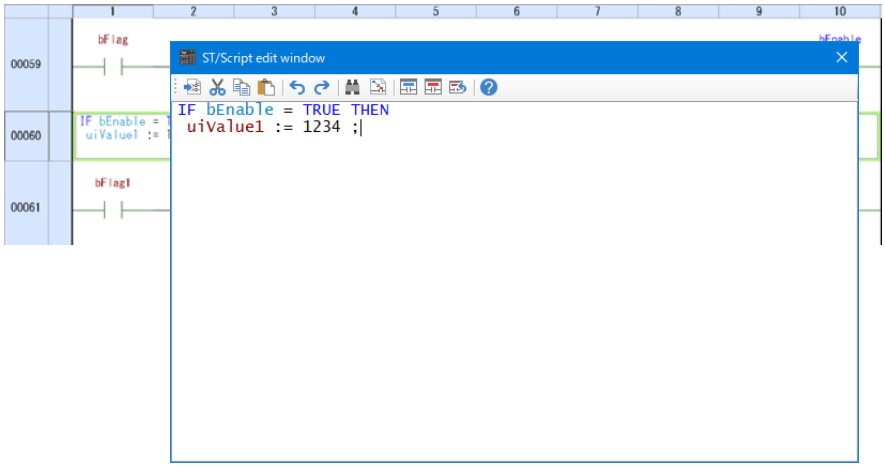


From Menu, select "ST/Script (S)" → "Change edition method (N)" → "Direct input (D)".

Other way From Menu, select "Tool (T)" → "Option (O)", and then "Window" in "Input method" of "ST/Script setting" tab.

■Window input

Input in the Text Editor window displayed during ST programming.



From Menu, select "ST/Script (S)" → "Change edit method(N)" → "Window(W)".

Other way From Menu, select "Tool (T)" → "Option (O)", and then "Window" in "Input method" of "ST/Script setting" tab.

Box ST

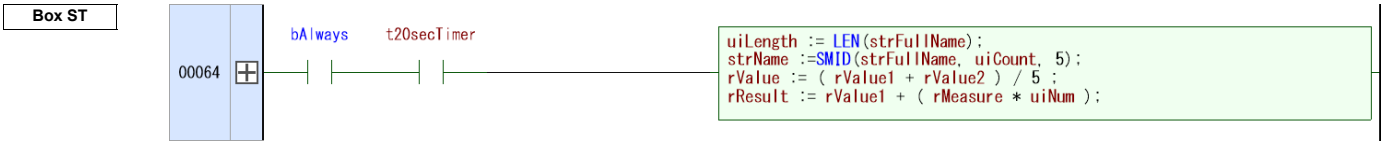
This section describes precautions on how to make/edit and use the Box ST.

Features of Box ST

Box ST refers to a ST with Ladder execution conditions that allows you to specify the executing timing of the program written in ST by using a Ladder.

Area ST is used in the following cases

Box ST is suitable for writing programs with a few lines of Operation expression or without complex Control statements.
In addition, by writing the execution conditions in Ladder, the program in the Box ST can be used easily without using the Control statement, because it can be written only with the execution Process statements.

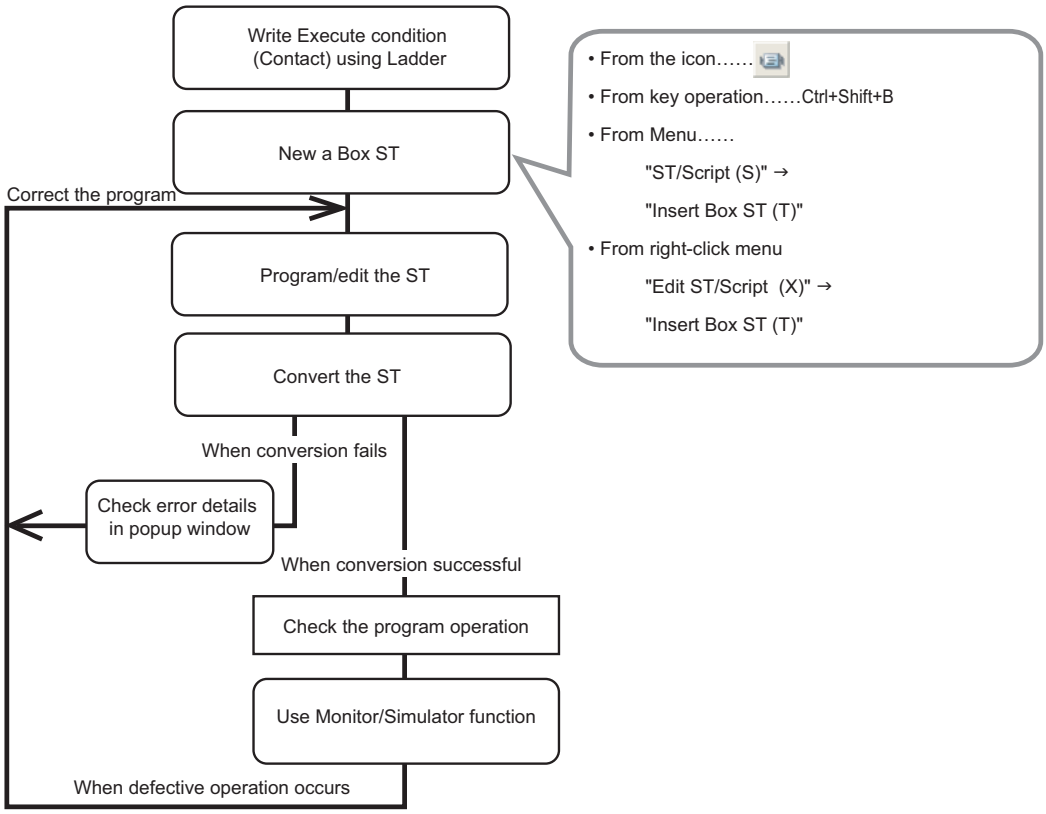


Box ST creating

The procedure from Box ST creating to debugging is described as follows.

Operation procedure

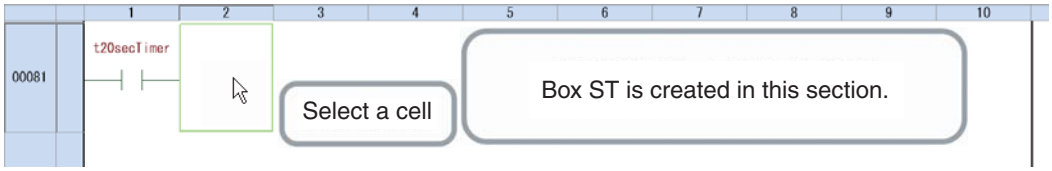
The operation procedure for creating/editing Box ST is described as follows.



Box ST newing

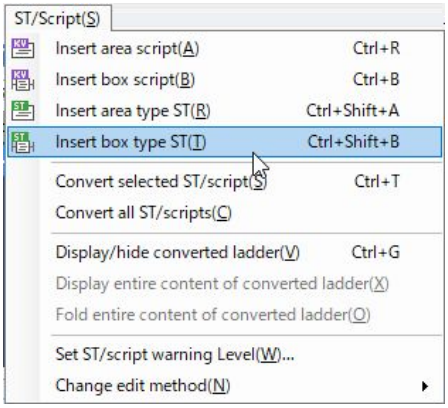
The newing steps for Box ST are described as follows.

1 Write the execution conditions in the Ladder Editor, and click the cell where you want to create Box ST.



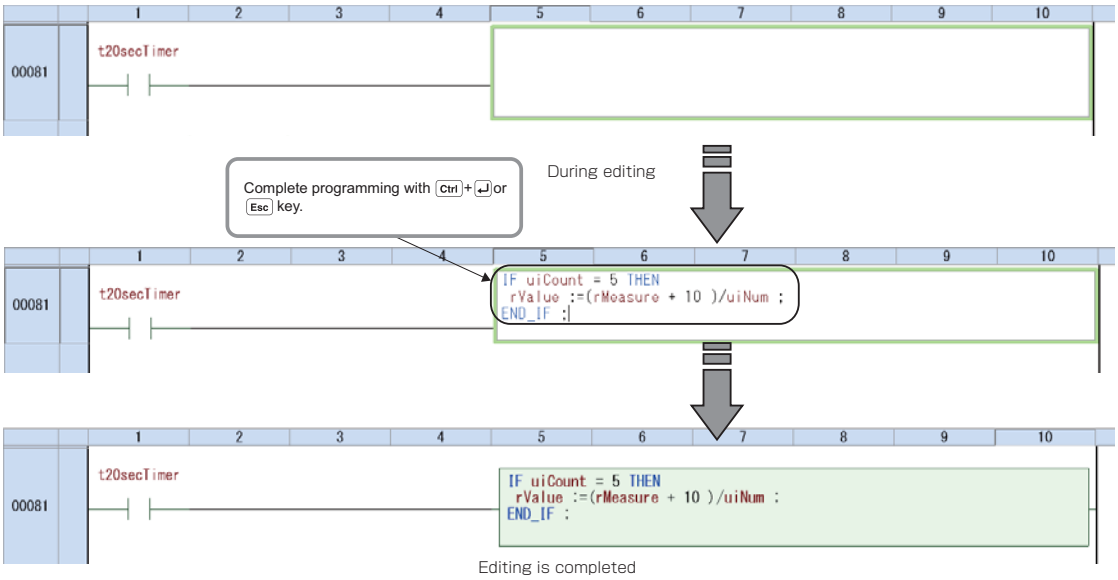
Reference
- Box ST occupies 6cells (5 to 10). Execution conditions can be written within cell 1 to cell 4.
- Regardless of the position of the selected cell, Box ST will be created in the cell 5 to cell 10.

2 **From Menu, select "ST/Script (S)" → "Insert Box ST (T)".**



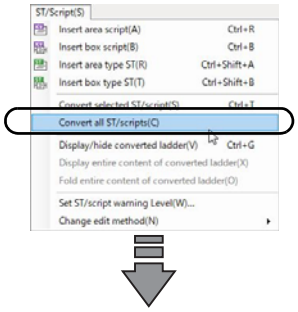
Other way
- Ctrl key + Shift key + B key
- From the right-click menu, select "Edit ST/Script (X)" → "Insert Box ST (T)"

3 **Box ST will be inserted. Write the program.**
After writing, press Ctrl key + ↵ key, or Esc key, or click the Ladder Editor window other than which Box ST is being edited to complete writing.

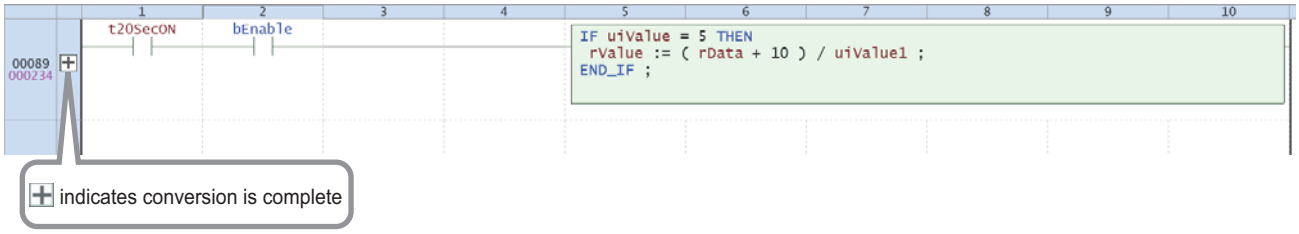


4 After writing, convert Box ST.

Other way From right-click menu, select "Edit ST/Script (X)" → "Convert all STs/Scripts (C)"



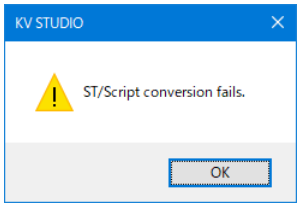
● When conversion successful



● When conversion fails

"ST conversion error" will be displayed.
Also, an error message will be displayed in the pop-up window.

"When conversion error occurs"(page 2-24)



Box ST reediting

The re-editing steps for Box ST are described as follows.

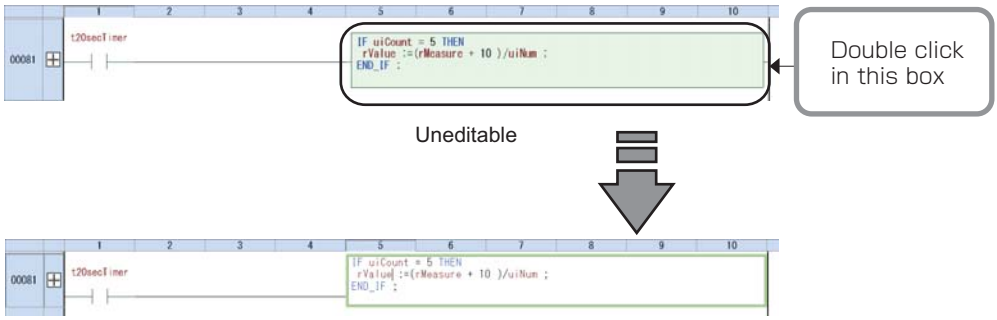
■Reediting Box ST

1 To re-edit Box ST that has been written, double-click the Box ST to be edited.

Other way

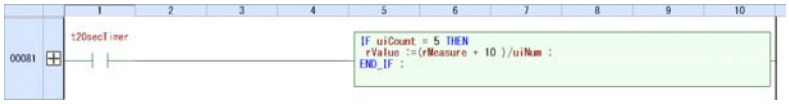
After selecting the Box ST to be edited, click the following keys:

- Esc key
- F2 key
- ↵ key

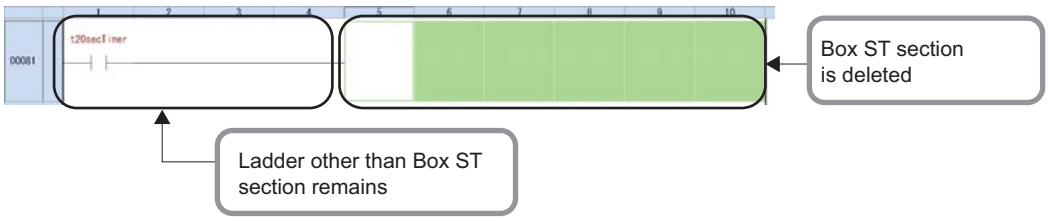


■Deleting Box ST

1 Select the Box ST to be deleted.

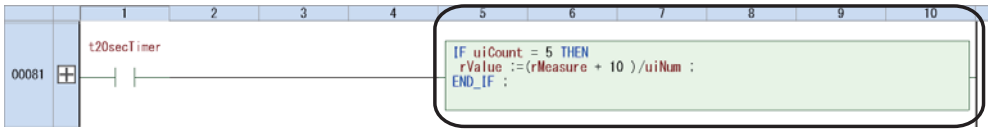


2 **Press Delete key on the keyboard.**



■Copying Box ST

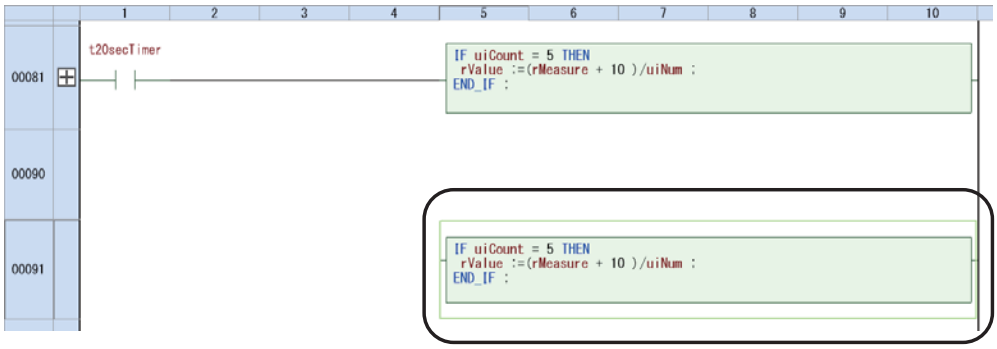
1 Select the Box ST to be copied, then select "Edit (E)" → "Copy (C)" from Menu.



Other way
- Ctrl key + C key
- From the right-click menu, select "Copy (C)"

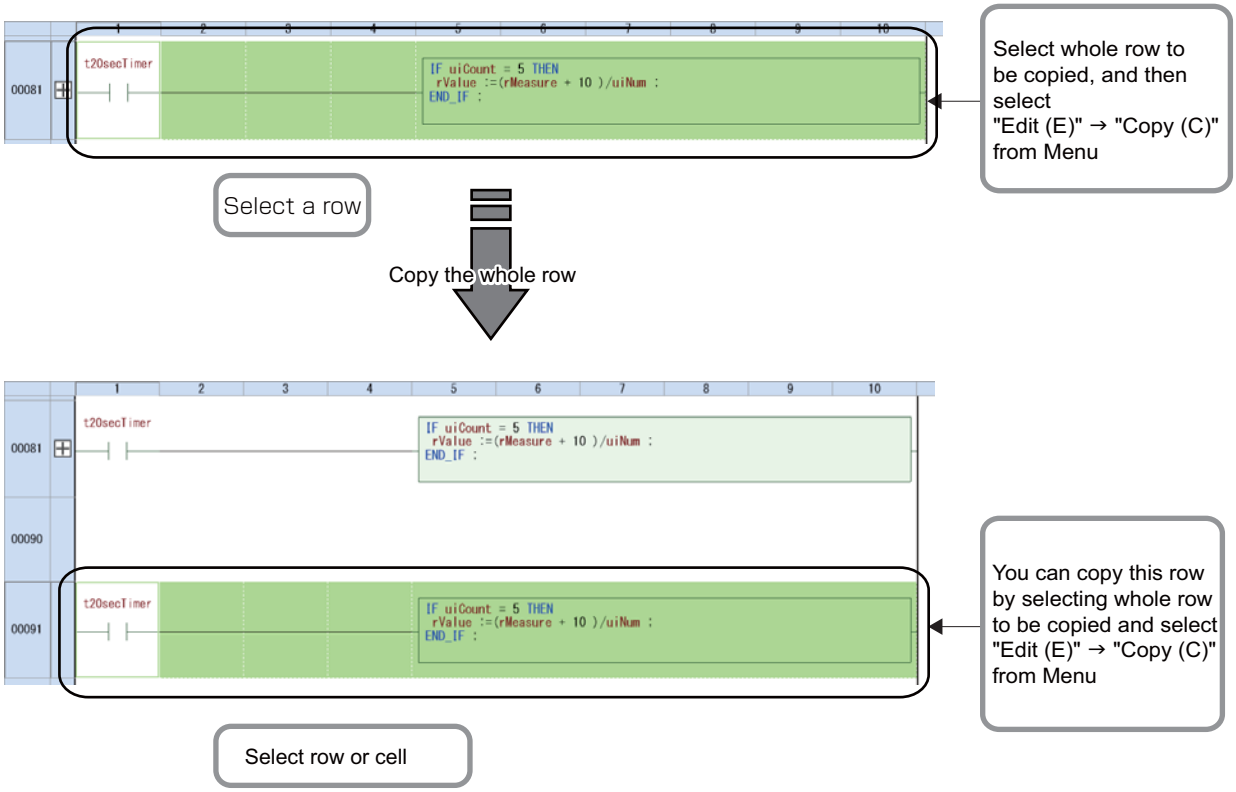
2 **Select the cell (line) to paste the Box ST copied in step 1.**

From Menu, select "Edit (E)" → "Paste (P)".



Other way
- Ctrl key + V key
- From the right-click menu, select "Paste(P)"

Reference The entire line including Box ST can be also copied.



Precautions on using Box ST

When the execution condition turned OFF, the ST program will be not executed by the Box ST. Therefore, pay attention to the following situations.

● When using Differential Execution Function

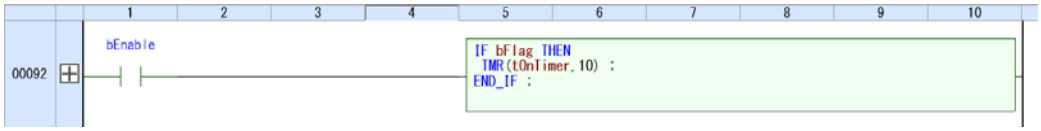
The previous state of the execution condition of the Differential Execution Function remains the state when the last Box ST (subroutine program) was executed.
Pay attention to the cases: when continuous scanning is used without causing the [n+1]th Box ST to be executed after the nth Box ST, and when executed twice in one scan.

For details, refer to Instruction Reference Manual, 📖 "Precautions on using Differential execution instruction"

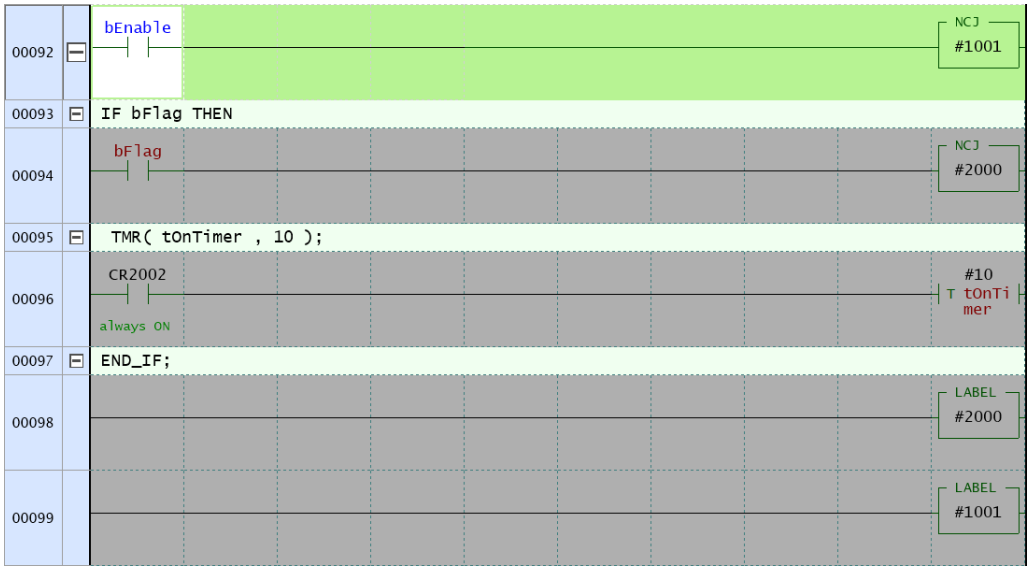
● When using Timer Function

When Timer Function is used by Box ST, TMR instruction will be written between NCJ to LABEL of the extended Ladder program. From this, the Timer contacts fail to be correct ON/ OFF sometimes. Besides, TMR Function whose execution condition omitted shall be also noted.
For example, when writing the following programs:

Box ST



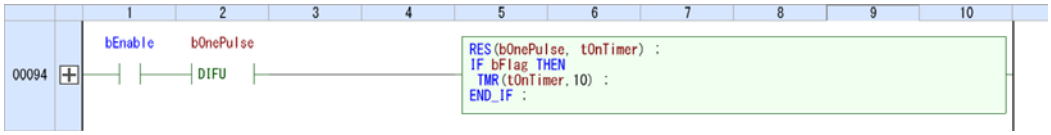
Extended Ladder



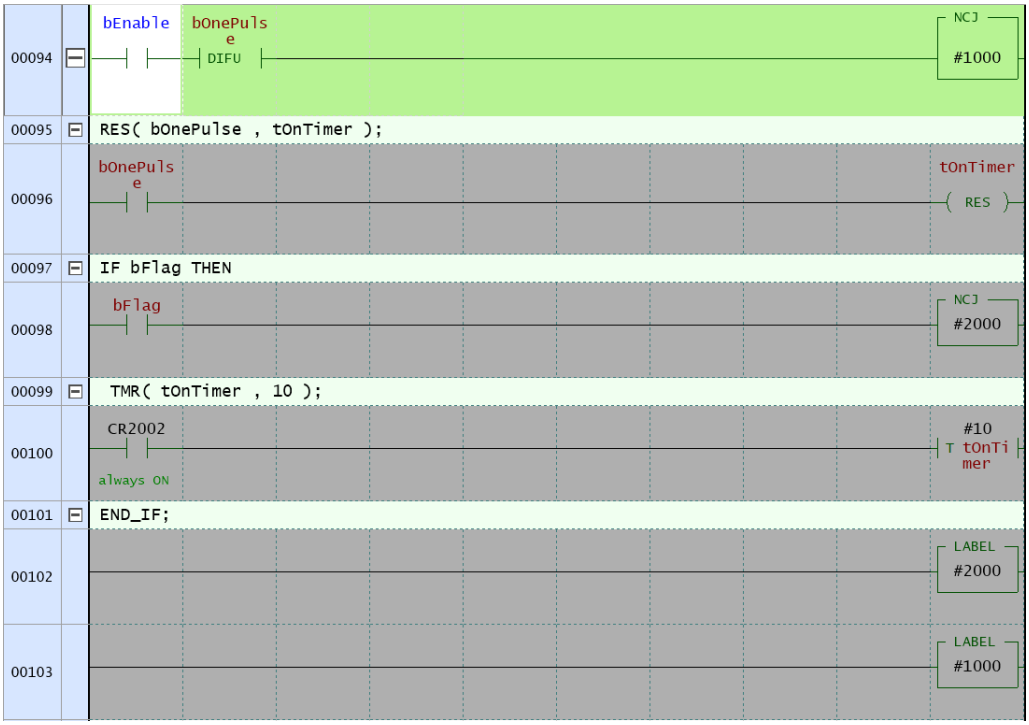
Check the extended Ladder program and confirm whether the reset condition of the Timer instruction is CR2002 (always ON). If it is, once the instruction is executed, the Timer cannot be reset.
To solve this problem, refer to "As the execution condition, connect DIFU instruction via AND".

As the execution condition, connect DIFU instruction via AND

Box ST



Extended Ladder



When the execution condition (MR1000) of Box ST starts up, MR1100 turns ON only in one scan by DIFU instruction. In this case, RES Function written in the first line of Box ST will reset the Timer. Therefore, the Timer Function (Contact) operates correctly as the Timer is always cleared immediately after the Box ST execution starting, regardless of the previous state of the Timer.

Area ST

This section describes precautions on how to create, edit and use the Area ST.

Features of Area ST

Area ST writes all lines in ST.

Area ST is used in the following cases

Area ST is every-scan execution type, suitable to the program including Condition Branch or Loop Control.
When writing complex condition branches or multiple branches, the program can be simplified.

Area ST

```
00090
//Select the value to store in the uiResult according to the value of the uiValue
CASE uiValue OF
 1: uiResult  := 10 ;
 2: uiResult  := 20 ;
 3: uiResult  := 30 ;
 4: uiResult  := 40 ;
ELSE
 uiResult  := 50 ;
END_CASE;

//If the calculation result is 100 or more, add +1 to the guitar value.
diWeight := ( diHeight * diWidth * diDepth ) * rDensity ;
IF diWeight > 100 THEN
 uiCount := uiCount + 1 ;
END_IF;
```

* The Description on the right of slash(//) will be processed as a comment.

Reference The area enclosed by the reference (* and *) is processed as a comment. When using an indirect reference (such as * EM0), sert one character space between (and *.

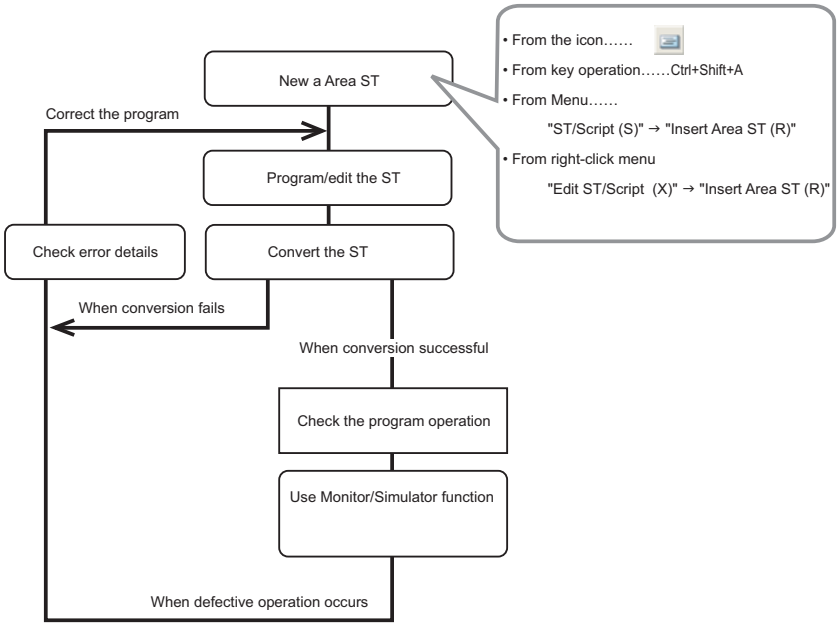
If there are multiple Condition Branches, or program is lengthened via Ladder programming, Area ST can enable improving programming efficiency.

Area ST creating

The procedure from Area ST creating to debugging is described as follows.

Area ST is used in the following cases

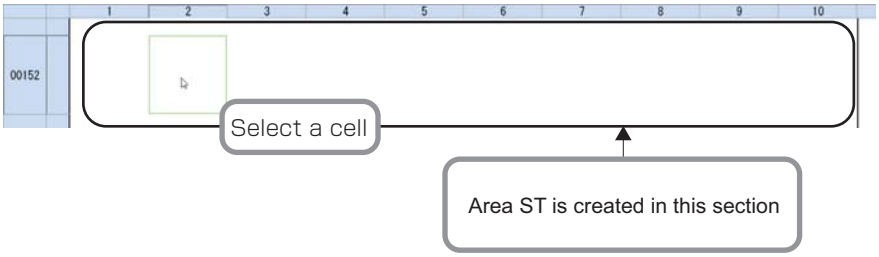
The operation procedure for Box ST creating/editing is described as follows.



Area ST newing

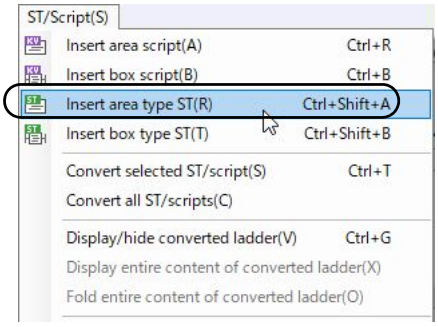
The creation steps of Area ST are described as follows.

1 The creation steps of Area ST are described as follows.



Reference Area ST occupies entire one line (10 cells).

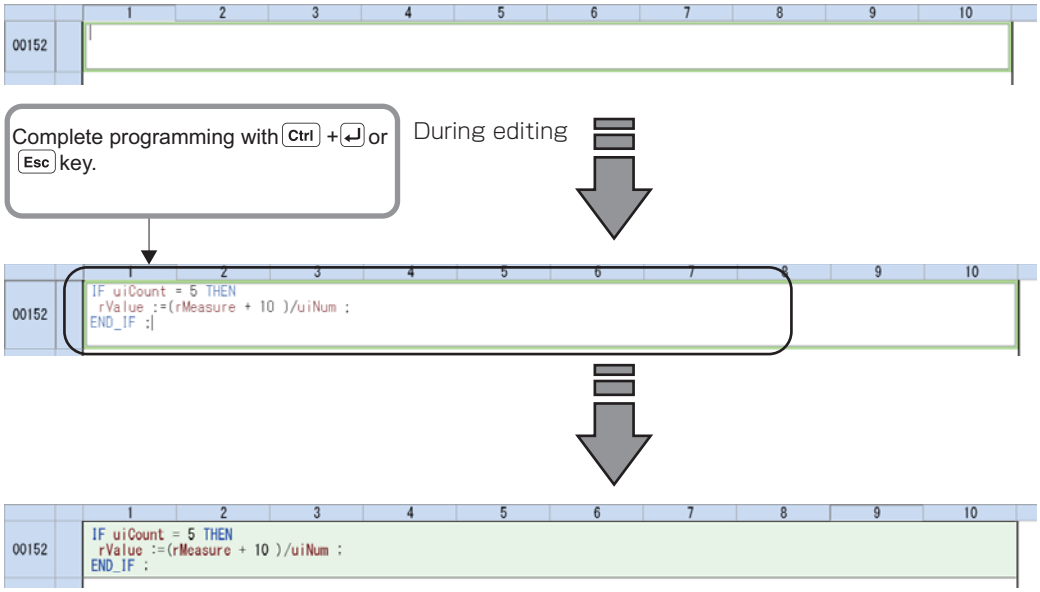
2 **From Menu, select “ST/Script (S)" → "Insert Area ST (R)".**



* Area ST can be inserted from Menu, Tool bar and the keyboard.

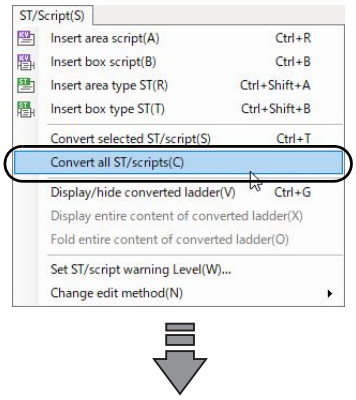
Other way
- Press Ctrl + Shift + A.
- From right-click menu, select "Edit ST/Script (X)" → Insert Area ST (R)"

3 **Area ST will be inserted. Write the program.**
After writing, press Ctrl key + ↵ key, or Esc key, or click the Ladder Editor window other than which Area ST is being edited to complete writing.

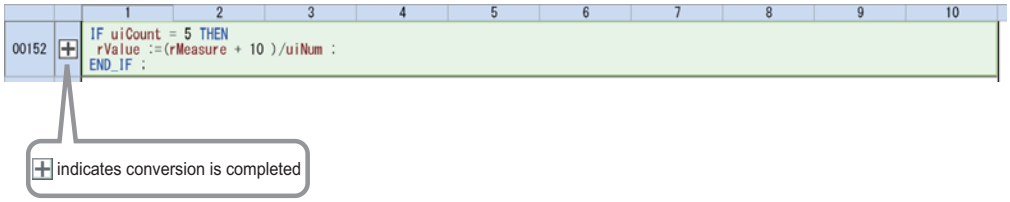


Editing is complete

4 **Convert the Area ST after the writing is completed.**



● When conversion successful



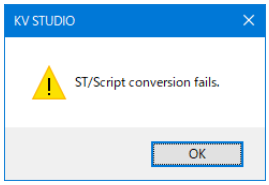
A "Warning" message may also be displayed in the pop-up window.

"Warning message"(page 2-24)

● When conversion fails

"ST conversion error" will be displayed.
Also, an error message will be displayed in the pop-up window.

"When conversion error occurs"(page 2-24)



Area ST reediting

The re-editing steps of Area ST are described as follows.

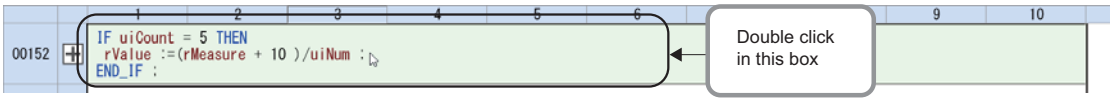
■Reediting Area ST

1 To re-edit Area ST that has already been written, double-click the Area ST to be edited.

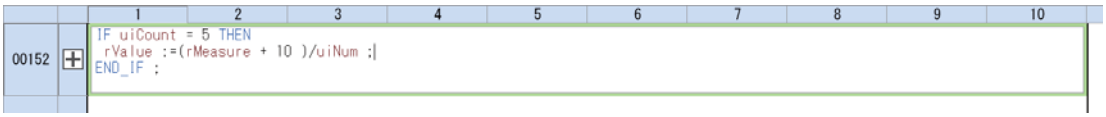
Other way

Select Area ST to be edited, press the following keys.

- Esc key
- F2 key



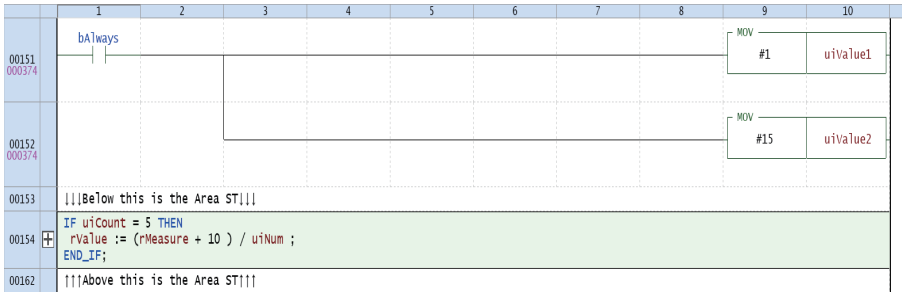
Uneditable



Editable

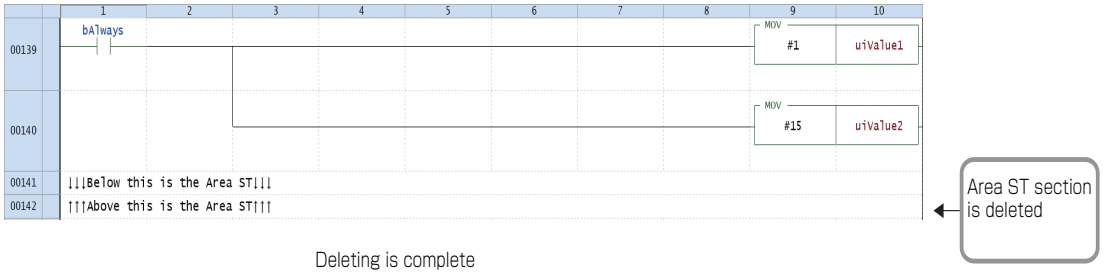
■Deleting Area ST

1 Select the Area ST to be deleted.



Area ST is selected

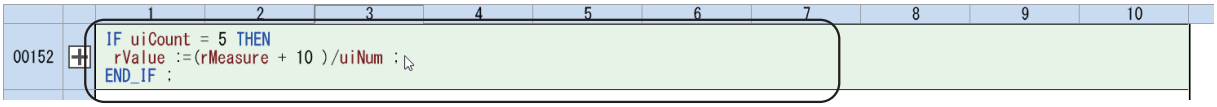
2 **Press Delete key on the keyboard.**



Deleting is complete

■Copying Area ST

1 Select the Area ST to be copied, then select "Edit (E)" → "Copy (C)" from Menu.

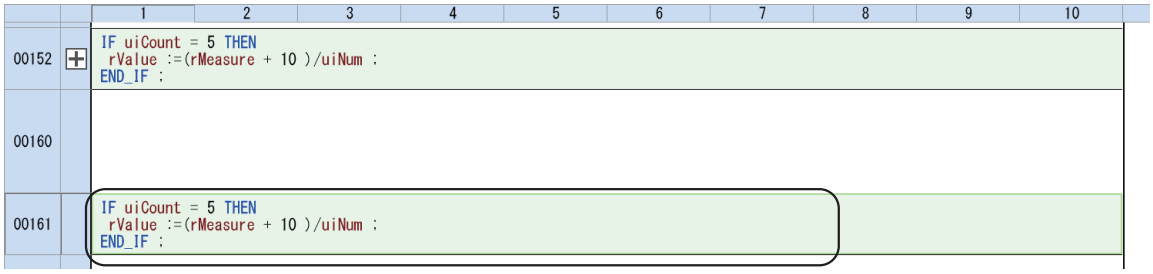


Area ST is seclected

Other way
- Esc key + C key
- From the right-click menu, select "Copy (C)"

2 **Select the cell (line) of Area ST copied in step 1.**

From Menu, select "Edit (E)" → "Paste (P)".



Copying is complete

Other way
- Esc key + V key
- From the right-click menu, select "Paste(P)"

Variable Programming and Arrays/Structures

This section describes the Variable programming methods and precautions, as well as how to use Arrays and Structures.

Variable programming

Variable refers to a programmatic element with a defined data type and name. ST Language can use Variables for programming in addition to Devices. Programming using Variables has the following features.

- Since the predefined names can be used for programming, the program readability is improved.
- The required Variables will be automatically assigned to the work area of the CPU unit according to the defined Variable size. Therefore, there is no need to care about the assignment.
- Device map is no longer created in advance, debugging efficiency, etc. is improved, and the man-hours for creating a program are shortened.
- Variables are automatically assigned even when a program is copied between different projects. Unlike Devices, Device map need not be changed before copying, which makes the program recalling easily.
- Since the touch panel display of VT5 series also supports the Variables of the KV-8000 series, there is no need to change the Device No., etc. on the touch panel display when changing a Device No. of PLC like the Devices.

Global/Local variables

There are 2 types of Variables: Their features are as follows.

Description	Global variables	Local variables
Entity	The same entity is processed jointly throughout the project. When the same Global variable name is written in different programs, the same entity is referenced.	Variables that can be used only in the program. Different entities are referenced even if the same Local variable name is written in different programs.
Assignment	There are two types: manual assignment (to Devices), and automatic assignment (to work memory by CPU unit).	The CPU unit automatically assigns the Variables to the work memory. Cannot be assigned to a Device.
Usage	Can be used in areas to communicate with external Devices such as touch panels. Also, the expansion Unit devices such as Operation Enable can be programmed using Variables with assigning the expansion Unit devices in advance.	Can be used as operation areas because no conflict occurs between different programs even if the same Variable name is used.

Variable/Data type list

To use Variables, it is necessary to define the data type in advance. The CPU unit automatically assigns the necessary data size to the work area according to the defined data type *1.
The work area sizes and Variable specifications that can be used with “KV-8000” are as follows.

Item		Description
Work area size	Bit	64000 points
	Word	589824 words
	Timer	4000 points
	COUNTER	4000 points
Variable specification	Definable size	100000 points*2
	Maximum length of Variable name	128 characters*3

*1 If it is defined as a Global variable, it can also be manually assigned to a Device of the CPU unit.
*2 The definable number refers to the Variable name that can be registered in the Variable edit window. For example, if one Array type variable with 10000 members is defined, it will be counted as one point.
*3 The character code of the Variable name is UTF-16.

Point

During automatic assignment, BOOL type variables may not be assigned to the start of the channel. When using a BOOL type variable for an operand that must specify the start of the channel, specify this variable assigned to the start of the channel with one of the following methods.

- **Create a local/global structure type variable with a BOOL type member having a bit offset of 0.**
- **Create a global variable with the start of the channel of the global device specified as the device to assign.**

The data type and data size of the Variables that can be used with “KV-8000” are as follows.

Data type		Data size
Type name	Description	(in bits)
BOOL	Bit	1
UINT	1-word unsigned integer	16
INT	1-word signed integer	16
UDINT	2-word unsigned integer	32
DINT	2-word signed integer	32
REAL	Single precision floating point number	32
LREAL	Double precision floating point number	64
STRING*1	Character string	(Number of characters *2+1) × 8 rounded up to an integral multiple of 16
Timer	Timer	—
COUNTER	COUNTER	—
ARRAY	Array	Data size in data type × Array size
Structure	—	Based on definition*3
Function block	Function block	Based on definition*3

*1 The character code processed in STRING type follows the language set in the project language settings. For details on project language settings, refer to "ST Language User's Manual".

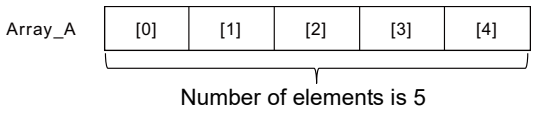
*2 The number of characters refers to the number of bytes up to NULL. For double-byte characters such as Shift-JIS, the number of characters in one character is calculated as 2.

*3 Structures and Function blocks are data types that use multiple data types. Therefore, the data size is determined by the settings.

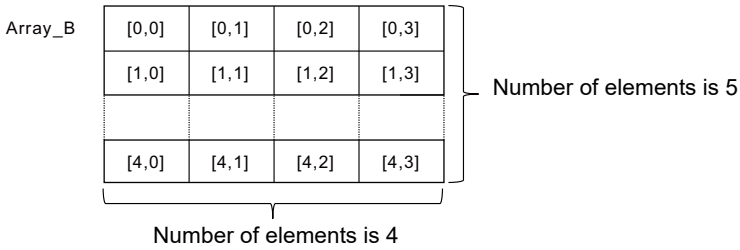
Array

Array refers to a data type that defines a data legion of the same data type with a common name. Each data that constitutes an Array is called Element. When using an Array, the data type, number of elements and number of dimensions should be set in addition to the Variable name. An Array with n dimensions is called an n-dimensional Array. An example of 1-dimensional Array to 3-dimensional Array are shown below.

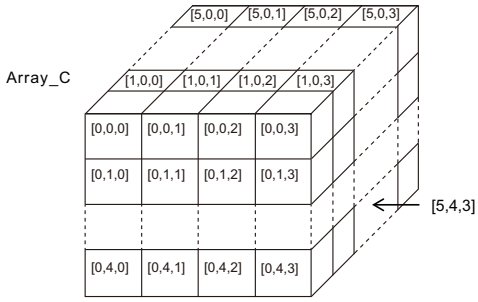
•1-dimensional Array In case of the Variable name is Array_A, and number of elements is 5



•2-dimensional Array In case of the Variable name is Array_B, and number of elements is 5×4



•3-dimensional Array In case of the Variable name: Array_C, and number of elements is 6×5×4



■Writing Array type variables

●Specifying elements for Array type variables

When an Array element is specified and used in a program, the combination of the Array name and the subscript of [ ] is used. For an Array with 2 or more dimensions, specify the element numbers separated by "comma (,)". For example, when using the elements [5,0,3] of 3-dimensional Array Array_C, write as "Array_C[#5,#0,#3]" in the program as follows.

```
Array_C[5,0,3] := 0 ;
```

●Specifying the Entire Array type variable

All elements that constitute an Array are called "Entire Array". When specifying an Entire Array in a ST, only write the Variable name. For example, when specifying the Entire Array of a Variable called Array_C, write as "Array_C".

```
uiCount := ACNT(Array_C, 2);
```

Array specification

Item	Description
Maximum number of elements per dimension	589823
Maximum number of dimensions	8
Maximum data size that can be occupied per Array	589823 words
Specifiable data types	BOOL/UINT/INT/UDINT/DINT/REAL/LREAL/STRING/Function block/Structure
Specifiable subscripts*1	Device/Constant/Variable/Operation expression*2
Data types available for subscripts	UINT/INT/UDINT/DINT

*1 Specify 0 or a positive integer for the subscript. If any other number is specified, a conversion error or operation error will occur.

*2 Only ST/Script can use Operation expressions as the subscript. Operation expressions cannot be used as subscripts in Ladder programs.

Point

- **When specified elements exceed the specifiable range, such as specifying Array_A[10] (11th element) for an Array called Array_A set with the number of elements "10", the value in an unintended area may be referenced. Especially when using Variables or Expressions as subscripts, the ACNT Function or WSIZE Function should be executed before referencing the Array to obtain the number of elements in the Array or the data size of the Entire array, and avoid referencing elements beyond the specifiable range.**
  "ACNT"(page 6-24)
  "WSIZE"(page 6-25)
- **If a Device is specified as the subscript of an Array, it is processed as a 16-bit unsigned integer.**
- **Bit devices (R, MR, etc.), Bit-positioned Word devices and BOOL type variables cannot be specified as the subscript.**
- **Depending on the KV STUDIO version, array data types are expressed as follows:**
  Example: UINT type array with three elements
  Ver. 11.20 or later: ARRAY[0..2] OF UINT
  Ver.11.10 or earlier: UINT[3]

Structure

Structure refers to a data type that defines a data legion of multiple data types with a common name. Each data that constitutes a Structure is called Member. When using a Structure, the names of the members that constitute the Structure and the data types of the members should be set in addition to defining its name. Structure has the following advantages.

- Registering the Structure members and data types in advance makes it easier to create Variables with the same Structure.
- All members can be monitored in batch by registering a Structure variable for monitoring in the Register Monitor window, etc.

The following is an example of using the following motor axis information in a program.

Current coordinate (CurCoordinate)
Current speed (CurSpeed)
Axis control (Control)
Axis error (Error)
Axis error code (ErrorCode)

If multiple axes are used without using a Structure, Variables that differ only in axis numbers should be registered as shown below.
In case of using a Structure for multiple axes, only predefining the data Structure of axis information as a Structure in advance sets the same Structure for the data type.

<When not using a Structure>

Variable name	Data type
Ax1_CurCoordinate	DINT
Ax1_CurSpeed	DINT
Ax1_Control	BOOL
Ax1_Error	BOOL
Ax1_ErrorCode	UINT
Ax2_CurCoordinate	DINT
Ax2_CurSpeed	DINT
Ax2_Control	BOOL
Ax2_Error	BOOL
Ax2_ErrorCode	UINT

<When using a Structure>

Variable name	Data type
Ax1	AxisInfo
Ax2	AxisInfo

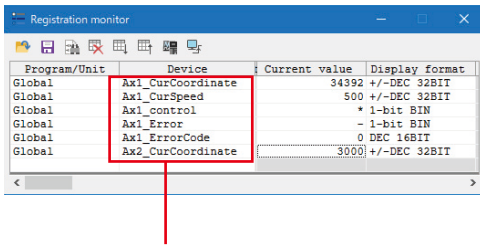
Definition of "AxisInfo" type data

Member name	Data type
CurCoordinate	DINT
CurSpeed	DINT
Control	BOOL
Error	BOOL
ErrorCode	UINT

Recall the preset Structure

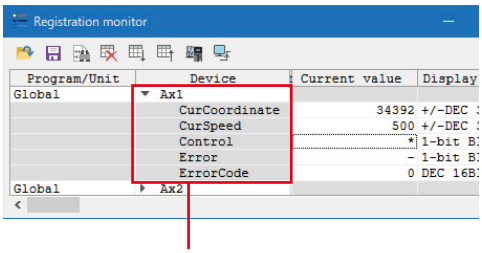
Also, if no Structure is used when monitoring on ST Language, it is necessary to manually enter all Variables, while if a Structure is used, the members in the Structure can be monitored in batch simply by entering a Variable name.

<When not using a Structure>



All Variables to be monitored must be entered.

<When using a Structure>



When using a Structure, the members can be monitored in batch simply by entering a Variable name.

■Writing Structure type variables

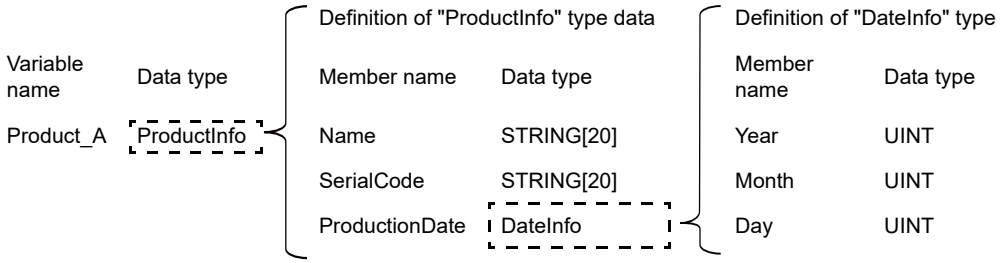
This section describes how to write Structure type variables when using them in a program.

●Specifying Structure type variable members

When using a member of a Structure type variable in a program, write as "Structure type variable name + Period (.) + Member name". For example, "Ax1.CurSpeed" is written to reference a CurSpeed member of AxisInfo type variable Ax1.

```
diSpeed := Ax1.CurSpeed ;
```

Structure can also be specified for the data type of a Structure type variable member. In the following example, a Structure type variable "ProductionDate" is nested in the Structure type variable Product_A of "ProductInfo".



When specifying Day of Product_A with a program, write as "Product_A.ProductionDate.Day".

```
Product_A.ProductionDate.Day :=13 ;
```

Point **Up to 8 stages can be nested for a Structure.**

●Specifying Entire Structure type variables

All members that constitute a Structure type variable are called "Entire Structure". When an Entire Structure is specified in a program, only the Variable name is written. For example, when specifying the Entire Structure of a Structure type variable called Product_A, write as "Product_A".

```
WSIZE(Product_A) ;
```

Point **When writing a value to the Entire Structure using the FMOV Function, etc., the value in an unintended area may be referenced if you specifying a send data value larger than the data count for the Entire Structure. When referencing the Entire Structure with the FMOV Function, etc., the WSIZE Function should be executed before the Function execution to obtain the data size of the Entire Structure.**

「WSIZE」(page 6-25)

Structure specification

Item	Description
Maximum number of Structures defined	800
Maximum number of members per Structure	500
Maximum number of stages for Structure nesting	8*1
Data types that can be specified as members	BOOL/UINT/INT/UDINT/DINT/REAL/LREAL/STRING/Array/Structure

*1 When stAAA.member is written, the number of nest stages is 1. When stAAA.stBBB.member is written, the number of nest stages is 2.

Variable setting

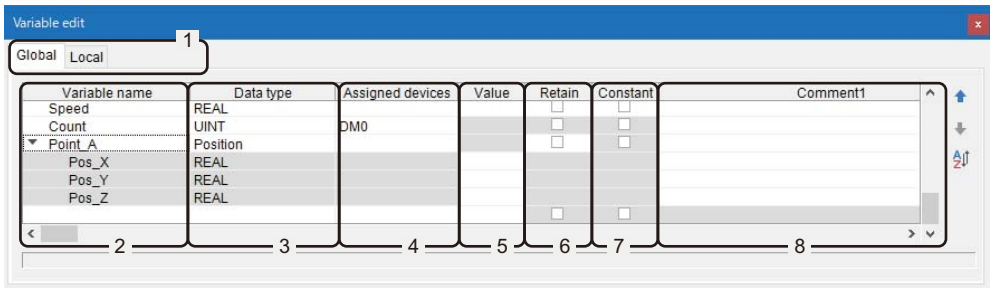
This section describes the procedure from defining Variables to using them in a Ladder program.

■How to set from Variable edit window

Global/Local variables can be set with this method.

1 Select "Display (V)" → "Variable edit window (L)" to open the "Edit Settings" window.

Other way Double-click "Variable" in the workspace



No.	Item	Description
1	Global/Local tab	Switch between the Global/Local tab of Variable edit window. "Global/Local variables"(page 2-15)
2	Variable name	Enter the Variable name (128 1-byte characters/128 full-width characters).
3	Data type	Select the data type of the Variable from below. BOOL/UINT/INT/UDINT/DINT/REAL/LREAL/STRING/ TIMER/COUNTER/ARRAY/Structures/Function block
4	Assigned device	Set when manually assigning Variables to Devices of the CPU unit. Assignment is not mandatory. This item does not exist in the Local tab of Variable edit window.
5	Value	When a "Constant" item is checked: Specify the initial value of the Variable. When the constant is checked, the value is fixed and cannot be changed from the program. When a "Constant" item is unchecked: Set the initial value of the Variable. The value can be changed in the program. When the assigned Device has been set, the initial value cannot be set in the Variable edit window. The initial value of the assigned Device should be set in the Device initial value setting. For details, refer to the CPU Unit User's Manual.
6	Retain	Checked Variables hold their values even when the power is turned off. The maximum number of Variables that can be held is 30000 including Bits, Words and Counters. When the assigned Device has been set, the Power Off Hold setting should be performed for the set Device which is not included in the above 30000 Devices. Variables without Hold checked will be initialized to the value set in "Value" when switching from PROGRAM mode to RUN mode. It is initialized to 0 when not be set. For details, refer to the CPU Unit User's Manual.
7	Constant	Variables checked are processed as constants. Variables that set as constants are included in the maximum of 100000 definitions, but they do not occupy the work area.
8	Comment	Set the Variable comments (128 1-byte characters/128 full-width characters). The set comment is displayed on the ST Language Ladder Editor window. Comments 1 to 8 can be set for one Variable.

*1 If the CPU function version is earlier than 2.1, it can be set to 32 half-width characters and 32 full-width characters.

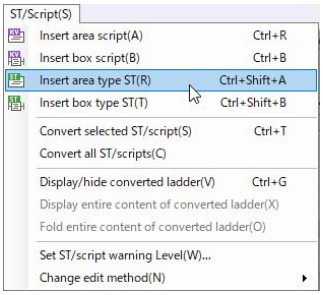
■How to import and set unregistered Variables in ST

With this method, only Local variables can be set.

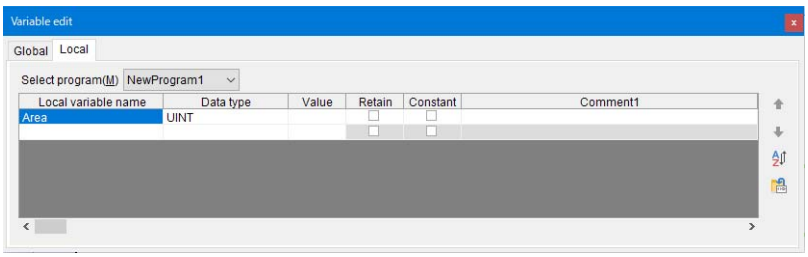
1 Write the ST using Variables.

```
Area := Height * Base /2 ;
```

2 **With the string to be registered as a Local variable in the ST selected, select "Register Local Variable(L)" from the right-click menu.**



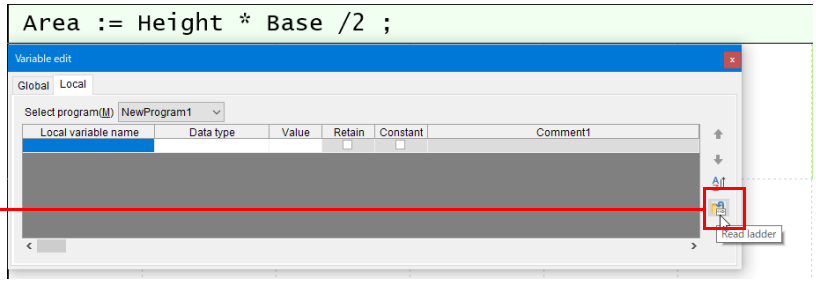
3 **The Variable edit window will be displayed with the Variable name imported. Set the data type.**



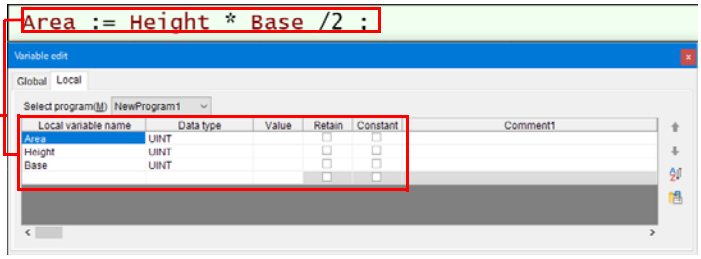
●Batch-registering Local variables

This section describes how to import unregistered Variables in a Module in batch as Local variables. Click in the Variable edit window when there are unregistered Variables. All unregistered Variables in the target Module are imported as Local variables. Set the data type.

Click the Variable import icon



Unregistered Variables are imported in batch, and the display color of registered



Point

All Variables in the Module will be registered after performing batch registration. Therefore, errors are also registered as Variables, and errors that originally occur during conversion may be converted normally.

Example **DN1000=10;..... Before batch registration: error occurs during ST conversion**

..... After batch registration: error will not occur during ST conversion

It is recommended to confirm the registered contents in the Variable edit window after batch registration.

Precautions on using Variables

This section describes restrictions and precautions on using Variables.

● Variable range check

On KV-8000, the range check is performed to avoid the Function execution result referencing an area that exceeds the Variable range. The content of the range check differs depending on the Variable settings.
The following is an example of using the Variable Array_A with the number of elements set to 10.

For Global variables (auto assignment)/Local variables
When a constant exceeding the number of elements (such as Array_A[20]) is specified, an error will occur during conversion.
When other than a constant (such as Array_A[idx]) is specified for the number of elements, no error occurs during conversion. If the idx value exceeds the range of 0 to 9 during program execution, an operation error will occur during instruction execution.

For Global variables (manual assignment)
The following is an example in which the leading of Array_A is assigned to DM0.
When a constant exceeding the number of elements (such as Array_A[20]) is specified, an error will occur during conversion.
When other than a constant (such as Array_A[idx]) is specified for the number of elements, a warning will be issued during conversion, but no error will occur. If the idx value exceeds the range of 0 to 9 during program execution, the object Device will be processed, and no operation error will occur. For example, when idx=19, Array_A[idx] will be processed as DM19.

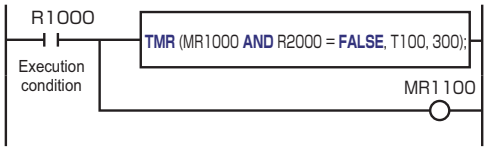
● Other precautions

- The Global variable can be set to the same Variable name as the Local variable, but it operates as a Local variable when used.
- Variable names cannot use the strings such as reserved words.
  "List of Characters That Cannot Be Used" page A-11
  Index modify cannot be performed for Global variables and Local variables automatically assigned.
- Index modify and Indirect designation cannot be performed for Variables of Array type and Structure type.
- When "Constant" is checked in the [Variable edit] window, Index modify and Indirect designation cannot be performed.

Precautions on ST Writing

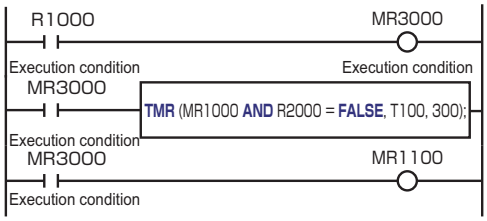
About Parallel connection of Area ST

OR connection cannot be used between Area ST and Ladder instruction.
Error will occur during conversion in the program on the right example.



● Ladder program equivalent to OR connection

Write it as if it is logically OR connected via another Device (MR3000 on the right).



Precautions on Writing Operation expression in ST

Accuracy may drop during operation. It is also possible to set to display a "warning" when accuracy drops. "Warning message" (page 2-24)

ST Conversion Messages

ST program is transferred to the CPU unit after converted to Ladder program.
This section describes the warning message and error message displayed while ST is converted to Ladder program.

Warning message

The message displayed when expected operation result cannot be obtained after the execution of Ladder program converted from ST program. Solve the warning causes as possible.
It will occur in the following cases: the result type of Assignment statement on the right side is different from the left, the Operation expression includes .D, or the Function cannot be executed with the current model.

■Data loss by type conversion

In the following ST, the required data may not be obtained if data loss occurs.

Program ST uiValue := iValue * 10000;

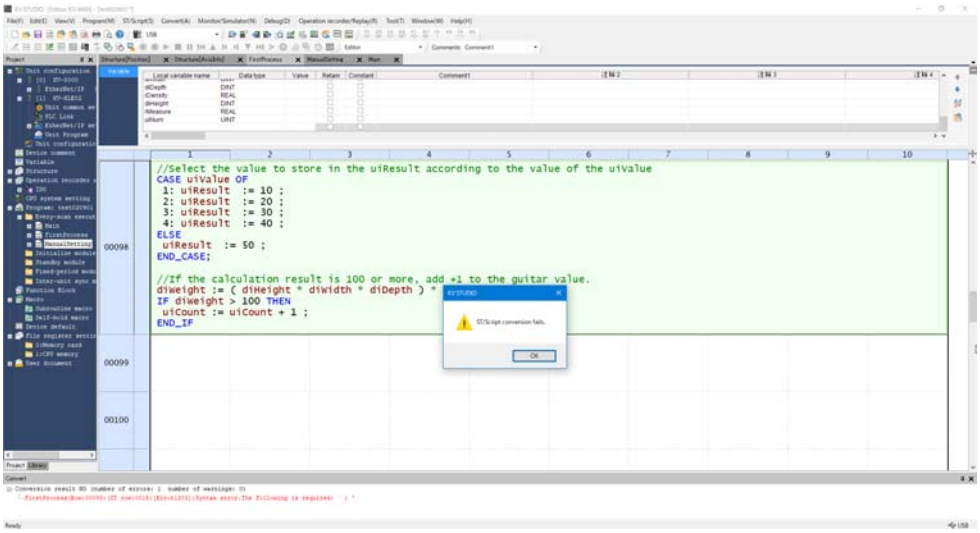
For example, in case of the value of iValue is -1 and the type of uiValue is UINT, the correct operation result cannot be stored. Also, if the value of iValue is 32767, the operation result exceeds the UINT range (0 to 65535), so the expected (correct) operation result cannot be assigned.

Error message

Error will occur when part of ST programs can not be converted to Ladder program, and error causes will be displayed in the pop-up window.
The program cannot be transferred to CPU unit unless the error cause is removed and the conversion is successful. Correct the ST according to error message displayed in the pop-up window.

When conversion error occurs

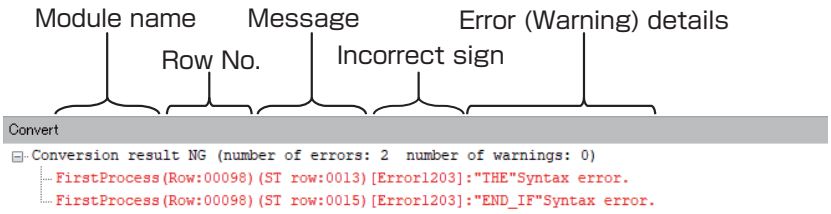
"ST conversion error" will occur and a dialog box will be displayed when converting the ST in the following cases: with a typo in the Function name, incorrect type (suffix) specification, or incorrect usage of the Control statement.



When a conversion error occurs, the error position and error details will be displayed in the pop-up window.

■Errors displayed in the pop-up window

The following error details will be displayed when a ST conversion error occurs:



"Error" messages : State in which the program cannot be converted to Ladder program. Programs cannot be transferred to PLC under this state.

"Warning" messages : The program can be converted to Ladder program. Programs can be transferred to PLC and executed by Simulator under this state.
However, when you execute the program, you may not get the expected operation result.

■How to use the pop-up window

Error (warning) details will be displayed in the pop-up window when the conversion error occurs.
Double-click the highlighted line and move the cursor to the error line in Ladder Editor window.
Correct errors according to error details.

1 Double-click the error details (①) displayed in the pop-up window.

The cursor moves to the line (②) with error details, and ST turns to editable state.

2 Correct the error program.

3 After correcting, only the corrected ST are converted.

Select “ST/Script(S)" → "Convert selected ST(S)" from Menu.

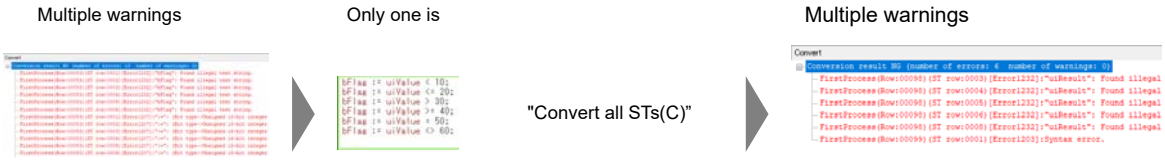
Other way

- Ctrl key + T key
- Select "Edit ST/Script (X)" → "Convert selected ST(S)" from the right-click menu.

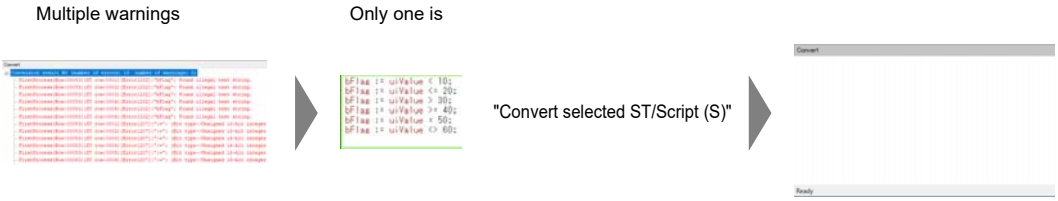
No error displayed in the pop-up window indicates that the correction is completed.

Reference

- In the case of multiple errors/warnings occurred, other errors will be still displayed when selecting "Convert all STs(C)" even after one error has been corrected.



In this case, select “Convert selected ST/Script (S)” from Menu.



- Only the remaining error details for the corrected ST will be displayed after the corrected ST is converted, and it is easier to check whether the correction is correct.
- By displaying/hiding (Ctrl+G) the converted Ladder, it can be displayed, confirmed or closed after expanded, while editing the ST.

Function Block

Function block refers to a generalized program that can be used to customize/ register the name in the Ladder circuit used repeatedly in the project.
The registered Function block can be called from the ST program, and can also be used in the ST within the Function block.

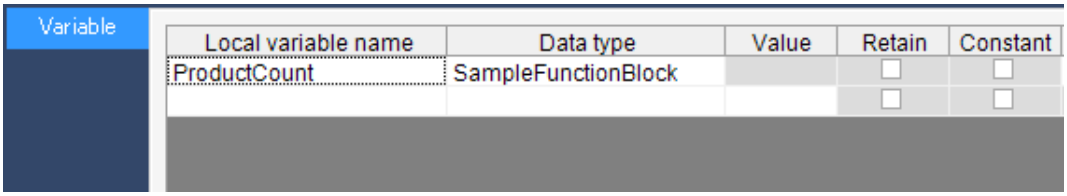
Calling Function block

■Calling Function block

In ST programs, you can use Function block instances as Functions as follows:

```
FBInstanceName (arg1, arg2, arg3);
```

When calling a Function block, first create a Function block instance.
By creating a Variable in the [Edit Variable] window and setting the data type of Variable to the Function block type, you can use the created Variable as a Function block instance




```
ProductCount(uiValue1, uiValue2, uiValue3);
```

Point

- **You cannot call a Function block that contains an ST program (that uses self-Function block instance).**
- **If the same Function block instance is called from another location, common resources will be used. To use the entities separately, prepare a Function block instance with another name.**

■Calling Function

In ST program, a Function name can be used as a Function as shown below.

```
FunctionName (arg1, arg2, arg3);
```

Function can be called simply by setting the Function name and Arguments in the ST program.

```
SampleFunction(uiValue1, uiValue2, uiValue3);
```

Debug function of ST

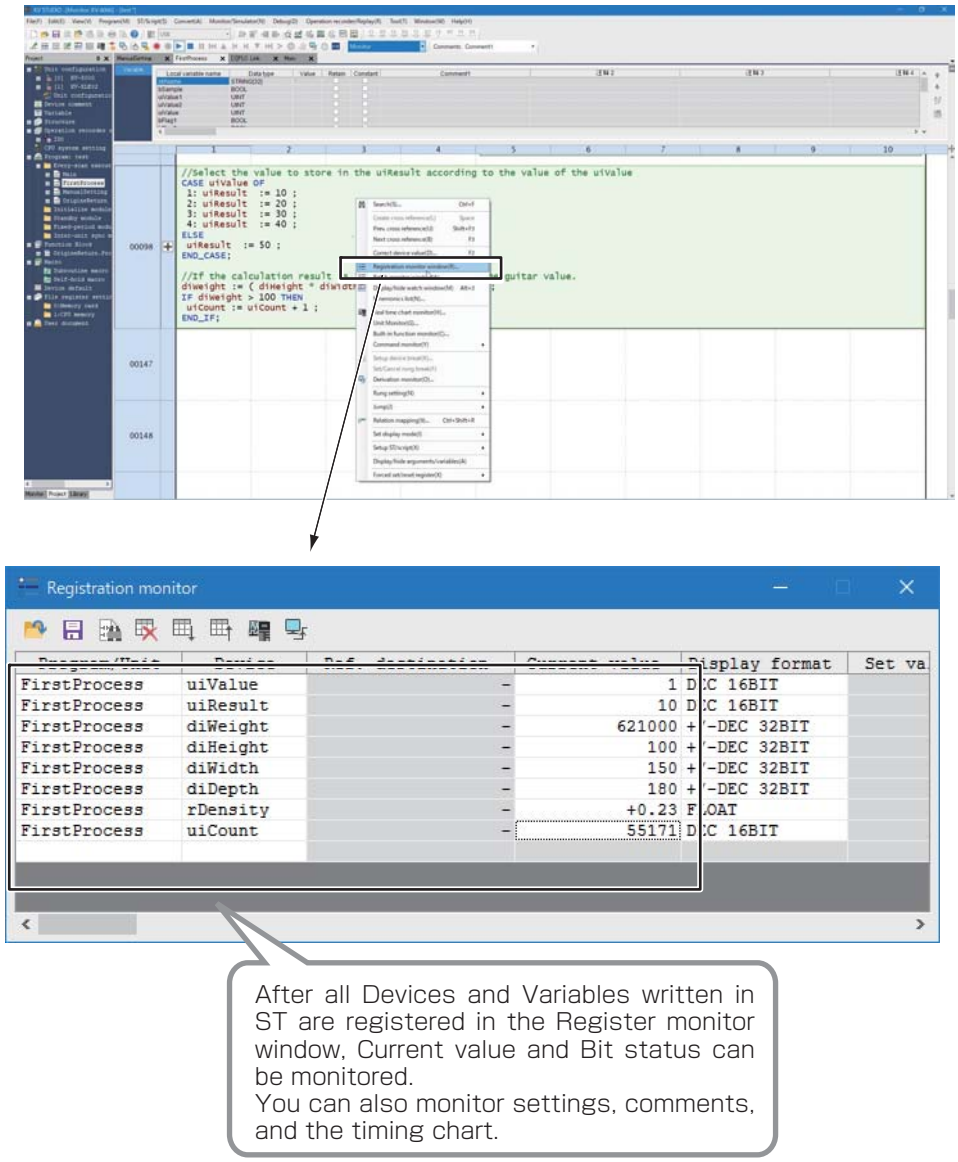
Monitoring function

Monitoring function allows to debug ST program while Online or simulator running.

■Register monitor

Available to monitor the Current value of every Device programmed in ST.

1 When ST Language is in Monitor or Simulator state, click the ST to be monitored, and then select "Register monitor" from the right-click menu.

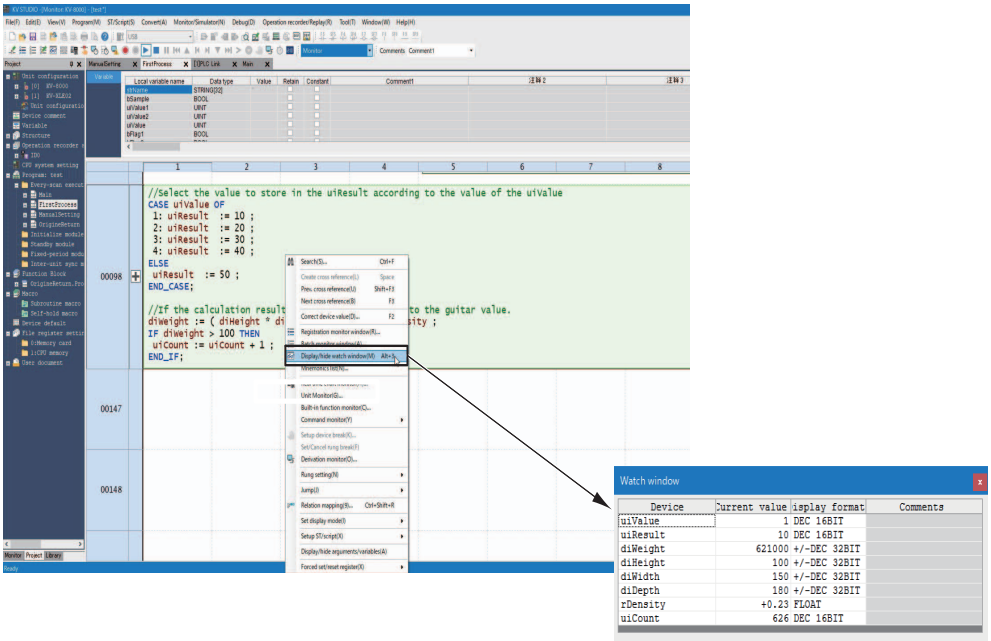


■Watch window

Through Watch window, you can monitor the Current values of all Devices or Variables in the ST by simply selecting the ST.

1 When ST Language is in Monitor or Simulator state, click the ST to be monitored, and then select "Watch window" from the right-click menu.

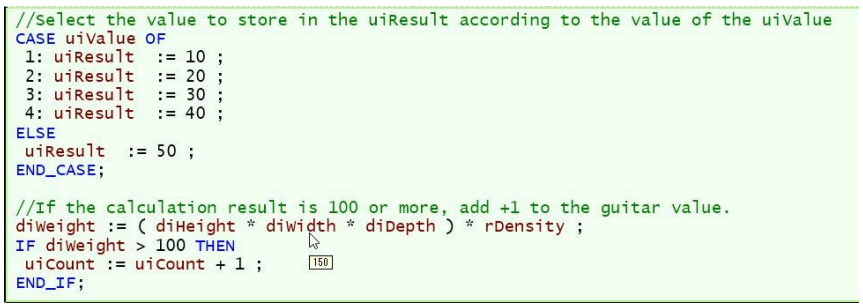
Other way



If selecting a different ST program, the Variables and Devices displayed in the Watch window will switch automatically.

■Monitoring with the Tooltip

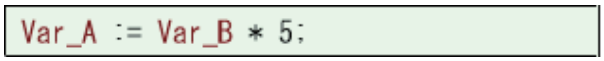
When ST Language is in Monitor or Simulator state, move the mouse cursor to the Device to be monitored, the Current value and Device comment (Variable comment in the case of a Variable) will be displayed in the Tooltip.



■Monitoring by ST Monitor

When KV STUDIO is in the monitor mode or simulator mode, the value of the variable will be displayed next to the variable.

Editor mode



Monitor mode/Simulator mode

Input Auxiliary Function and Others

About Input auxiliary function

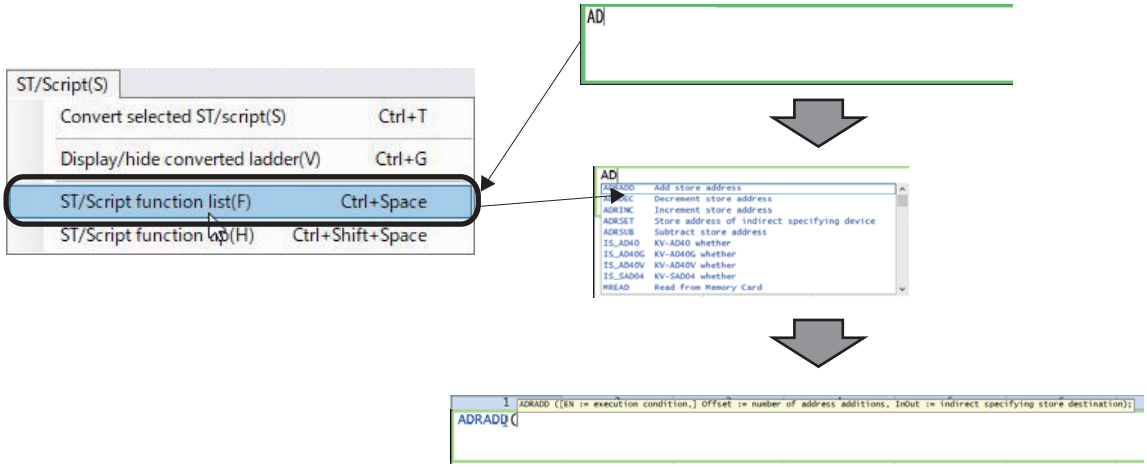
ST Language provides various convenient functions and input auxiliary functions for ST programming.

ST Function list

If the Function name is unknown, Function candidates can be called in one of the follows after entering the Function halfway.

● Calling Function list

From Menu, select “ST/Script (S)" → “ST/Script Function list (F)".



Select the Function to be used and press or double click ↵ key, the selected Function is inserted to the cursor position in ST Editor, and the ST Function expression prompt is displayed.

Other way

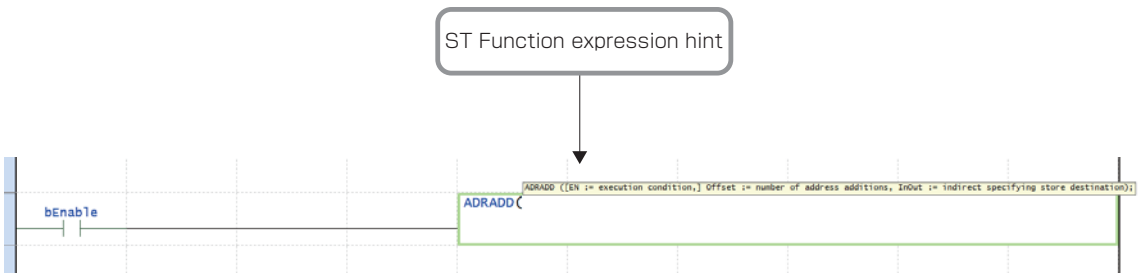
- Ctrl key + Space key
- From the right-click menu, select “ST/Script Function list (F)”

Reference

- ST Function candidates can be also called without any input. In this case, all candidate functions are displayed in alphabetic sequence.
- “ST Function candidates" can be also called during ST programming.

ST Function format hint

While entering a Function, "ST Function format hint" can simply inform messages such as Arguments and return values, etc.



After entering "Function name(", the “ST Function format hint" will be automatically displayed.

```
Return value := AVG (In := detection device, Count := number of data);
```

Return value | Function name | Number of Arguments (Two in this case)

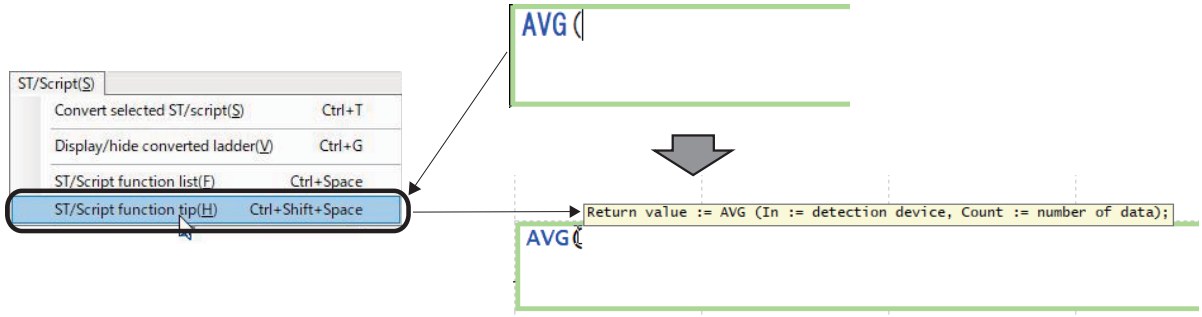
```
BMOV ([EN := execution condition,] Src := transfer source, Dst := transfer destination, Count := number of data);
```

Function name | Execution condition※ | Number of Arguments (Three in this case)

* Arguments written in [ ] can be omitted.

● Calling ST Function hint

Move the mouse cursor "I" to the right side of "(", and then select “ST(S)" →”ST Function hint (H)" from Menu.



Other way Ctrl Key + Shift key + Space key

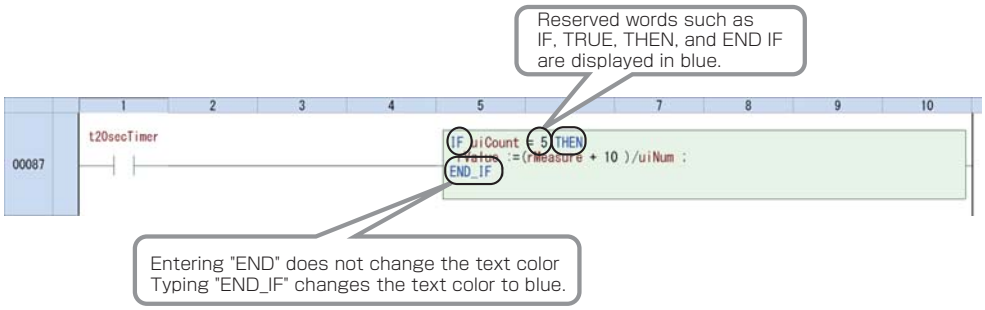
Reference “ST Function hint (H)" can be also called during ST programming.

Point **When calling “ST Function hint”, the cursor must be moved to the right side of "(" or the start position of Function. Otherwise, the ST Function hint cannot be called.**

Reserved words

Words (Reserved words) such as Control statements and functions, etc. registered to ST will be written in blue while programming. If the color not changes, it indicates an input error.

Reference The character color of reserved words can be changed via the "Option setting" dialog box of ST Language.



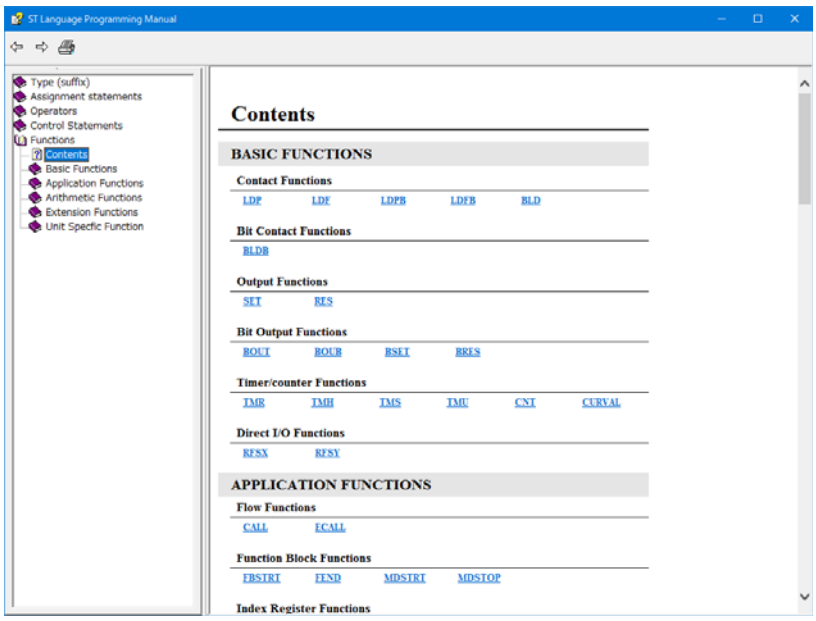
HELP function

For details on the format and usage of functions, as well as Control statements, refer to “ST Function Help".

■Starting ST Function Help

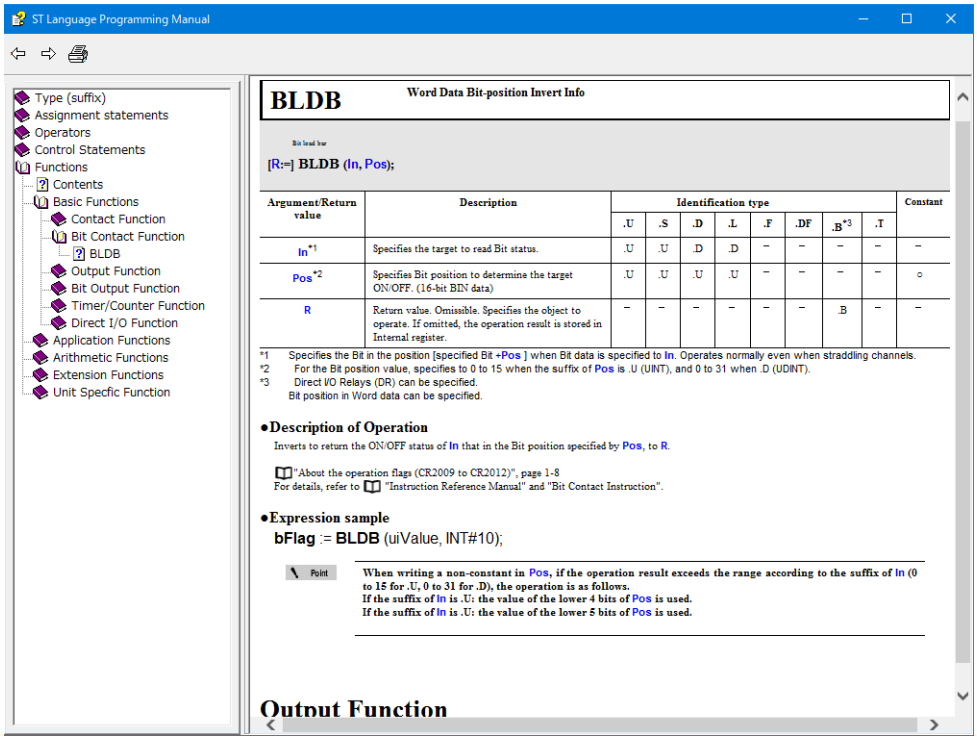
1 From Menu, select "Help (H)" → “ST Function Help (M)".

“ST Function Help" directory is displayed.



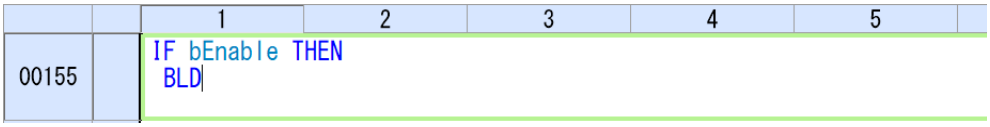
2 **Select the target Function.**

The Function details will be displayed.

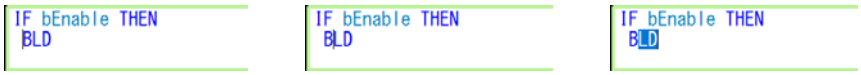


■Displaying the Help of specified Function on ST Editor screen

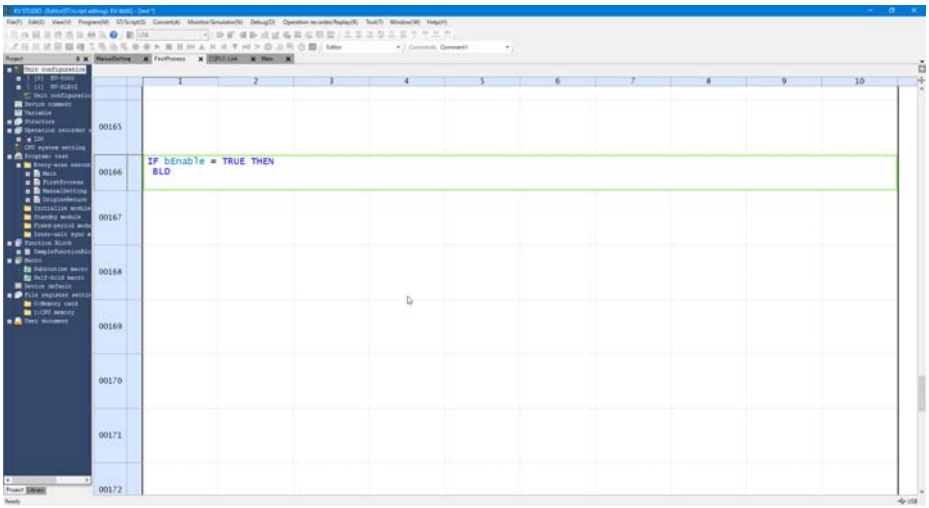
1 Move the cursor to the target Function.



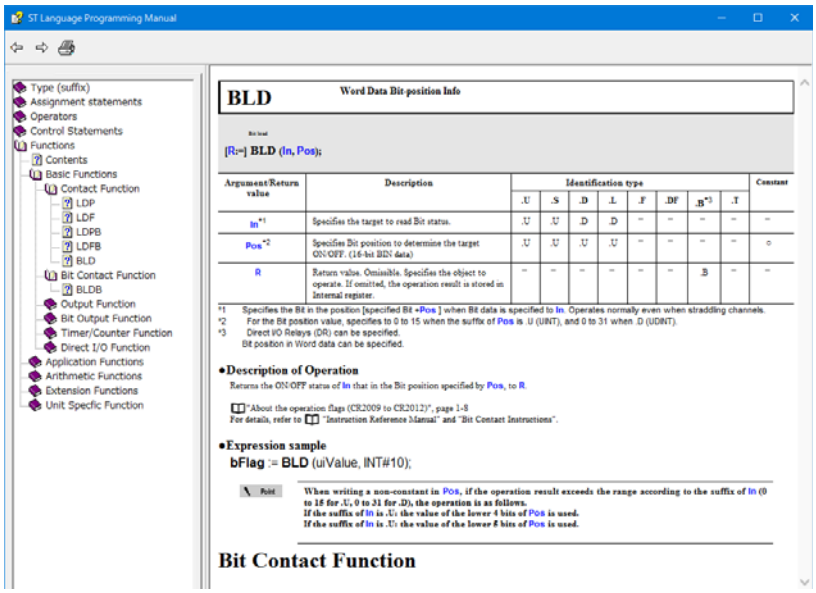
As shown in the figure below, move the cursor position to the front, back, and middle of the Function, or at least part of the Function is selected.



2 Press F1 key.



The Help screen of the selected Function is displayed.



About Comment function

ST supports one-line comment with "//" and range comment with "(* ... *)" and "/ * ... * /". Comment programming can improve efficiency of debug and program modification. It is recommended to write comments together with ST.

Reference

- Increasing comment lines will not influence the program size and scan time.
- The number of characters can be entered in one line is up to 1024 characters (1-byte) including ST and Comment.
- The following methods can be used to change multiple selected rungs into a single comment rung (KV STUDIO version 11.30 or later).
  - "Set line disable (Q)" on the right-click menu
  - "ST/Script (S)" → "Set line disable (Q)" on the menu
  - Ctrl + Q
- The following methods can be used to clear the selected multiple-rung comment (KV STUDIO version 11.30 or later).
  - "Cancel line disable (W)" on the right-click menu
  - "ST/Script (S)" → "Cancel line disable (W)" on the menu
  - Ctrl + W

Point

To use an indirect reference (with * at the leading) for an Argument, if you write it after "(*", it will be recognized as a Comment. Add a 1-byte space between "(" and "*" so that it will not be recognized as a Comment.

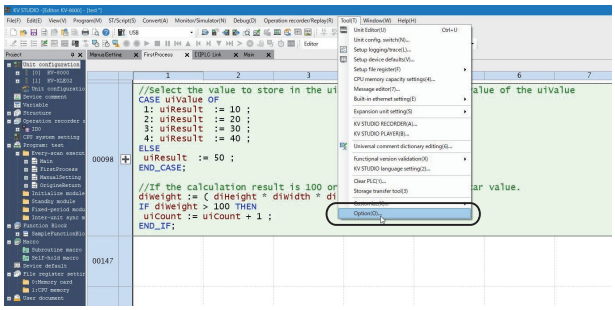
Precautions on Edit window and Customization

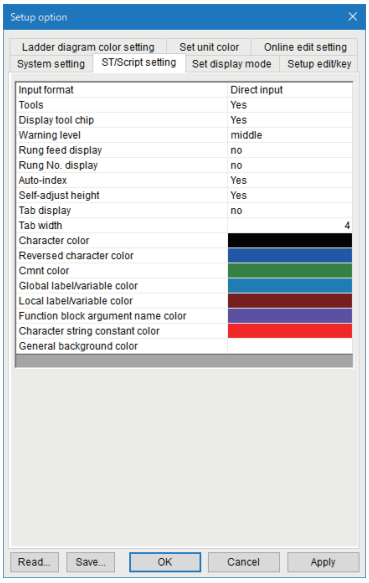
■Precautions on ST programming

- The maximum number of characters per line is 1024 characters (1-byte) including ST and Comment.
- Ladders with ST extended cannot be directly edited.

■Customizing in ST window

From Menu, select "Tool (T)" → "Option (O)" → "ST/Script setting" to customize in the Edit window.
For details, refer to "ST Language User's Manual"





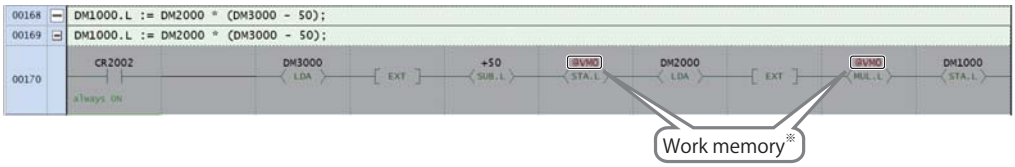
About ST/Script work memory (Work device)

Work memory (Work device)

Work memory (Work device) refers to the Device for temporarily string partial operation result such as complicated Operation expression. In the ST program, ST Language will automatically assign the Work memory from the area reserved for Module/Macro during ST conversion, so you can store without using additional Devices.

Program ST DM1000.L := DM2000 * (DM3000 - 50);

Ladder Convert

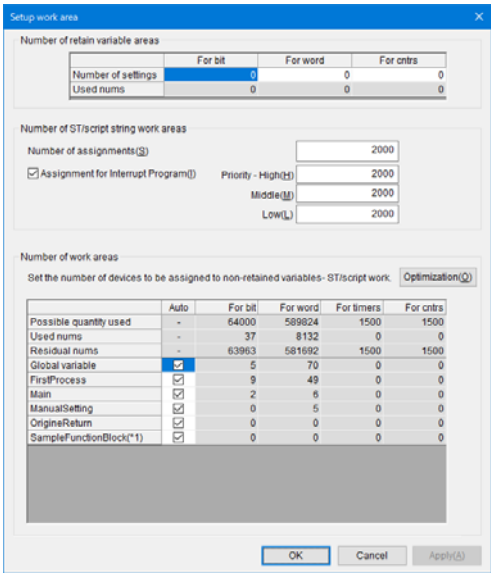


Description The result obtained by [DM3000-50] operation is stored in Work memory (@VM0) temporally, and then it is stored in left DM1000.L after multiplied by DM2000.

■Application area of Work memory

Work memory used for ST/Script can be checked via "Work Area Setting" dialog box.
"ST Language User's Manual"

Click [Work Area Setting (W)] to assign the work area for the Interrupt program.



XYM notation

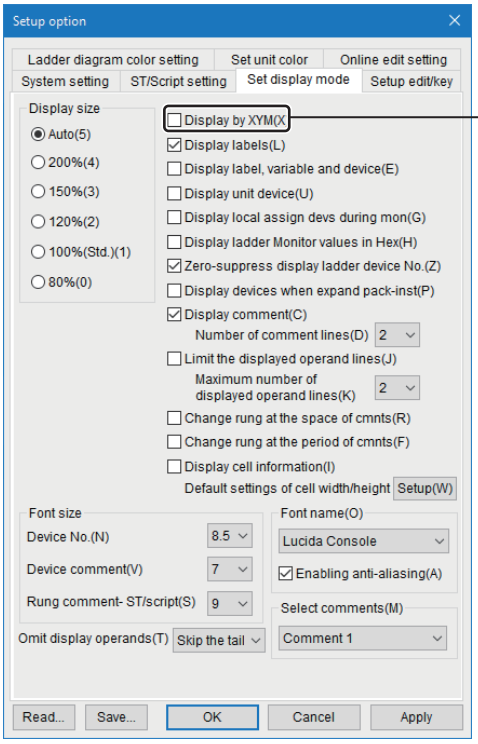
ST program can be written with XYM notation.

■Creating a program with XYM Notation

1 From Menu, select "Tool (T)" → "Option (O)".

"Option setting" dialog box is displayed.

2 Click the "Set display mode" tab.



Select the [Display by XYM (X)] check box and click OK.

3 Program with XYM notation.

```
IF D1000 > 20 THEN
 D0 := 50;
 SET (Y20);
END_IF;
```

Reference After selecting "Display with XYM", programming can be enabled with normal Device No. (mixable).

If XYM notation is canceled after editing the ST with XYM notation, an error will occur when the ST using X or Y is reconverted. For M and L, the Device No. changes to a CH notation (such as R) from a decimal. Please note that the Device No. of the conversion result may be changed.

Example) M100 := TRUE ;

The results of converting this ST to a M100 Ladder are:

- When using XYM notation, the ST is converted to MR604.
- When not using XYM notation, the ST is converted to MR100.

3

Data Processing

Devices. 3-2
Index modify and Indirect designation 3-25
Data Types (Suffixes) 3-31

Devices

Device refers to Member (Variable) used in internal Relay and Data memory, etc. of the CPU unit. It is usable to provide data used for operations and functions, and store Function execution results.

Device list

■Bit device

Device name	Range*1	Number of points	Main functions	When operation starts (Power ON, PRG → RUN)	Local device (@) *1	Index modify (:Z)	Indirect designation(*)*8	
							Reference	Store
Input relay*10	R000 to R199915*2	Total 32000 points	Imports ON/OFF info from peripherals.	Clear/Hold*3*12	○	○	○	×
Output Relay*10			Sends ON/OFF info to peripherals					
Internal auxiliary relay*10			Devices used in CPU and expansion unit					
Link relay	B00 to B7FFF*9	32768 points	Devices available only in CPU	Clear/Hold*3	×	○	○	×
Internal auxiliary relay	MR000 to MR399915	64000 points		Clear	○	○	○	×
Latch relay	LR000 to LR99915	16000 points		Hold	○	○	○	×
Timer (Contact)	T0 to T3999	4000 points	Contact devices of 10μs/1ms/10ms/100ms Down timer and 10ms Up/Down timer	Clear	○	○	○	×
Counter (Contact)	C0 to C3999	4000 points	Contact devices of Up counter and Up/Down counter	Clear/Hold*7	○	○	○	×
Control relay	CR000 to CR7915*4	1280 points	Devices for controlling PLC function or getting the state.	Hold	×	×	×	×

■Word device

Device name	Range*1	Number of points	Main functions	When operation starts (Power ON, PRG → RUN)	Local device (@) *1	Index modify (:Z)	Indirect designation(*)*8	
							Reference	Store
Data memory	DM0 to DM65534	65535 points	For string numerical data (16-bit)	Clear/Hold*3	○	○	○	○
Expansion data memory	EM0 to EM65534	65535 points		Clear/Hold*3	○	○	○	○
File register	Dial mode: ZF0 to ZF524287	524288 points		Clear/Hold*3	×	○	○	○
	Current bank : FM0 to FM32767	(32768×16 bank) points			○*11			
Link register	W0 ～ W7FFF*9	32768 points		Clear/Hold*3	×	○	○	○
Temporary memory	TM0 to TM511	512 points	For string numerical data (16-bit)	Clear	○	○	○	○
Timer (Current/Set value)	T0 to T3999	4000 points	Current/Set values of 10μs/1ms/10ms/100ms Down timer, and 10ms Up/Down timer (32-bit)	Current value: Clear*5 Current value: Hold	○	○	○	×
Counter (Current/Set value)	C0 to C3999	4000 points	Current/Set values of Up counter and Up/Down counter (32-bit)	Clear/Hold*7	○	○	○	×
Index register	Z1 ～ Z23*6	23points	Devices available for Index modify (32-bit)	Clear	×	×	×	×
Control memory	CM0 to CM7599*4	7600 points	Devices for controlling PLC function or getting the state. (16-bit)	Hold	×	×	×	×

*1 Since they are used as Local devices, the reserved number of points is limited by the usable range of Global devices. "Local device list" (page 3-4)
*2 R000 to R199915 can be specified as extended I/O Relay with Unit Monitor.
*3 Clear/Hold settings can be performed in the "Power Failure Holding" option.
*4 "CR/CM List" (page A-2)
*5 Up/Down timer (UDT) holds the Current value.
*6 System reserved Z11 to Z23 cannot used.
*7 Clear/Hold settings can be performed in the "Power Failure Holding" option. (only for the Contact and the Current value)
8 "Store" is a Device that can be written as "(Device)", while "Reference" is the Device that can be specified indirectly. "KV-8000 Series User's Manual", "Indirect designation"
Specify all Device numbers in HEX.
*9 Use "DR" notation for Directly Processing.
"KV-8000 Series User's Manual","About Direct Processing"
*10 Local device cannot be used for Bank switching.
*11 For Input relay, Clear/Hold cannot be set.
*12 Device list (for KV-8000)Macro operand device

■Function block argument

Argument type

Argument name/ Device		Description		Description	Local device	(@)	Index modify(:)
IN		Specified label		Receives a Device value or Constant as an Argument.	×	×	×
OUT		Specified label		Writes to the Device specified the Operation result.	×	×	×
IN-OUT		Specified label		Receives a Device as an Argument.	×	×	○
Unit	Leading R	UR	UR0 to UR7	Receives the leading Relay (R) assigned to the unit as an Argument.	×	×	○
	Leading DM	UM	UM0 to UM7	Receives the leading Data memory (DM) assigned to the unit as an Argument.	×	×	○
	No.	UV	UV0 to UV7	Receives the unit No. as an Argument.	×	×	×

■Constant

Constant	Range	Main functions
DEC	0 to 65535 (unsigned 16-bit) -32768 to +32767(signed 16-bit) 0 to 4294967295 (unsigned 32-bit) -2147483648 to +2147483647 (signed 32-bit)	Used for specifying DEC constant.
Single precision floating point real number	-3.4E+38 ≦ N ≦ -1.4E-45 N = 0 +1.4E-45 ≦ N ≦ +3.4E+38 (effective digits: 7 digits)	Used for specifying single precision floating point real number.
Double precision floating point real number	-1.79E+308 ≦ N ≦ -2.23E-308 N = 0 +2.23E-308 ≦ N ≦ +1.79E+308 (effective digits: 16 digits)	Used for specifying double precision floating point real number.
HEX (16#)	16#0000 to 16#FFFF (16-bit) 16#00000000 to 16#FFFFFFFF (32-bit)	Used for specifying HEX constant.
Fixed string (‘ ’)	Example) ‘Type ABC’, ‘MON/08:30:15’	Used to fixed string.

■Internal register

Device name	Range	Number of points	Main functions	When operation starts (Power ON, PROG -> RUN)
Internal register	–	1 point	Device for temporary data storage (16-bit/32-bit/64-bit)	Clear

Local device list

The initial values of the number and range of Local devices/Global devices occupied for creating a project are shown below. Number of reserved Devices can be changed from “CPU System Setting” → “Local All Assign Setting” on the “Work Space”.

Device name*1	Local device	Global device	Total
	Number of reserved Devices in whole project*2	Counts/Range	
Relay (R)*3	None	32000 points（2000 CH） R000 to R199915	32000 points（2000 CH）
Internal auxiliary relay (MR)*3	32000 points（2000 CH）	8000 points (500 CH) MR000 to MR199915	64000 points (4000 CH)
Latch Relay (LR)*3	8000points (500 CH)	8000 points (500 CH) LR000 to LR49915	16000 points(1000 CH)
Timer (T)*4	1500 points	2500 points T0 to T2499	4000 points
Counter (C)*4	1500 points	2500 points C0 to C2499	4000 points
Data memory (DM)	25000 points	40535 points DM0 to DM40534	65535 points
Expansion data memory (EM)	45000points	20535 points EM0 to EM20534	65535 points
Temporary data memory (TM)	None	512 points TM000 to TM511	512 points

*1 Link relay (B), Link register (W), File register (ZF), Control relay (CR), Control memory (CM), and Index register (Z) cannot be used as Local device.
@ When using @FMxxx, Local devices in workspace will be occupied.

*2 When newing a project, the device area used for Global devices can be batch occupied (The calculation is based on the assumption of using 10 Modules and calling Macro 50 times.)
The number of reserved Devices can be changed as required.

*3 The count can be set in “ch” unit.

*4 "@+ (Device No.)" can be used only when the Device No. is specified as an operand for Timer/Counter instructions
(Example of entering Mnemonic: TMR @0 #100).

Point

- **If Relay (R) and Temporary Data memory (TM) are used as Local device, the number of Relays and Data memories must be specified.**
- **The following Devices must be used as Global devices:**
  - **Device assigned to expansion unit**
  - **Device controlled by peripheral such as a touch panel**
  - **Common device used for whole project**
  - **Device for inter-module data transfer**
- **Where possible, Local devices should be coded from 0 and use consecutive numbers.**
- **Since R and DM are assigned to expansion units, they need space to be used as Global devices. Prioritize MR and EM for Local devices.**
- **If Local devices or Global devices is used outside the specified range, a conversion error will occur.**
- **When Index device or Indirect designation is used to specify outside of the usable range of Local devices or Global devices, an operation error will occur.**
- **When using various instructions that specify the leading Device as the operand, the Index modify or the Indirect designation, the available range of the Local devices and Global devices should not be exceeded.**

Device notation

● Index modify writing (modifying by Index register/Index constant)

• Modifying by Index register

Enter Z□ (□: numerical value 1 to 10) after the Devices (R(DR)/MR/LR/B/T/C/DM/EM/FM/ZF/W/TM/*DM/*EM/*FM/*ZF/*W/*TM).

Example

DM0:Z0If the value of Z1 is 100, it refers to DM100.
R1000:Z0If the value of Z1 is 100, it refers to R1604.

• Modifying by Index constant

Enter Z□ (□: numerical value 1 to 10) after the Devices (R(DR)/MR/LR/B/T/C/DM/EM/FM/ZF/W/TM/*DM/*EM/*FM/*ZF/*W/*TM).

Example

DM0:#1	**(It refers to DM1)**
R1000:#1	**(It refers to R1001)**

• Modifying by Operation expression

Enter the Operation expression after the Devices (R(DR)/MR/LR/B/T/C/DM/W/TM/CM/*DM/*W/*TM).

Example

DM0:(DM100 + 20)	**If the value of DM100 is 50, it refers to DM70**
R1000:(DM0 * 4)	**If the value of DM0 is 2, it refers to R1008**

Point

Index modify by Operation expression can only be used in ST/Script.

● Writing Indirect designation (*)

Enter * in front of Devices (DM/EM/FM/ZF/W/TM). Use it together with Indirect designation Command.

Example

＊DM0　＊TM100

"Indirect designation" page 3-27

● Writing Local device

Enter @ in front of the Devices (R/MR/LR/T/C/DM/EM/FM/TM).

These Devices are only valid in Modules and Macros. Indirect designation and Index modify are also available.

Example

＠DM0　＠R10000

● XYM notation

The available designations of Devices (R(DR)/MR/LR/DM/EM/FM) by XYM notation are listed below.

Device name	KEYENCE notation	XYM notation	Device No. by XYM notation
Relay	R(DR)	X(DX)	0000 to 999F
		Y(DY)	0000 to 999F
Internal I/O relay	MR	M	00000～63999
Latch relay	LR	L	0000～15999
Data memory	DM	D	00000～65534
Expansion data memory	EM	E	00000～65534
File register	FM	F	00000～32767

To use XYM notation, select “Tool” → →“Option” from Menu of “ST Language” and put a check on “XYM Notation” in “Display Mode Setting”.

● Writing the Bit Specify of Word device

Enter □ (□: the value of 0 to 15) following the Devices (R(DR)/MR/LR/B/T/C/DM/EM/FM/ZF/W/TM/CM).

Example

DM0.0 (it refers to the 1st bit of DM0)
EM10.4 (it refers to the 5th bit of EM10)

"Processing Word device as Bit device"（page 3-20）

● Suffix writing

Enter.□(□:U,S,D,L,F,DF,B,T) following the Devices
(R(DR)/MR/LR/B/T/C/DM/EM/FM/ZF/W/TM/CM/*DM/ *EM/*FM/*ZF/*W/*TM).

Example

DM0.F (calculates values of DM0 to DM1 as the floating point data)
R1000.U (calculates as 16-bit data from R1000 to R1015)

Point

- **Suffix writing can be used only in ST/Script.**

● Specifying the constant value type

With the ST Language, you can set the range that the constant can hold by adding a suffix to the decimal.

Type	Suffix	Data range	Program example
Unsigned 16-bit	UINT#	0～65535	UINT#910, UINT#505
Signed 16-bit	INT#	-32768～+32767	INT#23, INT#-600
Unsigned 32-bit	UDINT#	0～4294967295	UDINT#1234567, UDINT#5
Signed 32-bit	DINT#	-2147483648～+2147483647	DINT#123, DINT#-98335963
Single precision floating point type (32-bit)	REAL#	－3.4E＋38≦N≦－1.4E－45 N＝0 ＋1.4E－45≦N≦＋3.4E＋38	REAL#100, REAL#-3.4E38, REAL#0.12
Double precision floating point type (64-bit)	LREAL#	－1.79E＋308 ≦N≦－2.23E－308 N＝0 ＋2.23E－308≦ N≦＋1.79E＋308	LREAL#100, LREAL#-1.7E123, LREAL#0.0256

Bit device

■Bit state

The following notation can be used to indicate the state of a Bit device.

Bit state	Notation	Program example
ON (TRUE)	1、TRUE	MR1001 := 1 ;、MR1002 := TRUE ;
OFF (FALSE)	0、FALSE	MR2001 := 0 ;、MR2002 := FALSE ;

Reference These 2 states are called "Boolean value".

I/O Relay R

Overview Input relay is the Device that imports ON/OFF information from a peripheral equipment into PLC.
Output Relay is the Device that sends ON/OFF instructions or operation results from PLC to PLC peripheral equipment.

Notation Device No. configuration is as follows.

Example)

Channel No. (000 to 1999) Contact No. (00 to 15)

R00012

Device name Device No. (00000 to 199915)

Reference

・Writable by Zero suppressing.
Example) R12
・Writable by adding a suffix.
Example) R12.B
・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.
・Processed as Word device.
Example) R2000.U(16 bits occupied from R2000)
・If adding "D" to the Device name, it will operate as a Direct I/O Relay.
Example) DR500 (R500 operates as a Direct Output Relay)

"Direct I/O Relay"(page 3-21)

Range ・R000 to R199915

Usage Input Relay
・Available peripheral equipment connected include buttons and switches (such as the selector switch/limit switch/photoelectric switch/digital switch), etc.

Output Relay
・Available peripheral equipment connected include solenoids, electromagnetic switches, and indicators (traffic lights, digital displays), etc.

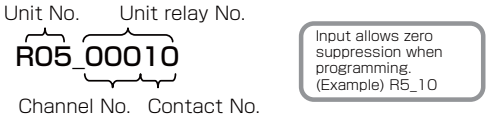
Details
・1-bit of Input relay is assigned to one input terminal.
・1-bit of Output Relay is assigned to one output terminal.
・Both'a'Contact (Normal Open) and'b'Contact (Normal Close) can be specified.
・In one project, the Input relay with the same Device No. can be written for multiple times.

Default By default, the state of Output Relay is cleared and turns off when the power ON and PROG → RUN.

Reference

・About the notation for Unit device of Relay (R)

In "ST Language", in addition to the normal notation, Relay notation based on the unit No. is available. To use Unit device Notation, select "View (V)" → "Unit device Notation (Y)" from Menu.



Unit No. : Units are assigned starting from CPU in the order of connection, such as 00, 01, 02, 03, 04, and so on.

Unit Relay No. : Number of Relay points occupied from 00000 is assigned as the Relay No.

Unit No.	00	01		02	03	04	05
KV-PU1	CPU unit	KV-SIR32XT 32-input 32-output	KV-7000C	KV-B16XC 16-input	KV-C32TD 32-output	KV-C32TXD 32-input 32-output	KV-C64TD 64-output
Relay No.		0000~0315		0000~0015	0000~0015	0000~0315	0000~0315

Link relay B

Overview It is used as a Link relay when connecting via EtherNet/IP or FL-net, etc. Unlike other relay, specifies the Device No. in hexadecimal notation.

Notation Device No. configuration is as follows.

Example)

Reference

・Writable by Zero suppressing.

Example) B7A

・Writable by adding a suffix.

Example) B7A.B

・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

・Processed as Word device.

Example) B80.U (16 bits occupied from B80)

Point

Use HEX (0 to F) to specify.

Local device (@) cannot be used.

Range ・B0000 to B7FFF

Usage It is used as a Link relay when connecting via EtherNet/IP or FL-net, etc.

Description

・Cannot be assigned to the Unit device of the expansion unit.

・In one project, the Link relay B with the same Device No. can be written for multiple times.

Default By default, the Link relay state is cleared and turns off when the power ON and PROG → RUN. The state can be held by [Power Off Hold Setting] in the CPU System Setting.

Internal auxiliary relay (R/B/MR/LR)

■R

Overview R unassigned to actual I/O Relay and Expansion Unit Relay is used as the Internal auxiliary relay.

Description Device No. configuration is as follows.

Example)

Channel No. (000 to 1999) Contact No. (00 to 15)

R00012

Device name Device No. (00000 to 199915)

Reference

・Writable by Zero suppressing.

Example) R12

・Writable by adding a suffix.

Example) R12.B

・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

・Processed as Word device.

Example) R2000.U(16 bits occupied from R2000)

Range ・R000 to R199915 (can be assigned as the expansion input unit No.)

■B

Overview When EtherNet/IP or FL-net is not connected, B that is not assigned to Link relay can be used as the Internal auxiliary relay.

Description Device No. configuration is as follows.

Example)

Channel No. (000 to 7FF) Contact No. (0 to F)

B1EE7

Device name Device No. (0000 to 7FFF)

Reference

・Writable by Zero suppressing.

Example) B7A

・Writable by adding a suffix.

Example) B7A.B

・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

・Processed as Word device.

Example) B80.U (16 bits occupied from B80)

Point

Use HEX (0 to F) to specify.

Local device (@) cannot be used.

Range B0000 to B7FFF

Description

・Cannot be assigned to the Unit device of the expansion unit.

・In one project, the Link relay B with the same Device No. can be written for multiple times.

Default By default, the state is cleared and turns off when the power ON and PROG → RUN. The state can be held by [Power Off Hold Setting] in the CPU System Setting.

■MR

Overview The device that can be used only inside the CPU.

Notation Device No. configuration is as follows.

Example)

Channel No. (000 to 3999) Contact No. (00 to 15)

MR02809

Device name Device No. (00000 to 399915)

Reference

・Writable by Zero suppressing.

Example) MR2809

・Writable by adding a suffix.

Example) MR2809.B

・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

・Processed as Word device.

Example) MR2900.U (16 bits occupied from MR2900)

Range MR00000 to MR399915

Point

MR200000 to MR399915 cannot be used because they have been assigned to the Local device.
(The reserved area can be changed from "Local All Assign Setting" in the CPU System Setting)

Description

・Cannot be assigned to the Unit device of expansion unit.

・In one project, the Relay MR with the same Device No. can be written for multiple times.

Default By default, the state is cleared and turns off when the power ON and PROG → RUN.

■LR (Latch Relay)

Overview The State-hold device that can only be used inside the CPU.

Notation Device No. configuration is as follows.

Example)

Channel No. (000 to 999) Contact No. (00 to 15)

LR02308

Device name Device No. (00000 to 99915)

Reference

・Writable by Zero suppressing.

Example) LR2308

・Writable by adding a suffix.

Example) LR2308.B

・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

・Processed as Word device.

Example) LR2400.U (16 bits occupied from LR2400)

Range LR00000 to LR99915

Description

・Cannot be assigned to the Unit device of expansion unit.

・In one project, the Latch Relay LR with the same Device No. can be written for multiple times.

Default By default, holds the state before the last power OFF.

Timer (Contact) T

Overview Timer refers to a Device with Contact (1 bit) and Current value (32 bits). When Current value of Down timer matches 0 (time up), the Contact turns on.

Notation Device No. configuration is as follows.

Example)

Device No. (0000 to 3999)

T0501.B

Device name Suffix

Reference

・Writable by Zero suppressing.

Example) T501.B

・Writable by adding a suffix. When used as the Timer contact, add a suffix.B; when processing the Current value of Timer, add a suffix.D.

・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

Range T0000.B to T3999.B

Description Four Timer Function types are available: 10μs Timer/1ms Timer/10ms Timer/100ms Timer.

Default By default, the state is cleared and turns off when the power ON and PROG → RUN.
Current value is identical to Set value when the power ON and PROG → RUN.
The Set value is held.

■How to program when operating the Timer Function with a ST

Program the Timer operation by specifying the execution condition of the used Timer (Timer Reset Relay), Timer No., and Set value (the time before time up) in each Argument of Timer Function.

Example)

Timer device (T0000 to T3999) Set value (0 to 4294967295, Device No.)

TMR (R1000, T501, 1000)

Timer function name (100ms down timer) Timer execution condition

[Difference in operation of each Timer]

Timer type (Function)	Device No.	Set value	Time before ON
100ms Down timer (TMR)	T0 to T3999	0 to 4294967295	n×100ms
10ms Down timer (TMH)			n×10ms
1ms Down timer (TMS)			n×1ms
10μs Down timer (TMU)			n×10μs

Reference Device (unsigned 32-bit data) or Expression can be specified as the Set value. "Timer/Counter Function"
「TMR function」(page 5-13)

■Timer error

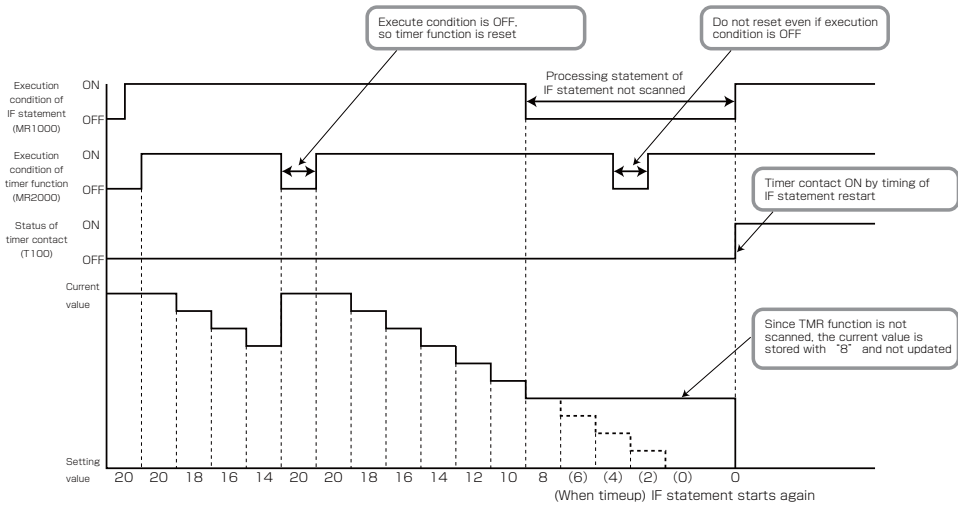
The Timer error is as follows:

TMR(100ms Timer) :within± (100ms+1 scan time)
TMH(10ms Timer) :within± (10ms+1 scan time)
TMS(1ms Timer) :within± (1ms+1 scan time)
TMU(10μs Timer) :within± (10μs+1 scan time)

■Precautions on using the Timer Function

When using Timer Function in IF statement or Box ST, the Current value of Timer will be not updated if IF statement or Box ST stops during Timer Function execution.
If the time measurement is continued and each instruction executed again, the elapsed time will be reflected in the Current value of Timer.
If reaching the set time, Timer contact turns on rapidly after executed again.

ST Program
```
IF MR1000 = TRUE THEN
   TMR(MR2000 = TRUE, T100, 200);
END_IF;
```



■How to process Set/Current value of Timer

The table below shows how to read and change the Set/Current value of Timer in the Ladder with a ST.

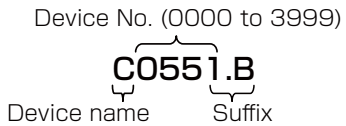
		How to read	How to change
Current value		Storage data: = CURVAL (timer data); Example) udiValue1: = CURVAL (timer);	Current value cannot be changed. Initialize it with TMR Function etc.
Set value	Device Specify	Storage DM= Set value Storage device	Set value Storage device = Value to change (Constant/ Device)
	Constant	Set value cannot be read.	Set value can't be changed.

Point **If Current value > Set value during change, the Current value will be changed to the same value as the Set value.**

Counter (Contact) C

Overview Counter refers to a device with Contact (1 bit) and Current value (32 bits).
When Current value identical to Set value (count up), the Contact turns on.

Notation Device No. configuration is as follows.
Example)



Reference
・Writable by Zero suppressing.
Example) C551.B

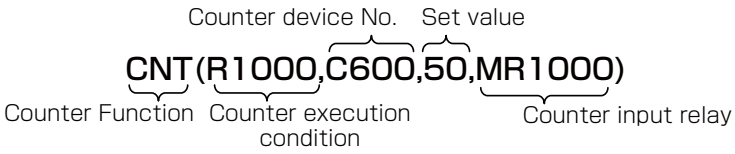
Writable by adding a suffix. When used as the Timer contact, add a suffix.B; when processing the Current value of Timer, add a suffix.D.

・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

Range C0000.B to C3999.B

■How to program when operating the Counter Function with a ST

Program the Counter operation by specifying the execution condition of the used Counter (Counter Reset Relay), Counter No., Set the value and Counter Input relay in each Counter Function Argument.



The Counter Function adds the Current value by 1 when the Counter input relay turns from OFF to ON (differential type), and turns on the Counter contact when Current value reaches Set value.

Reference Device (unsigned 32-bit data) or Expression can be specified as the Set value. "Timer / Counter Function"
"CNT script" (page 5-16)

■How to process Set/Current value of Counter

The table below shows how to read and change the Set/Current value of the Counter in the Ladder with a ST.

		How to read	How to change
Current value		Storage data: = CURVAL (counter data); Example) udiValue1: = CURVAL (counter);	Current value cannot be changed. Initialize it with CNT Function.
Set value	Device Specify	Storage DM= Set value Storage device	Set value Storage device = Value to change (Constant/ Device)
	Constant	Set value cannot be read.	Set value can't be changed.

Point **High-speed counter comparator (contact) CTCHigh-speed counter 3**

Control relay (Contact) CR

Overview Control relay is an Internal auxiliary relay to control PLC function and acquire PLC state.

Range CR0000 to CR7915

Reference
・Writable by adding a suffix.
Example) CR2002.B
・When writing without adding a suffix, the type will be processed as .B (Bit type) unless specified.

Description "CR/CM List"(page A-2)

Word device

Data memory/Expansion data memory DM/EM/FM

Overview Data memory and Expansion data memory are Devices with 16 bits per point for string the data referenced and the operation result while the Function is operating.

Notation Device No. configuration is as follows.
Example)

Reference

・Writable by Zero suppressing.

Example) DM6143

・Writable by adding a suffix.

Example) DM1000.S (signed 16-bit), DM2000.D (Unsigned 32-bit)

・When writing without adding a suffix, the type will be processed as .U (signed 16-bit) unless specified.

・Processed as a Bit device.

Example) Processes the lowest bit of DM1100.B DM1100

・Processed as a Bit device whose bit position is specified.

Example) Processes the 13th bit of DM1200.12 DM1200

Range

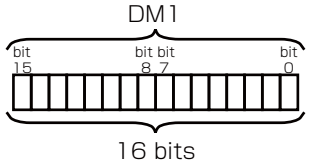
・DM00000 to DM65534
・EM00000 to EM65534
・FM00000 to FM32766

Description

・Only Data memory (DM) can be assigned as the Unit device of an expansion unit.

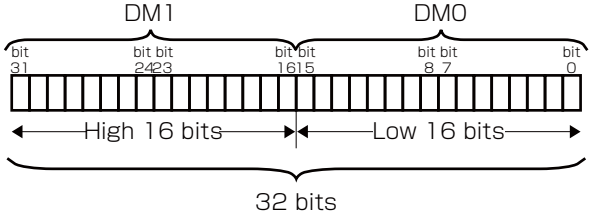
●When processing 16-bit data

- Bit size of one Device is 16-bit in which 0 to 65535 can be processed for unsigned DEC, -32768 to +32767 for signed DEC, and 0 to FFFF for HEX.
- When processing16-bit data, add a suffix .U (can be omitted) or .S to Device No.



●When processing 32-bit data

- You can also use two Devices to process 32-bit data. You can also use two Devices to process 32-bit data. For 32 bits, the lower 16 bits are stored in the Data memory of the specified Device No., and the upper 16 bits are stored in the Data memory of the next Device No. It can process values from 0 to 4294967295 for unsigned DEC, values from -2147483648 to +2147483647 for signed DEC, and values from 0 to FFFFFFFF for HEX.
- When processing 32-bit data, add a suffix .D or .L to Device No.



Default

By default, the value is held.

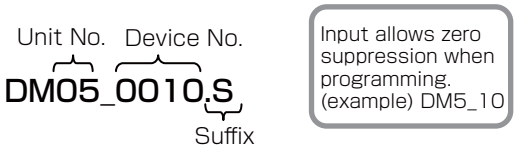
・The state can be cleared by setting [Power Off Hold Setting] in the CPU System Settings.

Reference

・About the notation for Unit device of Data memory (DM)

In "ST Language", in addition to the normal notation, the notation based on the Unit No. is available.

To use the Unit device notation, select "View (V)" → "Unit Device Notation (Y)" from Menu.



Unit No.:	Units are assigned starting from CPU in the order of connection, such as 00, 01, 02, 03, 04, and so on.
Unit DM No.:	Assign the occupied Data memory from 0000.
Suffix:	Writable by adding a suffix. If the suffix is omitted, processed as unsigned 16-bit (.U).

Example)

Unit No.	00	01	02	03		04	05
KV-PU1	CPU unit 230 words	KV-SAD04 60 words	KV-SDA04 40 words	KV-SSC02 70 words	KV-7000C	KV-TP40 90 words	KV-TP40 90 words
DM No.	000～229	000～059	000～039	000～069		000～089	000～089

The 5th of KV-SAD04 ..... DM01_0005
The 10th of KV-TP40 (Unit No. 04) ..... DM04_0009
The 2nd of KV-TP40 (Unit No. 05) ..... DM05_0001

File register ZF/FM

Overview There are two types of File register: ZF, which processes all areas in Serial No. mode, and FM, which divides an area into multiple (16 or 4) banks to process them in Bank Switch mode.
For details, refer to "File register" in the User's Manual of the used CPU unit.

Notation Device No. configuration is as follows.

Example)

Device name Device No.

ZF0372

Reference

・Writable by Zero suppressing.

Example) ZF372

・Writable by adding a suffix.

Example) ZF1000.S (signed 16-bit), FM12300.D (Unsigned 32-bit)

・When writing without adding a suffix, the type will be processed as .U (signed 16-bit) unless specified.

・Processed as a Bit device.

Example) Processes the lowest bit of DM1100.BDM1100

・Processed as a Bit device whose bit position is specified.

Example) Processes the 13th bit of ZF1200.12ZF1200

Point

Local device of ZF (@ZFxxx) cannot be used.
Local device of FM (@FM0 to @FM32767) does not support Bank Switch.
Local device of FM (@FM0 to @FM32767) uses Work memory.

Range

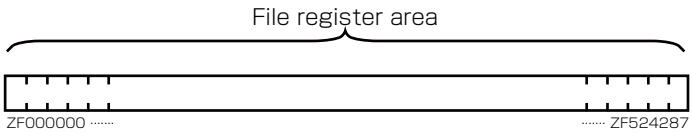
・ZF000000 to ZF524287
・FM00000 to FM32767
"Data memory/Expansion data memory DM/EM/FM"（page 3-13）

Description

・Cannot be assigned to the Unit device of expansion unit.
・The type and size of Device differ between Serial No. mode and Bank Switch mode.

●ZF: In the case of File register is processed in Serial No. mode

Processes whole area of File register as one Device.



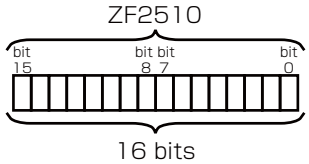
●**FM: In the case of File register is processed in Bank Switch mode**

Divides File register area into 16 banks and processes by switching banks.

One Device can be used as 16 different Devices via switching banks.
However, pay attention to the processing data which can be used in duplicate with the ZF device.
For details on Bank Switch, refer to "File register" in the User's Manual of the used CPU unit.

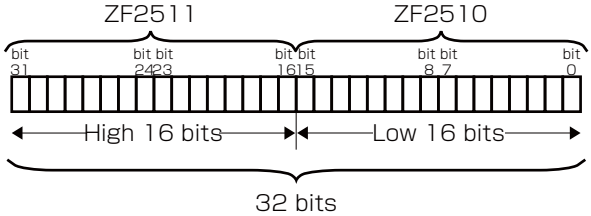
●**When processing 16-bit data**

- Bit size of one Device is 16-bit in which 0 to 65535 can be processed for unsigned DEC, -32768 to +32767 for signed DEC, and 0 to FFFF for HEX.
- When processing16-bit data, add a suffix .U (can be omitted) or .S to Device No.



●**When processing 32-bit data**

- You can also use two Devices to process 32-bit data. You can also use two Devices to process 32-bit data. For 32 bits, the lower 16 bits are stored in the Data memory of the specified Device No., and the upper 16 bits are stored in the Data memory of the next Device No. It can process values from 0 to 4294967295 for unsigned DEC, values from -2147483648 to +2147483647 for signed DEC, and values from 0 to FFFFFFFF for HEX.
- When processing 32-bit data, add a suffix .D or .L to Device No.



Default By default, the value is held.

・The state can be cleared by setting [Power Off Hold Setting] in the CPU System Settings.

Link register W

Overview It is used as a Link register when connecting via EtherNet/IP or FL-net. Unlike other Word devices, specifies the Device No. in hexadecimal notation.

Overview Device No. configuration is as follows.

Example)

Device name Device No. (0 to 3FFF)

W3A6F

Reference

・Writable by Zero suppressing.

Example) W7A

・Writable by adding a suffix.

Example) W7A.S (signed 16-bit), W3E76.D (Unsigned 32-bit)

・When writing without adding a suffix, the type will be processed as .U (unsigned 16-bit) unless specified.

・Processed as a Bit device. Bit position can be also specified.

Example) Processes the 13th bit of W1200.12W1200

Point

Use HEX (0 to F) to specify.
Local device (@) cannot be used.

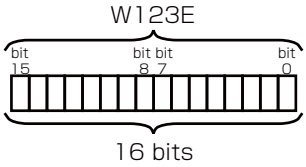
Range W0000 to W7FFF

Usage It can be used as a Link relay when connecting via EtherNet/IP or FL-net, etc.

Description Cannot be assigned to the Unit device of expansion unit.

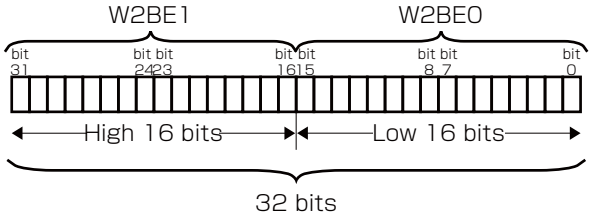
●When processing 16-bit data

- Bit size of one Device is 16-bit in which 0 to 65535 can be processed for unsigned DEC, -32768 to +32767 for signed DEC, and 0 to FFFF for HEX.
- When processing16-bit data, add a suffix .U (can be omitted) or .S to Device No.



●When processing 32-bit data

- You can also use two Devices to process 32-bit data. For 32 bits, the lower 16 bits are stored in the Data memory of the specified Device No., and the upper 16 bits are stored in the Data memory of the next Device No. It can process values from 0 to 4294967295 for unsigned DEC, values from -2147483648 to +2147483647 for signed DEC, and values from 0 to FFFFFFFF for HEX.
- When processing 32-bit data, add a suffix .D or .L to Device No.



Default By default, the value is held.
The state can be cleared (release) by setting [Power Off Hold Setting] in CPU System Setting.

Temporary Data memory (TM)

Overview Temporary Data memory is a 16-bit Word device per point that can be used as a Data memory in which some Device areas are also used to temporarily store the operation data or the operation result data.

Notation Device No. configuration is as follows.

Example)

Reference

・TM000 to TM003 are used as operation instructions such as DIV and MUL in the Ladder, and TM000 and TM001 are used for processing data exceeding 32 bits. It is recommended not to use them for any other purpose.

・Writable by Zero suppressing.

Example) TM10

・Writable by adding a suffix.

Example) TM10.D (Unsigned 32-bit)

・When writing without adding a suffix, the type will be processed as .U (unsigned 16-bit) unless specified.

Range TM000 to TM511

TM No.	Application
TM000･TM001	Used for 64-bit data
TM002･TM003	Used to store the remainder of division
TM004 to TM511	Common*

*Simple Indirect designation cannot be used for ST.

Default By default, the value is cleared to zero when the power ON and PROG → RUN.

Timer (Current value) T

Overview Processes Current value by adding a suffix .D to the Timer device.

Range T0000.D to T3999.D

Reference When processing Timer device as the Current value, suffix.D cannot be omitted. When writing without adding a suffix, the type will be processed as .B (Timer contact) unless specified.

Description Current value of Timer device can be changed within 0 to 4294967295.

Point **If Current value > Set value after Current value changed, the Current value will be changed to the same value as the Set value.**

"Timer (Contact) T"（page 3-11）

Counter (Current value) C

Overview Processes Current value by adding a suffix .D to the Counter device.

Range C0000.D to C3999.D

Reference When processing Counter device as the Current value, suffix.D cannot be omitted. When writing without adding a suffix, the type will be processed as .B (Counter contact) unless specified.

Description Current value of Counter device can be changed within 0 to 4294967295.

Point **If Current value > Set value after Current value changed, the Current value will be changed to the same value as the Set value.**

"Counter (Contact) C"（page 3-12）

Index register Z

Overview Word device with 32 bits per point. It is used for specifying the number to be added to the object Device No. during Index modify. Like the Data memory, it can be used to store data and operation results.

Range KV-8000 with CPU function version 2.0 or later: Z1 to Z23
Models other than the above: Z1 to Z12

Reference
・Writable by adding a suffix.
Example) Z01.L (signed 32-bit)
Z02.D (unsigned 32-bit)
・When writing without adding a suffix, the type will be processed as .L(signed32-bit) unless specified.

Point

- **Z11 are reserved for system and cannot be used.**
- **.F, .DF, .T, .B cannot be specified as the suffix.**
- **Bit cannot be specified for Word device.**
- **No Local devices. Use the Function ZPUSH and ZPOP.**

"ZPUSH"(page 6-7)
"ZPOP"(page 6-8)

Default By default, the value is cleared to zero when the power ON and PROG → RUN.

"Index modify"(page 3-25)

Control memory CM

Overview Control memory is a specific Word device to control PLC function and acquire the PLC state.

Range CM0000 to CM7599

Reference

・Writable by adding a suffix.
Example) CM1720.U
・When writing without adding a suffix, the type will be processed as .U (unsigned 16-bit) unless specified.

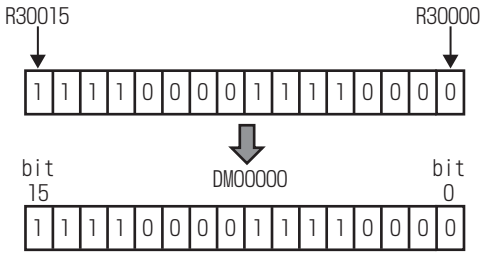
Description "CR/CM List"(page A-2)

Processing Bit device in word units

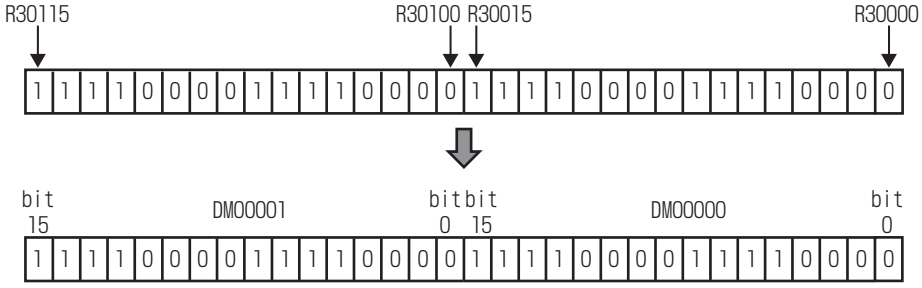
If the Bit device with a suffix other than .B is specified as the Function/Assignment statement which uses Word device, it will be processed in word units starting with the specified Bit device.

Example)

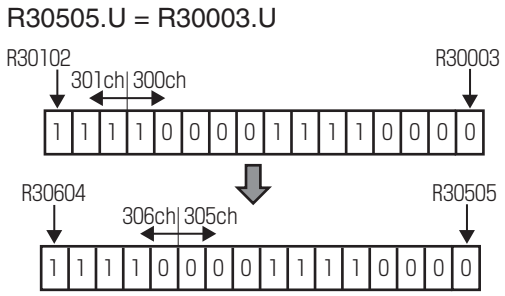
DM0000.U = R3000.U



DM0000.D = R3000.D



If the Bit device with a suffix is not the leading of a channel, it is processed as a word unit straddled the next channel.



Processing Word device as Bit device

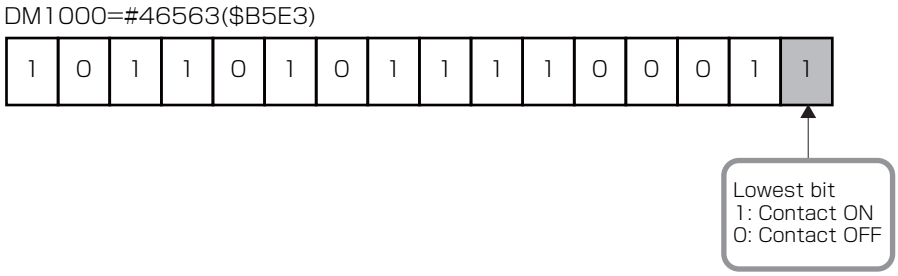
Normally, Word device is processed as the 16-bit/32-bit device, but it can be processed as a Bit device by adding Bit type suffix.

Point **Index register (Z) cannot process the Bit device.**

● Processing as lowest-bit-positioned Bit device

If a .B suffix is added to a Word device, or when a Word device is specified in the Argument or Assignment statement of a Contact/Output Function that processes a Bit device, the Word device will be processed as a lowest-bit-positioned Bit device.

When DM1000 value is 46563 (B5E3)



The lowest of DM1000 is 1, DM1000.B is processed as "Contact ON".

● Processing as random-bit-positioned Bit device

When the Bit position of ".0 to .15" is specified in a Word device, the Word device will be processed as a random-bit-positioned Bit device.

Notation Example)

DM1234.12

Random word device Bit position (.0 to .15)

Reference
- DM123.B and DM123.0 processes the same Bit.
- For Device comments, the comments set for the Word device are displayed.

Point
Use DEC (0 to 15) to specify the Bit position.
Local device (@) cannot be used.
Devices whose Bit can be specified are "DM, W, EM, FM, ZF, TM, CM" only.
Variables whose Bit can be specified are limited to: Variables whose data types are "INT, DINT, UINT, UDINT", elements of Array type variables set for these data types, and members of Structure type variables.
Indirect Reference and Index modify cannot be performed for Bit-positioned Devices.

When the value of DM123 is 59 (3B)

bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1

When DM123 is processed as a Bit device by positioning the Bit, the state of each Bit is as follows.

Device	Bit state
DM123.0	ON
DM123.1	ON
DM123.2	OFF
DM123.3	ON
DM123.4	ON
DM123.5	ON
DM123.6 to DM123.15	OFF

Direct I/O Relay

I/O relay (R), I/O relay (R) operates as a Direct I / O Relay by adding "D" to the leading of the Device No.

Notation Example)

Direct I/O relay

DR001

I/O relay

Point

- **Only the suffix .B can be added.**

Range I/O Relay

Variable

Variable refers to a programmatic element with a defined data type and name. ST Language allows you to use Devices or Variables for programming.
Unlike Devices, Variables can be programmed with the names defined by ST Language, and the defined Variables are automatically assigned to the work area in CPU unit. Therefore, there is no need to create a Device Map for managing the Device No. assigned functions. In addition, VT5 Series and Soft-VT also support Variables, there is no need to change the Device No. on the touch panel according to the Device No. on the PLC.

Point

Variables are only available in ST Language version 11.0 or later. The supported models are only KV-8000 with CPU function version 2.0 or later.
Variables can be read and written from the touch panel only in VT STUDIO version 8.0 or later. The supported model is limited to VT5 Series and the PLC model KV-8000 (Ethernet).

For details on how to set Variables, refer to "Variable Programming and Arrays/Structures"(page 2-15).

Variable/Data type list

To use Variables, it is necessary to define the data type in advance. The CPU unit automatically assigns the necessary data size to the work area according to the defined data type*1.
The work area sizes and Variable specifications that can be used with “KV-8000” are as follows.

Item		Details
Work area size	Bit	64000 points
	Word	589824 words
	Timer	4000 points
	Counter	4000 points
Variable specification	Definable count	100000
	Maximum length of Variable name	128 characters

*1 You can manually assign Devices to the CPU unit but not the work area. For details, refer to "KV-8000 Series User Manual".

The data type and data size of the Variables that can be used with “KV-8000” are as follows.

Data type		Data size (in bits)
Type name	Description	
BOOL	Bit	1
UINT	1-word unsigned integer	16
INT	1-word signed integer	16
UDINT	2-word unsigned integer	32
DINT	2-word signed integer	32
REAL	Single precision floating point type	32
LREAL	Double precision floating point type	64
STRING	Character string	(number of characters + 1) × 8 rounded up to an integral multiple of 16
Timer	Timer	—
Counter	Counter	—
Array	—	Data type × Array size
Function block	—	Based on definition*1
Function block	Function block	Based on definition*1

*1 Structures and Function blocks are data types that use multiple data types. Therefore, the data size is determined by the settings. For details, refer to "KV-8000 Series User Manual".

Constant

The constant and its range will vary with the Function used and the suffix of Device as well as the Variable type.

DEC constant

Overview Values expressed in 16-bit and 32-bit are shown in DEC notation.

Notation DEC value is expressed as it is. (signed notation is also available)

Example)

Value

06143

Reference ・Writable by Zero suppressing.

・if signed, only "+" can be omitted.

Range

・For unsigned 16-bit: 0 to 65535
・For unsigned 32-bit: 0 to 4294967295
・For signed 16-bit: - 32768 to + 32767
・For signed 32-bit: - 2147483648 to + 2147483647

Reference

Adding a suffix to the constant when writing expands the range of constant. (Suffixing is not required to constants normally)

UINT#300 :16-bit, unsigned (0 to 65535)
INT#500 :16-bit, signed (- 32768 to + 32767)
UDINT#3000 :32-bit, unsigned (0 to 4294967295)
DINT#5000 :32-bit, signed (- 2147483648 to + 2147483647)
REAL#123 :Single precision floating point type(-3.4E+38≦N≦1.4E-45
N＝0
1.4E-45≦N≦3.4E+38)
(Number of effective digits: approx. 7 digits)
LREAL#1.2345 : Double precision floating point type(-1.79E+308≦N≦-2.23E-308
N＝0
2.23E-308≦N≦1.79E+308)
(Number of effective digits: approx. 16 digits)

HEX constant

Overview

Values expressed in two bits range of 16-bit and 32-bit are shown in HEX notation.

Notation

To write by entering "16 #" at the beginning, followed by the HEX notation value.

Example)

HEX symbol Value
16#1AF7

Range

・ For 16-bit: 16#0 to 16#FFFF
・ For 32-bit: 16#0 to 16#FFFFFFFF

Point

- **If you write a HEX that contains A to F without adding "16 #", an error will occur. If it does not contain A to F, it will be recognized as a DEC.**
- **HEX constants cannot be signed.**

Single precision floating point type constant

Overview

It can be written as a single precision floating point real number.

Notation

・Add a sign and write the value in DEC notation as it is.

Example)

Value
1234.56

・Expressed in exponential notation with 8 digits or more.
・DEC constant is expressed with adding a suffix.

Example)

Value Single precision floating point type
1234F

Reference It is also possible to express without adding a suffix.

Example) **1234.0**

Indicates single precision floating point type

Range

-3.4E+38≦N≦-1.4E-45

N=0

1.4E-45≦N≦+3.4E+38 (number of effective digits: approx. 7 digits)

Double precision floating point type constant

Overview It can be written as a double precision floating point type real number.

Notation

- Add a sign and write the value in DEC notation as it is.

Example) Value: **12345.6789**

- Expressed in exponential notation with 17 digits or more.
- DEC constant is expressed with adding a suffix.

Example) Value: **LREAL**; Double precision floating point type: **#1234**

Range -1.79E+308≦N≦-2.23E-308

N=0

+2.23E－308≦N≦+1.79E+308 (Number of effective digits: approx. 16 digits)

Fixed character string (String Constant)

Overview Can be written by enclosing any character string in " ' ".

Notation The character string is written between " ' ".

Example) String: "Operation result"

Reference

- To write " ' " as a character, write as "$'".
- To process other characters with " ' ", combine the characters with "+".

strValue := '$'' + 'Operation result' +'$''; " 'Operation result' " is stored in strValue.

Range Up to 1999 character strings can be processed.

Index modify and Indirect designation

Devices that can be used with Index modify and Indirect designation are listed below.

Device			KV-8000 Series	
Type	Symbol	Name	Index modify	Indirect designation
Bit device	R	Relay	○	○*
	CR	Control relay	–	–
	T	Timer (Contact)	○	○*
	C	Counter (Contact)	○	○*
	MR	Internal auxiliary relay	○	○*
	LR	Latch relay	○	○*
	B	Link relay	○	○*
Word device	DM	Data memory	○	○
	EM	Expansion data memory	○	○
	FM	File register (Bank Switch)	○	○
	ZF	File register (Serial No. type)	○	○
	W	Link register	○	○
	TM	Temporary memory	○	○
	TS	Timer (Set value)	○	○
	TC	Timer (Current value)	○	○
	CS	Counter (Set value)	○	○
	CC	Counter (Current value)	○	○
	CM	Control memory	–	–

1 Indirect designation is possible, but the address cannot be stored with writing as "(Device)".

Point
- **Index modify is not possible for CR, Z, Variable (Variable Array), and Bit-positioned Word device.**
- **Indirect designation is not possible for CR, Z, and Bit-positioned Word device.**

Index modify

Index modify refers to a method of specifying a numbered Device by adding the value Index register (Z) and a constant.
The range of the value that can be added is signed 32-bit value (- 2147483648 to + 2147483647).
Available Devices vary by unit.
Index modify can be used together with Indirect designation (∗) or Local device (@).

Point
- **Index modify is not possible for Label (Label Array) and Bit-positioned Word device.**
- **Index modify cannot be performed for Variable. Index modify is not possible for Variables unless those manually assigned to Global devices.**
- **If a Device outside the Index modify usable range is specified as an Argument, the Control relay CR2012 turns ON and the Function is not executed. Operating will continue.**

■Modifying by Index register

It is used to change the Device No. while operating.

The modifying method with Index register is as follows:

(Device No.): Z(01 to 10)

Example)

DM1000:Z01 — Specify DM0990 when value in Z01 is -10

R0000:Z05 — Specify R0100 when value in Z05 is +16.

Point **Z11 or later cannot be used because it has been system reserved.**

Example

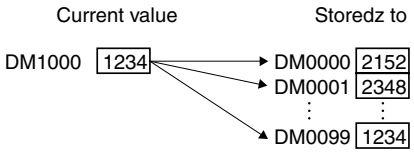
Example The Current value stored in DM1000 will be saved as the trigger signal for triggering R0001.

The Index register is initialized at the start of operation, and data is saved in order from DM0000 when R0001 starts up (OFF→ON).

ST Program

```
IF CR2008 THEN
    Z01=0
END IF
IF LDP(R0001) THEN
    DM0000:Z01 = DM1000
    INC(Z01)
END IF
```

- Set index register Z01 to 0 when starting operation.
- Write value of DM1000 to destination device to be index-modified at rising edge of R0001.
- Increment (+1) index register Z01



Point **Since the Device (value) to be index-modified is processed as 32-bit signed data, the Index register without type specifying is automatically executed as a ".L (32-bit signed)" suffix.**

Reference If 10 or more Index Registers (Z) are required in a project, batch clear or restore the Index Registers in each Module or Macro with the Index register Function (ZPUSH/ZPOP).

「ZPUSH function」(page 6-7)
「ZPOP function」(page 6-8)

■Modifying by Index constant

The modifying method with a constant is as follows:

(Device No.):(DEC constant)

Example)

DM1000:10 — DM1000+10 words -> DM1010

R0000:16 — R0000+16 bits (1 channel) -> R0100.

Reference If it is used when writing a Ladder program using Unit devices, it helps to confirm the Device No. from the assigned leading Device. It is also effective when using UR / UM as the Argument device when creating a Macro.

Example)

UR0:2

UM1:32

■Modifying by Device or Expression

ST allows the Index modify available by using other than Index register.
In this case, Index register (Z11/Z12) automatically system reserved will be used during Ladder conversion.

ST Program DM1000:(DM10+2) := 100;

* For Index modify with the Expression, be sure to enclose the Expression in ().

■Modifying the Device with a suffix

Write the suffix for a Device to be index modified as follows:

DM1000.D:Z01.S
DM1100.F:(DM10+10)

■Invalid Index modify

Index modify cannot be performed in the following cases:

- **(DM1000: Z01): Z02 · · · To modify an index modified Device**
- **(DM1000+3): Z02 · · · · To modify the Operation result**
- **DM1000: Z01+5 · · · · · ( ) is not used in the Index modify expression**
- **Z01:Z02 · · · · · · · · To modify the Device that cannot be index modified**

Indirect designation

■What is Indirect designation(∗)

Indirect designation (∗) refers to specify the Device indirectly (but not directly) by writing the Device in which the value (address) indicating the Device is stored.

●Indirect designation mechanism

Each Device is managed by internal memory in CPU unit, and has a separate address.

- **Indirect designation (∗) is used to change the Device that references the address by operating it with a dedicated Function.**
- **Index modify is expressed as "Reference device + Relative offset value (Index constant)", while Indirect designation is expressed as "*" + Reference destination address save device.**
- **Indirect designation can be used with Index modify (: Z /: DEC constant) and Local device (@).**
  Two words are required to store the address due to the address is managed in 32-bit.

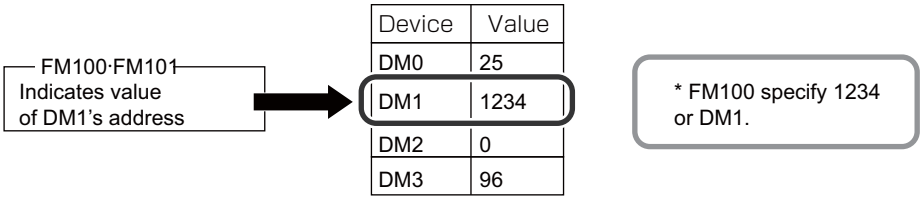
Point
- **If you indirectly specify a Device usable as an Argument, the operation of the Function is not guaranteed.**
- **If you specify a Device unavailable in the Operation expression, the operation result is not guaranteed.**

●Write with Indirect designation

Writing as "∗ (Device)" specifies the Device or Device value corresponding to the address stored in that Device.

Example)

＜When the address of DM1 is set to FM100 and written as ∗FM100＞



Available Devices vary by unit. For details, refer to “Devices that can be used with Index modify and Indirect designation.”

Point

- **Indirect designation is possible for Variable. Variables can also be specified indirectly unless the following cases:**
- **Subscript of Array type variable**
- **Element of Array type variable**
- **Member of Structure type variable**
- **Bit-positioning for Word data is impossible.**

Reference

- If the Device to be indirectly specified is a Timer (T) or Counter (C), Contact, Current value, and Set value are determined by the used Function Argument the added suffix.
- Write as "*@ (Device)" when using the Local device.
- Indirectly specified Devices are processed as .U (16-bit unsigned).
- Type specifying is also possible for Indirect designation.

●Using the Indirect designation

Perform the Indirect Specify using the following procedure.

1 Use the Address Set Function (ADRSET) to write the address of the specified Device.

Example)

```
ADRSET(R1000,DM0,FM100.D);
```

Store address of DM0 to FM100/FM101 when execution condition (R1000) is ON.

Point

Devices that can be used vary by unit. For details, refer to "Devices that can be used with Index modify and Indirect designation."

「ADRSET function」(page 6-9)

2 To indirectly specify each Argument (Device) of a Function, add "∗" to the leading of the Device that stores the address.

Example)

```
EM1000.U := AVG( ∗FM100.U,10)
```

∗Store average vaue of 10-word data starting from FM100 (DM0) to EM1000.U.

Point

Before adding "*" to the Address save device, be sure to store the address of the Device indirectly specified by the ADRSET Function.

3 To change the specified Device, Indirect Designation Function (ADRINC/ADRDEC/ADRADD/ ADRSUB) can be used to change the stored address.

Example)

```
ADRSET(EM300, FM100.D);
ADRINC(R1001, FM100.D);
*FM100 := DM1000;
```

Set EM300's address to FM100.

Increment (+1) DM0 address stored in FM100/ FM101 when execution condition (R1001) is ON

∗ Store svalue of DM1000 in FM100 (EM300).

Point

When changing the specified address, do not use common Operators (+, -,*, /,etc.). Otherwise, Ladder program fails to operate correctly.

● Indirect designation device type

As with other Devices, the data range that can be processed of the Indirect designation device (*[Device]) varies by the suffix.

Device type can be written by adding a suffix to Indirect designation device.

ST Program
```
*DM1000.F := 123.567;      //Indirect designation device type-specified by DM1000 is processed as .F
                           //(single precision/floating point type) type.
```

- If you omit the suffix of an Indirect designation device without type-specified, its type is processed as .U.

ST Program
```
*DM1000 := INT#65535;      //Indirect designation device type-unspecified by DM1000
                           //is processed as .U (unsigned 16-bit).
```

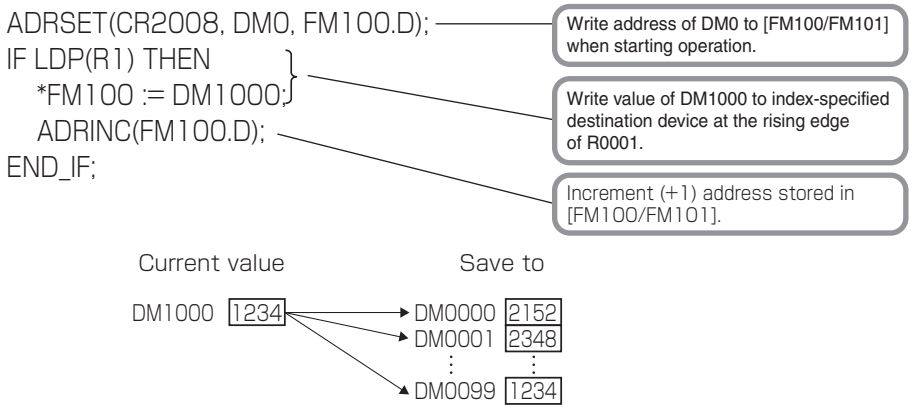
In this case, even if the type of Indirect designation device is not .U, but if the its suffix is omitted, type of the Indirect designation device is also processed as .U.

ST Program
```
ADRSET (MR2000, DM1000.D);   //MR2000 address is stored in DM1000.D.
*DM1000 := 123;              //MR2000 is a Bit device.
                             //If the suffix is omitted, //*DM1000(=MR2000) is processed as .U (Word Device).
```

● Usage example for Indirect designation

At the first scan after operation starting, the address indicating DM0 is stored in FM100 and FM101.

Triggered while R1 starting up, the Current value of DM1000 is stored in order from DM0.



Combination of Index modify and Indirect designation

■ About writing method

To use Index modify for Indirect designation device, write with the following method.

Example)
```
(*DM1000.U):Z01    ← * Index-modify DM1000 by Z01.
*(DM1000.U:Z01)    ← Indirect-specify DM1000 by Z01.
```

If ( ) is omitted, the meaning changes:

Example)
```
*DM1000.U:Z01      ← * Index-modify DM1000 by Z01.
```

● Example for combination of Index modify and Indirect designation

Triggered while R1 starting up, the Current value stored in DM1000 is saved.

Data is saved in order from DM0 if R2 is ON while R0 starting up, and from EM0 if R2 is OFF while R0 starting up.

```
IF LDP(R0000) THEN
   Z01 := 0; ..................................... ①
   ADRSET(R2, DM0, FM100.D); ................... ②
   ADRSET(R2 = FALSE, EM0, FM100.D);........ ③
END_IF;
IF LDP(R1) THEN
   *FM100:Z01 := DM1000; ...................... ④
   Z01 := Z01 +1; ............................... ⑤
END_IF;
```

[When starting measurement]

①Write 0 to Index register Z01. (Index register initialization)

②If R2 is ON, write the DM0 address to FM100/FM.

③If R2 is OFF, write the EM0 address to FM100/FM101.

[When trigger occurs]

④Write the value of DM1000 to the Device with an address stored in FM100/FM101 index-modified with Z01 (adding the value of Z01).

Example) when R2 is OFF,

∗FM100→EM0

Z01 := 23;

→ *FM100:Z01→EM23

⑤Z01 increment (+ 1).

■Simple Indirect designation

ST cannot use simple Indirect designation (#TM). To use simple Indirect designation, write it with Ladder program.

Data Types (Suffixes)

In ST, Device type and Constant type can be represented by adding a suffix to the Device.
Devices with the suffix ".U .S .D .L .F .DF .B .T" process different types of data depending on the suffix.

Suffix

A □ suffix refers to a sign attached to the end of the Device, and can be divided into eight types as follows.

□.U □.S □.D □.L □.F □.DF □.B □.T

*The Device with the corresponding suffix is written in.

According to the suffix, each Device can process the value of Data memory, etc. as follows:

Suffix □.U is processed as unsigned 16-bit data. (0 to 65535)
Suffix □.L is processed as signed 16-bit data. (- 32768 to 32767)
Suffix □.D is processed as unsigned 32-bit data. (0 to 4294967295)
Suffix □.L is processed as signed 32-bit data. (- 2147483648 to 2147483647)
Suffix □.F is processed as the single precision floating point type real number data.

－3.4E+38 ≦ N ≦ －1.4E－45
N = 0
1.4E－45 ≦ N ≦ 3.4E+38
（Number of effective digits: approx. 7 digits)

Suffix □.DF is processed as the double precision floating point type real number data.

－1.79E+308 ≦ N ≦ －2.23E－308
N = 0
2.23E－308 ≦ N ≦ 1.79E+308
(Number of effective digits: approx. 16 digits)

Suffix □.B is processed as Bit data. (ON:TRUE, OFF:FALSE)
Suffix □.T is processed as the character string data.

Reference
The suffix specified by Ladder program can be only added at the end of an instruction, but not a Device (In ST, the suffix is only added at the end of the Device, but not the Function).

Example

```
DM1000 (.U)
DM1000.S
DM1000.D
DM1000.L
DM1000.F
DM1000.DF
DM1000.B
DM1000.T
```

Point
In ST language, Variable cannot be used with adding a suffix.

● Omitting the suffix

When writing a Device without adding a suffix, the type depends on the Device type.

Device type	Suffix-omitted type (default type)
R(DR),B,MR,LR,T,C,CR,UR	.B (Bit type)
DM,W,EM,ZF,FM,TM,CM, Indirectly designation device	.U (unsigned 16-bit data type)
Z*	.L (signed 32-bit data type)

* Default type varies with different CPU units.

Point **When the suffix is omitted for a Variable, it is processed according to the data type set in the Variable Edit window.**

ST program

```
DM1000 := 2000;                    //'When omitted, processed as unsigned 16-bit (.U)
DM1100.D := 75535;                 //DM1100 is processed as unsigned 32-bit (.D)
IF DM1000 = 3000 THEN              //
   DM3000.L := DM1100 + 1000;      //Right side is calculated with signed 32-bit.
END_IF;                            //
                                   //
DM1000.L := -21474800;             //DM1000 is processed as signed 32-bit (.L)
IF DM1000 > 0 THEN                 //
   DM3000.D := DM1000.S * -1;      //Right side is expanded to .L to calculate, and the solution is converted to .D
END_IF;                            //to enter.
```

■About .U and .S

Suffix ☐ .U processes BIN data as unsigned 16-bit data.
Suffix ☐ .S processes BIN data as signed 16-bit data.
Therefore, the relationship between the BIN data and the DEC notation depending on the suffix is as follows

Device (DM1000)	Constant type	For the Bit data 1111111111111111
DM1000.U	Unsigned DEC	65535
DM1000.S	Signed DEC	-1

* When comparing 16-bit BIN data (1111111111111111) stored in DM1000 with the DEC constant 100, if the suffix is .U, the value of DM1000 is calculated as 65535.

ST program

```
IF DM1000.U > 100 THEN
   INC(DM2000);
END_IF;
```

Therefore, DM1000 > 100 is TRUE, "1" is added to DM2000.

- If the suffix is .S, DM1000 value is calculated to "-1".

ST program

```
IF DM1000.S > 100 THEN
   DEC(DM2000);
END_IF;
```

Therefore, DM2000 is not subtracted due to DM1000 < 100.
For details on Control statements, refer to "Control Statements"(page 4-6).

■About .D and .L

Suffix ☐ .D processes BIN data as unsigned 32-bit data.
Suffix ☐ .L processes BIN data as signed 32-bit data.

Therefore, the relationship between the BIN data and the DEC notation depending on the suffix is as follows

Device (DM1000)	Constant type	For Bit data 11111111111111111111111111111111
DM1000.D	Unsigned DEC	4294967295
DM1000.L	Signed DEC	-1

- When comparing 32-bit BIN data (11111111111111111111111111111111) stored in DM1000/DM1001 with the DEC constant 100, if the suffix is .D, the value of [DM1000/DM1001] is calculated as 4294967295.

ST program
```
IF DM1000.D > 100 THEN
   DM20000 := DM2000 + 1;
END_IF;
```

Therefore, DM1001 (upper bit)/DM1000 (lower bit) (4294967295) > 100, the Contact turns ON and "1" is added to DM2000.

•If the suffix is .L, the value of [DM1000/DM1001] is processed as "-1".

ST program
```
IF DM1000.L > 100 THEN
   DM2000 := DM2000 -1;
END_IF;
```

Therefore, DM1001 (upper bit)/DM1000 (lower bit) (-1) < 100, the Contact turns OFF, DM2000 is not subtracted.

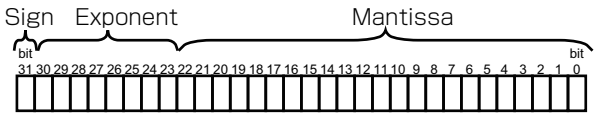
Point **When processing 32-bit data (.D,.L,.F), use even number for the Device No.**

■About .F

□.F suffix processes BIN data as the single precision floating point type real number. The 32-bit data.
Single precision floating point type real number data is expressed (Internal expression) with a special Bit configuration.
Therefore, avoid parsing Internal expressions and operating Bits directly as much as possible.
Single precision floating point type real number value is expressed according to the following method.

$$-1.234\times2^{-3}$$

Sign Mantissa Exponent



Sign (1 bit)
0＝plus
1＝minus

Mantissa section（23 bit）
0～8388607

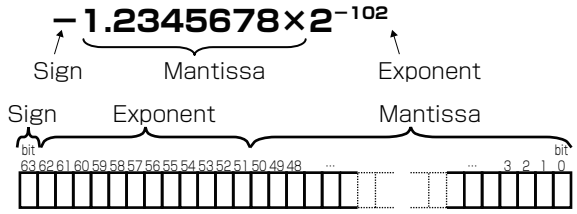
Exponent (8-bit)
00000000（16#00）＝not used
00000001（16#01）＝-126
00000010（16#02）＝-125
00000011（16#03）＝-124
:
01111110（16#7E）＝-1
01111111（16#7F）＝0
10000000（16#80）＝1
10000001（16#81）＝2
11111100（16#FC）＝125
11111101（16#FD）＝126
11111110（16#FE）＝127
11111111（16#FF）＝not used

Point

When processing 32-bit data with Word devices, add the suffix .D, .L and .F to the Device such as DM/W/EM/FM/TM/ZF, etc.
In this case, the Device uses 2 words with continuous No.
The specified Device is added a suffix to the lower 16-bit.
Example) For DM1000.D, use "DM1001, DM1000".

■About .DF

□ .DF suffix processes BIN data as the double precision floating point type real number. The 64-bit data.
Double precision floating point type real number data is expressed (Internal expression) with a special Bit configuration.
Therefore, avoid parsing Internal expressions and operating Bits directly as much as possible.
Double precision floating point type real number value is expressed according to the following method.



Sign (1 bit)
0＝plus
1＝minus

Mantissa section（51 bit）
0 to 2251799813685248

Exponent (12-bit)
000000000000（16#000）＝not used
000000000001（16#001）＝－2046
000000000010（16#002）＝－2045
000000000011（16#003）＝－2044
:
011111111110（16#000）＝－1
011111111111（16#7FF）＝0
100000000000（16#800）＝1
100000000001（16#801）＝2
:
000000000000（16#000）＝2045
000000000000（16#000）＝2046
000000000000（16#000）＝2047
111111111111（16#FFF）＝not used

Point

When processing Double precision floating point type real number with Word devices, add the suffix .DF to the Device such as DM/W/EM/FM/TM/ZF, etc.
In this case, the Device uses with continuous 4 words. Use even-numbered Devices when processing the suffix .DF.
The specified Device is added a suffix to the lower 16-bit.
Example) For DM1000.DF, use "DM1003･DM1002･DM1001･DM1000".

■About .B

.B suffix processes BIN data as the Bit data.

Example) Set the Bit device MR1000 to ON (1, TRUE).

ST program
```
SET(MR1000.B);
```

Reference

When the suffix .B is added to a Word device, the lowest bit will be the Bit type judgment object. In the above example, if DM10.B is specified as the Argument of the SET Function, the lowest bit of DM10 will be 1. The TRUE/FALSE judgment of the Word device is as follows.

```
Example)  IF DM10.B = TRUE THEN
              Process statement;
          END_IF;
```

Example) Set the Bit device MR1000 to OFF (0, FALSE).

ST program
```
RES(MR1000.B);
```

■About .T

Suffix □.T processes BIN data as the character string data.

Example) To store the string 'KEYENCE' on consecutive Devices with the DM100 leading.

ST program DM100.T := 'KEYENCE' + CHR(16#0D) + CHR(16#0A);

In the case of above example, the data storage state is as follows.

Reference NUL(00H) will be automatically stored at the end of the character string.

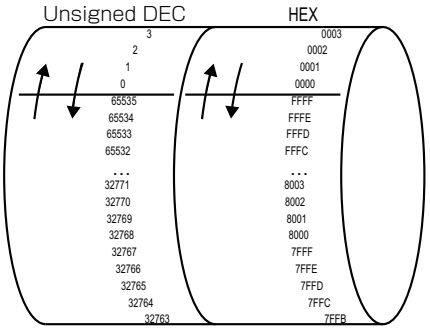
Point **Up to 1999 character strings can be processed.**

■Value changes when exceeding the Max./Min. value

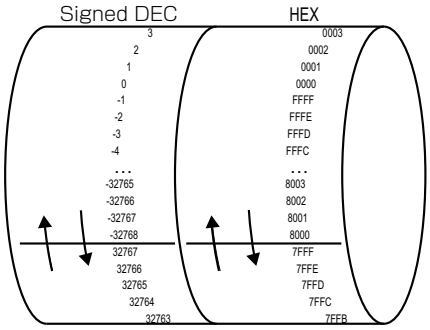
When the arithmetic operation result exceeds maximum/minimum value, values will change as follows depending on the data type.

● For 16-bit data

Value change when □.U larger than 65535

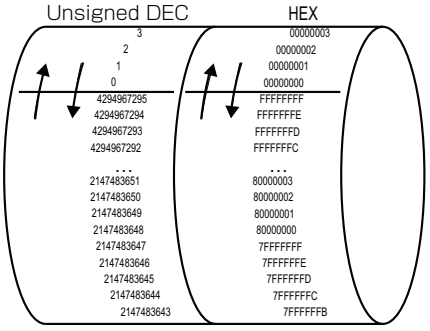


Value change when □.S larger/less than max./min. value

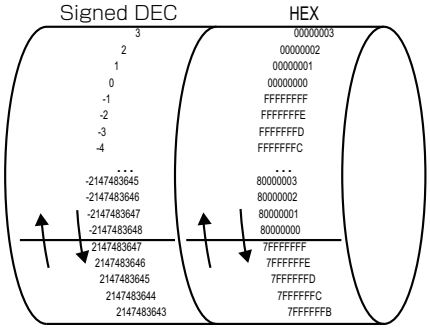


● For 32-bit data

Value change when □.D larger than 4294967295



Value change when □.L larger/less than max./min. value



Reference ·For the single-precision floating-point data

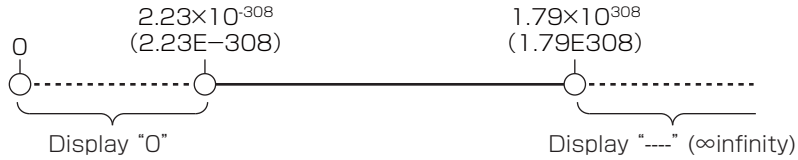
Overflow: when the absolute value is larger than 3.4×10^{38}, changes to "∞ (infinity)".

Underflow: when the absolute value is less than $1.4\times10^{-45*}$, changes to "0".

・For the double-precision floating-point data

Overflow: when the absolute value is larger than 1.79×10^{308}, changes to "∞ (infinity)".

Underflow: when the absolute value is less than 2.23×10^{-308}, changes to "0".



About Operation type

Unlike Ladder language, ST uses Operators (+, -, *, /) instead of using special instructions for writing program when performing numerical operations.

■Data range processable by Operation expression

All terms in an Operation expression are processed as .L (signed 32-bit), .F (single precision floating point type) or .DF (double precision floating point type). Therefore, all Devices in an expression are automatically sign-extended to .L, .F, or .DF depending on the operation expression to be executed in sequence. In addition, the operation result is converted and stored according to the data type of the storage destination. Please note that accuracy may drop depending on the order of operation and the result value.

4

ST Statements and Precautions

Assignment statements 4-2
Operators 4-4
Control Statements 4-6
Precautions on ST Programming 4-18

Assignment statements

Assignment statement refers to a statement consisting only of Expressions in which the left and right sides are connected by an equal sign (=), and the result or value of the right side is assigned (stored) to the left side.

How to use Assignment statements

Value/String/Bit, etc. can be written as the right side of the Assignment statement (Device/Variable/Expression). You can write them with a Constant/Operation expression/Function (with a Return value).

● Value assignment statement

Assign right Value constant or Value operation result to the left side.

ST program

```
uAnalogData := 123;                          //Assignment of Value Constant
lSettingParam := uTempU + 123;               //Assignation of Value operation result
uCount := DCNT(uSettingPrm,auMeasure[0],20); //Assignment of Function process result
```

● String assignment statement

Assign right String

constant or String operation result to the left.

ST program

```
stProductName := 'ABCD';                     //Assignment of String Constant
DM200.T := 'Current' + DM300.T + 'use';      //Assignation of String Concatenation Result
```

● Bit assignment statement

- Assign right Bit constant or Bit operation result (Condition expression evaluation) to the left.

ST program

```
R1000 := TRUE;                               //Assignment of Bit constant (Boolean value)
R1000 := R1100 OR (R1200 AND R1300);         //Assignment of Bit operation result
R1000 := DM100.S > 100;                      //Assignment of Condition expression evaluation
R1000 := R2000 = R3000;                      //Assignment of Comparison operation result
```

- Bit state can be represented by 1, 0 or TRUE, FALSE.

ST program

```
R1000 := INT#1;
```

Description

Set R1000 to ON.

- Only Boolean values can be assigned to Bit data.

```
R1000 := TRUE;      Turns R1000 on
R1000 := FALSE;     Turns R1000 off
```

- Constants other than 1 and 0 and the Value operation expression result can not be assigned to Bit data.

×: R1000 := DM1000 + DM2000;“ . . . . . +” is unavailable

- It is possible to assign the Logic value operation expression result or the Comparison operation result (both results are Boolean values) to Bit data.

○: R1000 := R2000 AND DM2100.B;Logical operators can be used
R1000 := R2000 = R2100; . . . . . Equal/inequality sign (expression whose operation result only includes Boolean value) are available

■Precautions on using Assignment Statement

●When the type on right side is different from that on left side

- Assignment statement is FALSE when the types on right side and left side are different (type conversion cannot be performed).

ST program stCurData := uTemp; //Error

Description Attempting to assign right uTemp (Constant: unsigned 16-bit) to stCurData (String type) (Conversion error)

- However, even if the types are different, the difference in type between .U.S.D.L.F can be converted (precision loss may occur).

ST program DM1100.U := DM2100.D;

Description Assign right DM2100.D (unsigned 32-bit) to DM1100.U (unsigned 16-bit).
In such a case, only the lower 16-bit is assigned to DM2100.D.

●When it cannot be assigned to left side

- In the ST below, "3" on the left side is a numerical value, so it is incorrect as an Assignment statement.

ST program 3 := DM1000.U; //Error (can not be assigned to Constant)

Description Attempting to assign right DM1000.U (unsigned 16-bit) to 3 (Constant). (Conversion error)

- Even if when writing the left side with Expression or Function, it is incorrect as an Assignment statement and error occurs.

ST program DM1100.U + 30 := DM1000.U; //Error (can not be assigned to Expression)

Description Attempting to assign right DM1000.U (unsigned 16-bit) to left Expression. (Conversion error)

Operators

Operator refer to data processing signs, such as the plus sign (+) and multiplication sign (∗), etc. used in Expressions.

Operator list

■Operator type

"Arithmetic operator","Comparison operator","String concatenation operator" and "Logic operator" etc.

●Arithmetic operator

Returns the arithmetic operation result of 2 values.

Sign	Processing content	Program example
**	Calculates the power	uiValue1 := uiValue2 ** 2;
∗	Calculates the product of 2 values (multiplication)	uiValue1 := uiValue2 * 3;
/	Calculates the quotient of 2 values (division)	uiValue1 := uiValue2 / 4;
MOD*1	Divides 2 values and returns the remainder	uiValue1 := uiValue2 MOD 5;
+	Calculates the sum of 2 values (addition)	uiValue1 := uiValue2 + 6;
−	Calculates the difference of 2 values (subtraction)	uiValue1 := uiValue2 -7;

*1 A 1-byte space is required before and after the sign.

●String concatenation operator

Returns the concatenation result of 2 character strings.

Sign	Processing content	Program example
+*2	Concatenates 2 character strings	strName := 'KEY' + 'ENCE';

*2 +” is automatically is automatically determined for arithmetic operations and string concatenation.

●Comparison operator

Returns the evaluation result of size relation between 2 values as a Boolean value (TRUE or FALSE).

Sign	Processing content	Program example
<	Less than	bFlag := uiValue < 10;
<=	Less than or equal to (Lower)	bFlag := uiValue <= 20;
>	Larger than	bFlag := uiValue > 30;
>=	Larger than or equal to (Upper)	bFlag := uiValue >= 40;
=	Equal to (equivalent)	bFlag := uiValue = 50;
<>	Unequal to (unequal)	bFlag := uiValue <> 60;

●Logic operator

Returns the Logical operation result of 2 values as a Boolean value (TRUE or FALSE) for bit-to-bit operations, and as a word value for word-to-word operations.

Sign	Processing content	Program example
NOT*3	Calculates Logical negative value (NOT) of the value	bFlag1 := NOT bFlag2;
AND*3	Calculates Logical product (AND) of 2 Devices (for Bit data only, or Word data only)	bFlag1 := bFlag2 AND bFlag3;
OR*3	Calculates Logical sum (OR) of 2 Devices (for Bit data only, or Word data only)	bFlag1 := bFlag2 OR bFlag3;

Sign	Processing content	Program example
XOR*3	Calculates Exclusive logical sum (XOR) of 2 Devices (for Bit data only, or Word data only)	bFlag1 := bFlag2 XOR bFlag3;

*3 A 1-byte space is required before and after the sign.

Point

- **Specify the operation data (operation element) of only Bit data or only Word data. Logical operations cannot be performed between Bit data and Word data, etc.**
- **If the size of the type of data (operation) to be calculated is different, the operation is performed according to the larger size type.**
  Example) DM1000.U and DM2000.D.... Operates with 32-bit type

Operator priority

When multiple Operators are used in one Expression, each operation section of the Expression is evaluated (calculated) in a fixed order. This order is called Operator priority.

■About priority

If different types of Operators are used in one Expression, they are evaluated in order of priority in the table below.

Minus sign, Logic negative	-, NOT
Power	**
Multiplication, Division, Remainder	*, /, MOD
Addition, Subtraction	+, -
Less than, Larger than, Lower, Upper	<, >, <=, >=
Equal to, Unequal to	=, <>
Logic product	AND, &
Exclusive logical sum	XOR
Logic sum	OR

Control Statements

Control statement refers to the statement that changes the programming procedure by branching the processing flow, repeating the operation steps, and changing the content and frequency of execution, according to the conditions.

Control Statement List

There are three types of control statements: "Condition branch statement", "Loop control statement", and "Program execution end statement" shown below.

■Condition branch statement

Condition branch statement that branches the processing flow under the specified conditions.

Control statement	Control content	
IF statement	①Branch control statement that executes according to the Condition expression	
	IF ␣Condition expression A ␣ THEN Process statement B; END_IF;	If Condition expression A is TRUE Executes the Process statement B. If Condition expression A is FALSE, nothing will be processed.
	②Dual branch control statement that executes separate processes according to Condition expression	

	IF ␣ Condition expression A ␣ THEN
	Process statement B;
	ELSE
	Process statement C;
	END_IF;
		If Condition expression A is TRUE
		executes the Process statement B.
		If Condition expression A is FALSE,
		executes the Process statement C.

	③Multiple branch control statement that executes separate processes according to the condition value	
	IF ␣Condition expression A ␣THEN Process statement B; ELSIF ␣Condition expression C ␣THEN Process statement D; . . . END_IF;	If Condition expression A is TRUE Executes the Process statement B. If Condition expression A is FALSE and Condition expression C is TRUE, Process statement D is executed. If neither of the Condition expression A, C・・・ is FALSE, nothing will be processed.
CASE statement	Multiple branch control statement that selects the corresponding process and executes according to the condition value	
	CASE ␣comparison source ␣OF Condition value A: Process statement B; Condition value C: Process statement D; : ELSE Process statement E; END_CASE;	 If the comparison source and the condition value A are equal, Process statement B will be performed. If the comparison source and the condition value C are equal, Process statement D will be performed. If the comparison source is other than the condition values A, C, etc., Process statement E will be performed.

■Loop control statement

Loop control statement that repeats the process under the specified conditions

Control statement	Control content	
FOR statement	Loop control statement that repeats the process until the specified number of Condition expression is satisfied (TRUE)	
	FOR␣initial value␣TO␣final value␣BY␣incremental value␣DO Process statement A; END_FOR;	Process statement is repeatedly executed while adding incremental values from the initial value to the final value.
WHILE statement	The indefinite number of Loop control statement that repeats processing while the Condition expression is satisfied (TRUE)	
	WHILE␣Condition expression␣DO Process statement; END_WHILE;	Process statement is repeatedly executed while Condition expression is TRUE
REPEAT statement	The infinite number of Loop control statement that repeatedly executes processing until the Condition expression is satisfied (TRUE)	
	REPEAT Process statement; UNTIL␣Condition expression END_REPEAT;	Process statement is repeatedly executed while Condition expression is TRUE
EXIT statement	Loop control statement that exits the loop regardless of the Loop condition	
	<Loop process statement> ・ ・ EXIT; ・ ・ <Loop process statement>	When EXIT is executed, the Loop control statement including EXIT ends regardless of the condition.

■Program execution end statement

Ends the execution of the written program.

Control statement	Control content	
RETURN statement	Control statement that terminates the execution of the written program	
	Process statement; RETURN; Process statement;	Ends the execution of the written program.

IF ~ statement (Condition branch ①)

The most simple Condition branch statement that determines whether or not to process with one Condition expression.

IF ～ THEN ～ END_IF	
Basic expression	**Description**
IF␣<Condition expression>␣THEN <Process statement 1>; END_IF;	If Condition expression is TRUE, Process statement 1 will be executed. IF statement ends here

* ␣ Is indicates inserting a 1-byte space.
↵

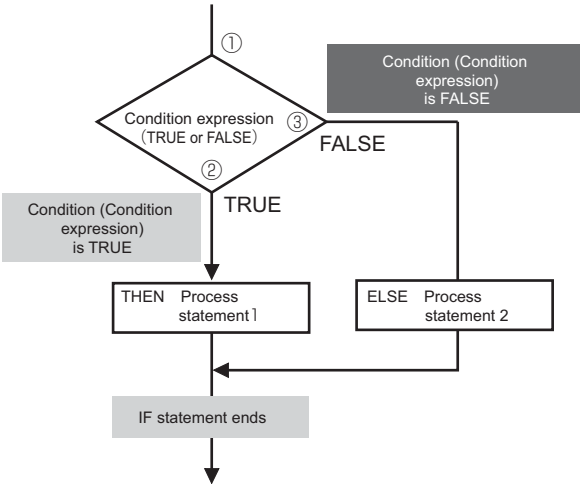
Expression example	Description

IF ␣ bFlag = TRUE ␣ THEN
uiValue := uiValue + 1;
END_IF;
	If bFlag is ON,
	the constant 1 is stored in uiValue.

One-Point

- If the result of Condition expression is Bit (return Boolean value) expression, it can be used in any Expression.
- Process statement refers to a processing flow used for writing between THEN to END_IF, which can be written across multiple lines as needed.
- IF statements can be nested. By further writing IF statement in the Process statement, multiple branch processing can be performed while narrowing the scope according to the AND condition.

Processing flow



① Judges "TRUE" or "FALSE" in Condition expression.

② If Conditional expression is "TRUE", executes Process statement after THEN and executes END IF.

③ If Condition expression is "FALSE", jumps to END_IF and moves to the processing after END_IF without executing.

IF ~ ELSE statement (Condition branch ②)

Simple Condition branch statement that can divide the processing method into two types with one Condition expression.

IF ～ THEN ～ ELSE ～ END_IF	
Basic expression	**Description**

IF ␣ ＜Condition expression＞ ␣ THEN
＜Process statement 1＞;
ELSE
＜Process statement2＞;
END_IF;
	If Condition expression is TRUE,
	Process statement 1 will be executed.
	If Condition expression is FALSE,
	Process statement 2 will be executed.
	IF statement ends here

* ↵ is indicates inserting a 1-byte space.

Expression example	Description

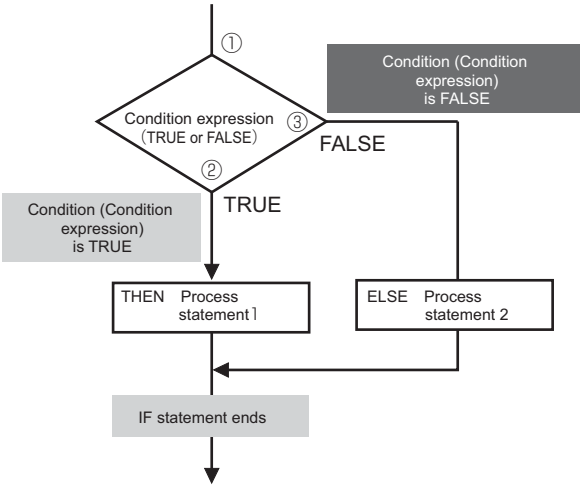
IF ␣ bFlag = TRUE ␣ THEN
uiValue := 50;
ELSE
uiValue := 10;
END_IF;
	If bFlag is ON,
	the constant 50 is stored in uValue.
	If bFlag is OFF,
	the constant 10 is stored in uValue.

One-Point

- If the result of Condition expression is Bit (return Boolean value) expression, it can be used in any Expression.

- Process statement refers to a processing flow used for writing between THEN to END_IF, which can be written across multiple lines as needed.
- IF statements can be nested.

Processing flow



① Judges "TRUE" or "FALSE" in Condition expression.

② If Conditional expression is "TRUE", executes Process statement 1 after THEN, and jumps to END_IF (expressions between them are not executed).

③ If Condition expression is "FALSE", executes Process statement 2 after ELSE without executing anyone between THEN and ELSE.

IF ~ ELS IF ~ statement (Condition branch ③)

Condition branch statement that can correspond multiple conditions to determine again under the next Condition expression when the condition of the first Condition expression is not satisfied.

IF ～ THEN ～ ELSIF ～ END_IF	
Basic expression	**Description**
IF ␣＜Condition expression 1＞␣THEN ＜Process statement 1＞; ELSIF ␣＜Condition expression 2＞␣THEN ＜Process statement2＞; ELSIF ␣＜Condition expression 2＞␣THEN … ELSE ＜Process statement n＞; END_IF;	If Condition expression 1 is TRUE Process statement 1 will be executed. If Condition expression both 1 and 2 are FALSE, Process statement 2 will be executed. If Condition expression both 1 and 2 are FALSE, and Condition expression 3 is TRUE, *2 … If all Condition expression are FALSE, Process statement n will be executed. IF statement ends here.

* ␣ is indicates inserting a 1-byte space.

*1 Omissible.

*2 You can set any number of Condition expressions and Processing statements. There is no upper limit.

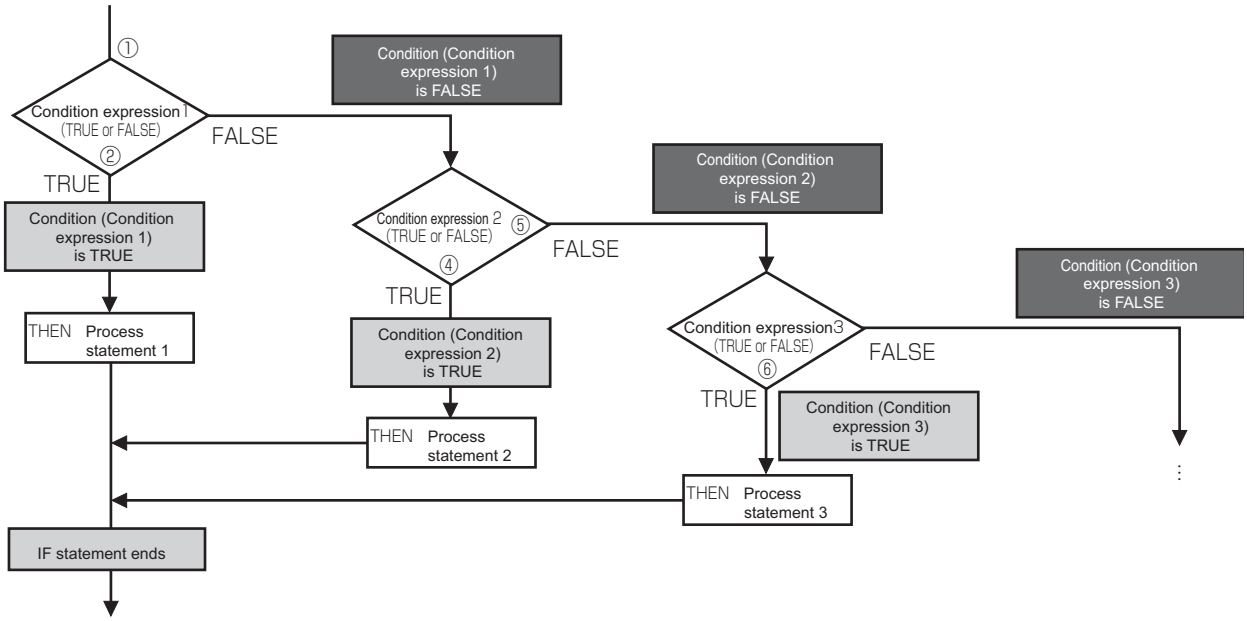
Expression example	Description

IF ␣ bFlag ␣ THEN
uiValue := 1000;
ELSIF ␣ bFlag1 ␣ THEN
uiValue := 1001;
ELSIF ␣ bFlag2 ␣ THEN
uiValue := 1002;
ELSIF ␣ bFlag3 ␣ THEN
uiValue := 1003;
ELSIF ␣ bFlag4 ␣ THEN
uiValue := 1004;
ELSE
uiValue := 1005;
END_IF;
	When bFlag is ON (abbreviation for bFlag = ON),
	1000 is saved in uValue.
	When bFlag is not ON, and bFlag 1 is ON,
	Adds 1001 to uValue.
	When bFlag and bFlag1 are not ON, and bFlag 2 is ON,
	1002 is saved in uValue.
	When bFlag, bFlag1, and bFlag2 are not ON, and bFlag 3 is ON,
	1003 is saved in uValue.
	When bFlag, bFlag1, bFlag2, and bFlag3 are not ON, and bFlag 4 is ON,
	1004 is saved in uValue.
	When all members of bFlag are not ON,
	1005 is saved in uValue.

One-Point

- If the result of Condition expression is Bit (return Boolean value) expression, it can be used in any Expression.
- Process statement is a series of programming processing between THEN and END_IF, capable of programming among lines if required.
- Unlike the IF statement, the ELS IF statement does not have a nested structure even if it is written in multiple lines.

Processing flow



① Judges "TRUE" or "FALSE" with Condition expression 1.
② If Condition expression is "TRUE", executes Process statement 1 after THEN and jumps to END_IF (the expression between them are not executed).
③ If Condition expression is "FALSE", Condition expression 2 determines "TRUE" or "FALSE".
④ If Condition expression 2 is "TRUE", executes Process statement 2 and jumps to END_IF (the expression between them are not executed).
⑤ If Condition expression 2 is "FALSE", Condition expression 3 determines whether it is "TRUE" or "fFALSE".
⑥ If Condition expression 3 is "TRUE", executes Process statement 3 and jumps to END_IF (the expression between them are not executed).
⑦ If Condition expression 3 is "FALSE", Condition expression 4 determines whether it is "TRUE" or "FALSE" ...go on

CASE ~ statement (Multi-branch control)

Condition branch statement that allows you to select to process based on the value stored in the reference Word data.

CASE	
Basic expression	**Description**
CASE ␣ <word data> ␣ OF <Constant 1>: Process statement 1; <Constant 2>: Process statement 2; <Constant 3>, <Constant 4>: Process statement 3; <Constant 5> .. <Constant 6>: Process statement 4; ELSE <Processing statement n>; END_CASE;	Based on the Word data value, executes Process statement 1 when the Condition Branch Constant 1 and Word data value are equal. When Constant 1 and Word data value are equal, Process statement 1 will be executed. When Constant 2 and Word data value are equal, Process statement 2 will be executed. When Constant 3 and Word data value are equal, … When no Constant and Word device data are equal, Process statement n will be executed. SELECT statement ends here.

* ␣ is indicates inserting a 1-byte space.

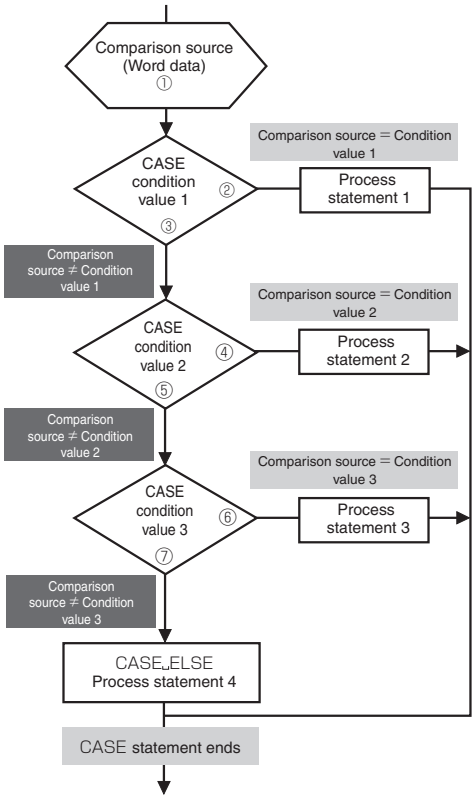
*1 Omissible.

*2 Up to 200 “CASE Constants (ELSE)” can be written.

*3 String/Device (which processing character string)/Bit device/BOOL variable cannot be specified as Constant 1.

Expression example	Description
CASE ␣ uiCount ␣ OF	Based on the value stored in uiCount.
1 : uiValue := 1000;	When uiCount = 1, 1000 is saved in uiValue.
2..10 : uiValue := 2000;	When uiCount = 2 to 10, 2000 is saved in uiValue.
11,15,60 : uiValue := 3000;	When uiCount = 11 or 15 or 60, 3000 is saved in uiValue.
101..65535 : uiValue := 4000;	When uiCount exceeds 100, 4000 is saved in uiValue.
ELSE	
uiValue := 5000;	When uiCount value is an value other than above, 5000 is saved in uiValue.
END_CASE;	

Processing flow



① Declare as the identification reference value of the condition value.

② Compares the condition value 1 with the comparison source, and when "Condition value 1 = Comparison source" is TRUE, executes Process statement 1. After executing Process statement 1, jumps to END_CASE.

③ When "Condition value 1 = Comparison source" is FASLE, moves to the next condition value comparison.

④ Compares the condition value 2 with the comparison source, and when "Condition value 2 = Comparison source" is TRUE, executes Process statement 2. After executing Process statement 2, jumps to END_CASE.

⑤ When "Condition value 2 = Comparison source" is FASLE, moves to the next condition value comparison.

⑥ Compares the condition value 3 with the comparison source, and when "Condition value 3 = Comparison source" is TRUE, executes Process statement 3. After executing Process statement 3, jumps to END_CASE.

⑦ When "Condition value 3 = Comparison source" is FASLE, executes Process statement 4.

Processing flow

One-Point

- The selected constant may be single specifying constant, or multiple specifying constant and range specifying constant.

  Example) 1 : Single specifying (1 only)
  3, 9, 12, 50 : Multiple specifying（ 3 or 9 or 12 or 50 ）
  8..20 :Range specifying (8 or more and 20 or less)

- ELSE "Process statement" writing can be omitted.
  However, the ELSE of " All conditions are FALSE " can help to solve the problem when an unexpected event occurs.

FOR ~ END_FOR statement (Frequency Specified Loop Control)

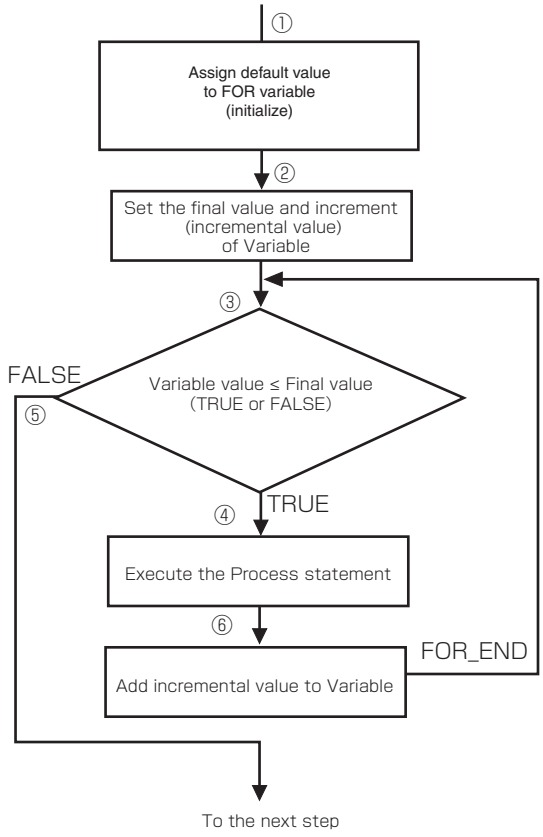
Control statement that determines the statement executing frequency based on the Loop condition.

FOR～END_FOR	
Basic expression	
1	FOR ␣ <Variable> := <Initial value> ␣ TO ␣ <Final value> ␣ BY ␣ <Incremental value> ␣ DO
2	<Process statement>;
3	END_FOR;
Description	
1	Determines Initial value and Final value of Variable that limit the frequency of Loop control. BY decides how many Variable values to increase for each process.
2	Executes the statement each time until the Variable exceeds the final value.
3	Adds an incremental value to the Variable.

*1 Writing in [ ] can be omitted. If omitted, will be executed as BY1.
␣ indicates inserting a 1-byte space.

Expression example	Description
FOR ␣ uiIdx := 0 ␣ TO ␣ 100 ␣ BY ␣ 5 ␣ DO	Cycles until the value of uiIdx goes from 0 to 100.
uiValue := uiValue + 5;	Stores the result of adding 5 to uiValue in uiValue.
END_FOR;	Adds incremental value 5 to current value of uiIdx, returns to FOR ~

Processing flow



① Assigns the initial value to Variable.

② Set the final value indicating the end of Loop process. Also, set the increment value added on the Variable value.

③ Judges whether the Variable value is less than the final value.

④ If the condition in ③ is "TRUE", executes Process statement.

⑤ If is "FALSE", ends the Process statement.

⑥ Adds the incremental value to the Variable value, and return to ③.

WHILE statement (Pre-judge Loop Control)

Control statement that executes a Loop control statement while the Loop condition is "TRUE".

WHILE ~ END_WHILE;	
Basic expression	**Description**
WHILE ␣ <Condition expression> ␣ DO <Process statement>; END_WHILE;	When Condition expression is FALSE,*1 executes the Process statement. Returns to the leading of the sentence (WHILE). ;

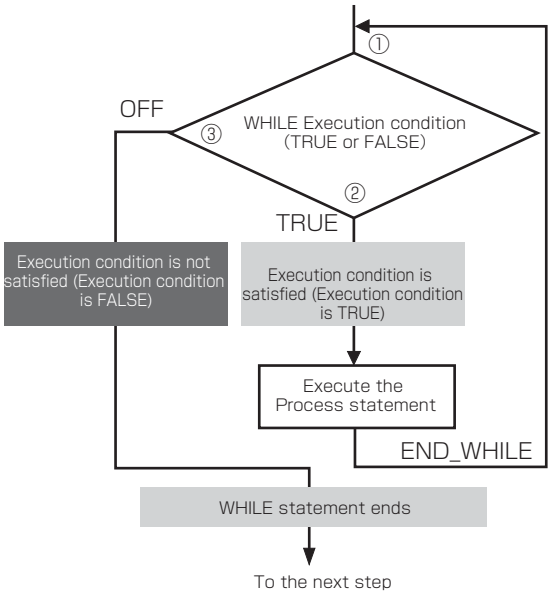
* ␣ is indicates inserting a 1-byte space.
*1 1When the Condition expression is FALSE, Loop Control is ended.

Expression example	Description

WHILE ␣ uiValue < 100 ␣ DO
uiValue := uiValue + 5;
END_WHILE;
	When the value of uiValue is less than 100,
	Adds 5 to uiValue.
	Returns to the leading of the sentence (WHILE).

*2 2If the value of uiValue is 100 or more, executed from the next line of END_WHILE without executing Process statement.

Processing flow



① Judges "TRUE" or "FALSE" by the WHILE execution condition, and decides the processing method between WHILE and END_WHILE.

② If the execution condition is "TRUE", executes the Process statement and returns to ① (the leading of the WHILE statement).

③ If the Condition expression is "FALSE", proceeds to the next step of the END_WHILE statement.

REPEAT statement (Post-judge Loop Control)

Control statement that executes a Loop statement while the UNTIL condition is “FALSE”.

REPEAT~ END_REPEAT	
Basic expression	**Description**
REPEAT <Process statement > UNTIL ␣ <Condition expression> END_REPEAT;	Executes Loop Control. executes the Process statement. When Condition expression is FALSE, returns to the leading of the sentence (REPEAT).* 1

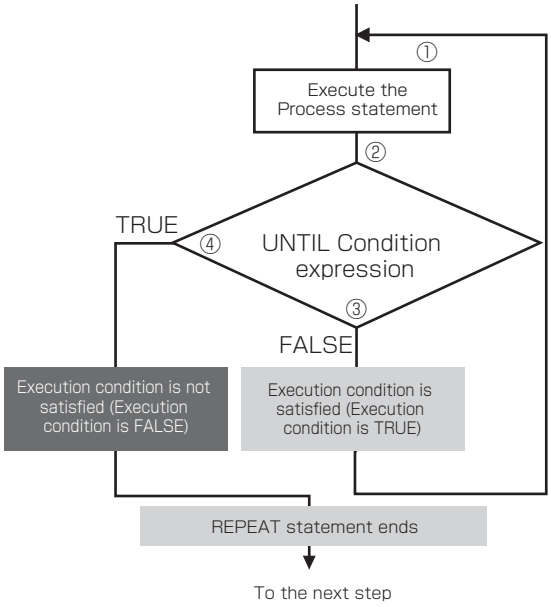
*1 1When the Condition expression is TRUE, Loop Control is ended

Expression example	Description

REPEAT
uiValue := uiValue + 5;
UNTIL ␣ uiValue >= 100
END_REPEAT;
	Performs the process from the next step.
	Adds 5 to uiValue.
	When the value of uiValue is not 100 or more, returns to the leading of the sentence (REPEAT).* 2

*2 2If the value of uiValue is 100 or more, executes from the next line of END_REPEAT without executing the Process statement.

Processing flow



① Executes the Process statement.

② Judges "TRUE" or "FALSE" by the UNTIL Condition statement, and decides whether to continue Loop process.

③ If the Condition expression is "FALSE", proceeds to step ①.

④ If the Condition expression is "TRUE", proceeds to the next step.

EXIT Processing

It is used to forcibly end the Loop Process (exit from the loop).

EXIT	
Basic expression	**Description**
<Loop control statement starts> EXIT; <Loop control statement ends>	Writes in Loop Process.* 1 Forcibly ends the Loop Process. Loop control statement ends here.

*1 EXIT must be programmed in the Loop process statement.

Expression example	Description

FOR␣uiValue1 := 0␣TO␣50␣BY␣2 DO
uiValue2 := uiValue2 + 5;
IF␣uiValue2 > 70␣THEN
EXIT;
END_IF;
END_FOR;
	Loops until uiValue1 goes from 0 to 50.
	Adds 5 to uiValue2.
	If the value of uiValue2 exceeds 70,
	forcibly ends the Loop Control (FOR~END_FOR).
	IF statement ends here.
	Adds 2 to uiValue1 and returns to the leading of the FOR statement.

* Except FOR, Loop process statement can also be used in WHILE/REPEAT statements.

Point

EXIT must be written inside the Loop process statement.
If it is written outside the Loop process statement, the conversion error will occur.

RETURN Processing

It is used to forcibly end the execution of the program (Module/Macro/Function block/Function) in which RETURN is written.

RETURN	
Basic expression	**Description**
<Process statement> RETURN; <Process statement>	Ends the execution of the written program.

Expression example	Description
FOR␣uiValue1 := 0␣TO␣50␣BY␣2␣DO uiValue2 := uiValue2 + 5; IF␣uiValue2 > 70␣THEN RETURN; END_IF; END_FOR;	Loops until uiValue1 goes from 0 to 50. Adds 5 to uiValue2. If the value of uiValue2 exceeds 70, jumps to the END instruction of this program and ends the execution of the program. IF statement ends here. Adds 2 to uiValue1 and returns to the leading of the FOR statement.

Precautions on Using Control Statement

Loop control statement and processing time

With the Loop control statements (FOR, WHILE, REPEAT), Loop Process programming can be easily performed by ST. However, it should be performed within the allowable scanning time of the PLC.

■About processing time

Let's discuss the scanning time when writing a program with multiple cycles.

Example) To calculate the sum of 1 to 200,000 by 1+2+3+····200,000.

ST program

```
udiIdx := 0;
udiSum := 0;
WHILE␣udiIdx < 200000␣DO          //Cycles steps ① to ② 200,000 times
                                  //Executes Loop Control.
udiIdx := udiIdx + 1;             //①
udiSum := udiSum + udiIdx;        //②
END_WHILE;
```

Available Devices vary by unit. Refer to " " page 3-24.

For details, refer to "WHILE statement (Pre-judge Loop Control)"(page 4-13).

The processing time of above sample program is as follows.

Time for 1 loop	→	Approximately 1.65μs
Time for 200,000 loops	→	Approximately 1.65μs×200,000=Approximately 330,000μs(330ms)

*The above processing time is just a reference example. During actual processing, it depends on the number of connected expansion units and the usage environment.

In this case, there is no problem if the Loop Process is performed only once. If the Loop Process is performed 200,000 times, the total processing time exceeds the allowable scanning time (300 ms).

Using a program in a ST enables easy to simply write a complex program or Loop Process. However, it is necessary to pay attention to the scanning time.

Reference When Ladder program exceeds allowable scanning time (300ms), the scanning time overflow may occur (error No.:30)

When the processing time of Loop control statement is longer (processing complicated operation), scanning time overflow may occur, even with few cycles.

To avoid scanning time over, the writing method needs to be changed to perform the process with multiple scans (or each scan).

Example) To calculate the sum of 1 to 200,000 by 1+2+3+····200,000.
(programming without scanning time overflow)

ST program

```
IF ␣ bFlag = FALSE ␣ THEN
bFlag := TRUE;                    //Flag during operation
udiIdx := 0;                      //Initialization of udiIdx
udiSum := 0;                      //Initialization of udiSum
END_IF;
IF ␣ udiIdx < 200000 ␣ THEN
udiIdx := udiIdx + 1;             //Counter
udiSum := udiSum + udiIdx;        //Calculation of the sum of values 1 to 200,000
END_IF;
```

Description

When bFlag is FALSE, 0 is stored (initialized) in udiIdx and udiSum, and bFlag is set to TRUE.
When bFlag is TRUE and udiIdx is less than 200,000, 1 is added to the current value of udiIdx, and the values from 1 to 200,000 are added in order and stored in udiSum (the operation is executed for each scan).

For details, refer to "IF ~ statement (Condition branch ①)"(page 4-7).

■ About Timer operation in Loop control statement

Timer cannot be used to forcibly end Loop control statement.
Timer/Counter contact do not change in Loop Control, the following ST will be in an infinite loop.

[ST programming that should not be executed]

```
TMH(timer, 1);                    //10ms Timer
WHILE ␣ bFlag = FALSE ␣ DO        //When bFlag becomes TRUE, the Loop process ends.
udiValue := udiValue + 1;
udiSum := udiSum + udiValue;
IF ␣ timer ␣ THEN                 /EXIT is executed when the Timer contact timer becomes TRUE
EXIT;                             //Process statement is not executed because the Contact of the Timer does not change.
END_IF;
END_WHILE;
```

「TMR function」(page 5-13)

Precautions on ST Programming

Precautions on ST Programming

Timer/Counter processes the Current value and Contact state with the same Variable.
Contact state can be processed by Variable itself.
To get Current value, specify the object Variable as the Argument of CURVAL Function.

● Reading the Current value of Timer/Counter

udiValue1 := CURVAL(timer);
udiValue2 := CURVAL(counter);
* Current value of the Variable cannot be rewritten directly from ST. Use the Timer/Counter Function.

● Contact of Timer/Counter

SET (timer, bFlag1); //When the Timer contact timer is TRUE, bFlag1 turns ON (hold).
SET (counter = FALSE, bFlag2); //When the Counter contact counter is FALSE, bFlag2 turns ON (hold).

■Precautions on using Timer Function

Timer Function is provided an execution condition in the Expression. Unlike other Functions, its execution condition also serves as a Timer reset instruction. The execution condition of Timer Function can be omitted, but in that case it cannot be reset and continues to operate as an always-ON Function.
TMR (timer, 500); // When the execution condition is omitted

Description The Contact of the Timer variable timer keeps ON 5 seconds after the program starts.
The Contact can be cleared only when RES (timer) is executed, the PLC power is turned off, or "RUN⇒PROG" switched.

■Precautions on using Timer Function in Box ST and IF statement

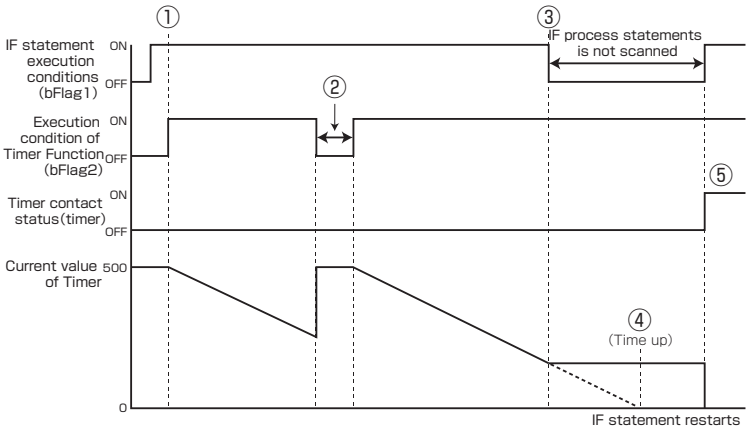
● Using Timer Function in IF statement of Area ST

Pay attention to the follows when using the Timer Function in IF statement of Area ST:

ST program

```
IF ␣ bFlag1 = TRUE ␣ THEN          //Write in the IF statement
TMR(bFlag2 = TRUE, timer, 500);
END_IF;
```

Description



①Timer contact timer operates while both the IF statement execution condition (bFlag1) and the Timer execution condition (bFlag2) are ON. It is a setting to turn on the Timer contact timer 5 seconds after the operation starts.

②When IF statement execution condition (bFlag1) is ON and the Timer execution condition (bFlag2) is OFF, the Timer is reset.

③Before the Timer starts to run to the end of the set time, if the Execution condition of IF statement turns OFF, even if the Execution condition of the Timer is ON, the Process statement (Timer Function) of the IF statement will not be scanned. The Timer Function seems to be in a paused state, but in fact it continues to run (because no reset command is received).

④However, even if the time is up, the Process statement is not scanned. Timer contact is not output and the Timer is in the standby state.

⑤When the execution condition of the next IF statement is satisfied, the Timer contact turns ON without Timer operation.

Workaround

If the Execution condition of the IF statement is OFF, this problem can be avoided by resetting the Timer Function included in the Process statement.

Change the ST program as follows.

ST program

```
IF ␣ bFlag1 = TRUE ␣ THEN          //Write in the IF statement
TMR(bFlag2 = TRUE, timer, 500);
ELSE                               //Reset the Timer
RES(timer);
END_IF;
```

Description

Timer contact timer operates while both the IF statement execution condition (bFlag1) and the Timer execution condition (bFlag2) are ON.

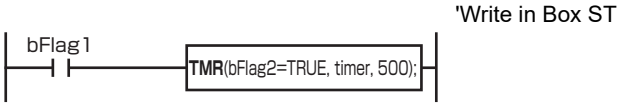
Turns Timer contact timer on after 5 seconds from operation starting.

Resets Timer (timer) is reset when IF statement execution condition (bFlag1) turns OFF.

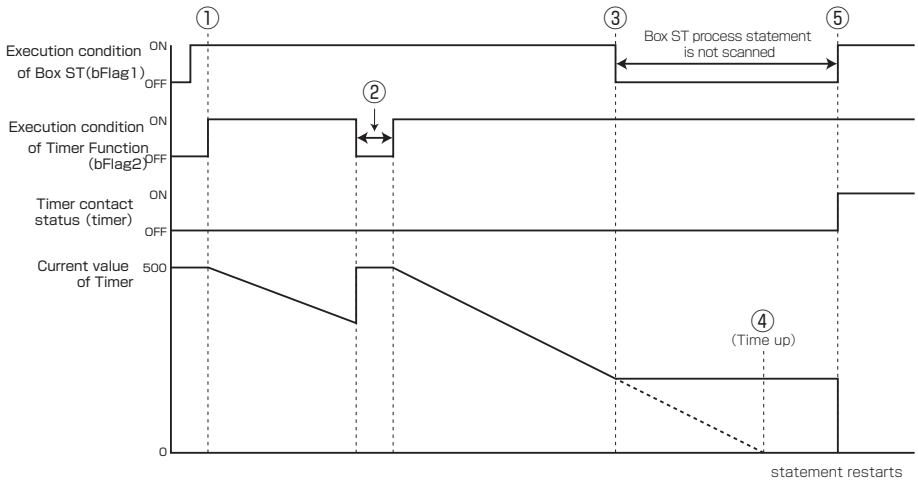
● Specifying Timer Function using Box ST

When using Timer Function in Box ST, pay attention to the same things as when using it in an IF statement.

ST program



Description

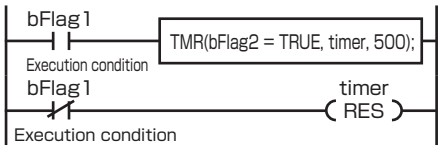


(1)Timer contact timer operates while both the Box ST execution condition (bFlag1) and the Timer execution condition (bFlag2) are ON. It is a setting to turn on the Timer contact timer 5 seconds after the operation starts.
(2) When Box ST execution condition (bFlag1) is ON and the Timer execution condition (bFlag2) is OFF, the Timer is reset.
(3) Before the Timer starts to run to the end of the set time, if the Execution condition of Box ST turns OFF, even if the Execution condition of the Timer is ON, the Process statement (Timer Function) of the Box ST will not be scanned. The Timer Function seems to be in a paused state, but in fact it continues to run (because no reset command is received).
(4) However, even if the time is up, the Process statement is not scanned. Timer contact is not output and the Timer is in the standby state.
(5) When the execution condition of the next Box ST is satisfied, the Timer contact turns ON without Timer operation.

Workaround

When the Execution condition of Box Script is OFF, resetting Timer Function contained in Box ST can avoid this problem. Change the ST program as follows.

ST program



Description

Timer is reset by the B contact of the Box ST execution condition.

5

Basic Functions

How to Read Functions 5-2
Contact Function. 5-3
Output Function 5-8
Bit Output Function 5-10
Timer/Counter Function 5-13
Direct I/O Function 5-18

How to Read Functions

BMOV Batch-transfers Word Block

[]:Omissible Arguments

The data type of Argument. which is actually recognized

Block moving

BMOV ([EN], Src, Dst, n);

Argument/ Return value	Description	Identification type .U	.S	.D	.L	.F	.DF	.B	.T	Constant
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	○
Src*1	Specifies the leading of Transfer source data block.	.U	.U	.D	.D	.D	.DF*2	—	—	○
Dst*1	Specifies the leading of Transfer destination data block.	Types same as Src.						—	—	—
n	Specifies Transfer data amount or its Save destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

Means that if an Argument of .D (UDINT) data type is specified for n, the Function will be recognized as an .U (UINT) type data and executed.

Whether or not Constants can be directly input in addition to Devices/Variables.

Point **You can use reordered Arguments in the Function by writing notations such as Src, Dst to expressions such as Src :=DM0.**

●Variable data types

Data type of a Variable to be specified by ST Function must be verified with the leading prefix.

Prefixes to be assigned are as follows:

Data type	Prefix	Data type	Prefix
BOOL	b	BOOL[n]	ab
INT	i	INT[n]	ai
DINT	di	DINT[n]	adi
UINT	ui	UINT[n]	aui
UDINT	udi	UDINT[n]	audi
REAL	r	REAL[n]	ar
LREAL	l	LREAL[n]	al
STRING	str		
TIMER	t		
COUNTER	c		

Example)i iAnalogData… INT type variable

audiSendData…Array type UDINT variable

■Description of Notation

The notation such as Dst+(1) indicates the address offset from Variable/Device specified by Dst.

- **For Devices**
  If Dst is specified as DM0, Dst (1) represents DM1.
- **For Variables**
  IfDst is specified as auiValue (1-word Array), it represents auiValue [1].
  If Dst is specified as adiData (2-word Array), it represents the upper word of adiData [0].
  If Dst is specified as Sample (Structure type variable), it represents a member offset by one word from Sample.

Contact Function

LDP Rises Derivative for Specified Bit

Load pulse
[R:=] LDP (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*2	.T	
In*1	Specifies the rising edge detect object.	—	—	—	—	—	—	.B	—	—
R	Return value. Omissible. Specifies the object to operate. If omitted, the operation result is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 Detects the status of the lowest bit when Word data is specified to In.
*2 Direct I/O Relays (DR) can be specified.
Bit position in Word data can be specified.

●Description of Operation

Turns R on for one scan when the rising edge (changing from OFF to ON) of In is detected.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Contact Instructions".

●Expression sample

bEdge := **LDP** (bEnable);

Point
Pay attention when using in Box ST or IF statement.
"Precautions on using Box ST", page 2-9

Reference
- You can also set Operation expression result as Execution condition.
- It can also be used in Execution conditions of other Functions.

LDF Falls Derivative for Specified Bit

Load pulse falling
[R:=] LDF (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*2	.T	
In*1	Specifies the falling edge detect object.	—	—	—	—	—	—	.B	—	—
R	Return value. Omissible. Specifies the object to operate. If omitted, the operation result is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 Detects the status of the lowest bit when Word data is specified to In.
*2 Direct I/O Relays (DR) can be specified.
Bit position in Word data can be specified.

●Description of Operation

Turns **R** on for one scan when the rising edge (changing from ON to ON) of **In** is detected.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Contact Instructions".

●Expression sample

bEdge := **LDF** (bEnable);

Point

Pay attention when using in Box ST or IF statement.
"Precautions on using Box ST", page 2-9

Reference

- You can also set Operation expression result as Execution condition.
- It can also be used in Execution conditions of other Functions.

LDPB Invert-rises Derivative for Specified Bit

Load pulse fall bar
[R:=] LDPB (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*2	.T	
In*1	Specifies the rising edge detect object.		–	–	–	–	–	.B	–	–
R	Return value. Omissible. Specifies the object to operate. If omitted, the operation result is stored in Internal register.	–	–	–	–	–	–	.B	–	–

*1 Detects the status of the lowest bit when Word data is specified to **In**.
*2 Direct I/O Relays (DR) can be specified.
Bit position in Word data can be specified.

●Description of Operation

Turns **R** on for one scan when the rising edge (changing from OFF to ON) of **In** is detected.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Contact Instructions".

●Expression sample

bEdge := **LDPB** (bEnable);

Point

Pay attention when using in Box ST or IF statement.
"Precautions on using Box ST", page 2-9

LDFB Invert-falls Derivative for Specified Bit

Load pulse bar
[R:=] LDFB (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*2	.T	
In*1	Specifies the falling edge detect object.	—	—	—	—	—	—	.B	—	—
R	Return value. Omissible. Specifies the object to operate. If omitted, the operation result is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 Detects the status of the lowest bit when Word data is specified to **In**.

*2 Direct I/O Relays (DR) can be specified.
Bit position in Word data can be specified.

●Description of Operation

Turns **R** on for one scan when the rising edge (changing from ON to ON) of **In** is detected.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Contact Instructions".

●Expression sample

bEdge := **LDFB** (bEnable);

Point

Pay attention when using in Box ST or IF statement.
"Precautions on using Box ST", page 2-9

Bit Contact Function

BLD Word Data Bit-position Info

Bit load
[R:=] BLD (In, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*3	.T	
In*1	Specifies the target to read Bit status.	.U	.U	.D	.D	–	–	–	–	–
Pos*2	Specifies Bit position to determine the target ON/OFF. (16-bit BIN data)	.U	.U	.U	.U	–	–	–	–	○
R	Return value. Omissible. Specifies the object to operate. If omitted, the operation result is stored in Internal register.	–	–	–	–	–	–	.B	–	–

*1 Specifies the Bit in the position [specified Bit +Pos] when Bit data is specified to In. Operates normally even when straddling channels.
*2 For the Bit position value, specifies to 0 to 15 when the suffix of Pos is .U (UINT), and 0 to 31 when .D (UDINT).
*3 Direct I/O Relays (DR) can be specified.
Bit position in Word data can be specified.

●Description of Operation

Returns the ON/OFF status of In that in the Bit position specified by Pos, to R.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Bit Contact Instructions".

●Expression sample

bFlag := **BLD** (uiValue, INT#10);

Point

When writing a non-constant in Pos, if the operation result exceeds the range according to the suffix of In (0 to 15 for .U, 0 to 31 for .D), the operation is as follows.
If the suffix of In is .U: the value of the lower 4 bits of Pos is used.
If the suffix of In is .U: the value of the lower 5 bits of Pos is used.

BLDB Word Data Bit-position Invert Info

Bit load bar
[R:=] BLDB (In, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*3	.T	
In*1	Specifies the target to read Bit status.	.U	.U	.D	.D	—	—	—	—	—
Pos*2	Specifies Bit position to determine the target ON/OFF. (16-bit BIN data)	.U	.U	.U	.U	—	—	—	—	○
R	Return value. Omissible. Specifies the object to operate. If omitted, the operation result is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 Specifies the Bit in the position [specified Bit +Pos] when Bit data is specified to In. Operates normally even when straddling channels.
*2 For the Bit position value, specifies to 0 to 15 when the suffix of Pos is .U (UINT), and 0 to 31 when .D (UDINT).
*3 Direct I/O Relays (DR) can be specified.
Bit position in Word data can be specified.

●Description of Operation

Inverts to return the ON/OFF status of In that in the Bit position specified by Pos, to R.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Bit Contact Instruction".

●Expression sample

bFlag := **BLDB** (uiValue, INT#10);

Point

When writing a non-constant in Pos, if the operation result exceeds the range according to the suffix of In (0 to 15 for .U, 0 to 31 for .D), the operation is as follows.
If the suffix of In is .U: the value of the lower 4 bits of Pos is used.
If the suffix of In is .U: the value of the lower 5 bits of Pos is used.

Output Function

SET Turns On Specified Bit Output

Setting
SET ([EN], InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*1	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut	Specifies the target to be held after ON.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Direct I/O Relays (DR) can be specified.
Bit position in Word data can be specified.

●Description of Operation

When Execution condition is TRUE, turns **InOut** on. Once turned on, it is held in the ON state even if the execution condition becomes FALSE. (The ON target is turned off by RES Function)

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Output Instructions".
"RES function", page 5-8

●Expression sample

SET (bEnable, bStart);

When bEnable turns on, bStart is turned on.

RES Turns Off Specified Bit Output

Reseting
RES ([EN], InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B*1	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut	Specifies the target to be held after ON.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Direct I/O Relays (DR) can be specified
Bit position in Word data can be specified.

●Description of Operation

When Execution condition is TRUE, turns **InOut** off. (The target turned on by SET Function is turned off by RES Function)

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Output Instruction".

"SET function", page 5-8

●Expression sample

RES (bComp, bStart);

When bComp turns on, bStart is turned off.

Bit Output Function

BOUT Outputs Word Data Bit-position

Bit outputting
BOUT ([EN], InOut, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1	Specifies the target to output Input status.	.U	.U	.D	.D	—	—	—	—	—
Pos*2	Specifies Bit position to output.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 Outputs to the object Relay calculated by the value of [object+Pos] when a Bit data is specified for InOut. Operation across channels is also available.

*2 For the value of Pos, specifies 0 to 15 when the target suffix is .U (UINT), and 0 to 31 when .D (UDINT).

●Description of Operation

When Execution condition is TRUE, turns on the Bit of InOut that in specified Pos, and when is FALSE, off.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Bit Output Instructions".

●Expression sample

- InOut: for 16-bit target
  BOUT (bEnable, uiTarget, INT#12);
- InOut: for 32-bit target
  BOUT (iAnalogData > INT#1000, diTarget, INT#28);

Point

When writing a non-constant in Pos, if the operation result exceeds the range according to the suffix of InOut (0 to 15 for .U, 0 to 31 for .D), the operation is as follows.
If the suffix of InOut is .U: the value of the lower 4 bits of Pos is used.
If the suffix of InOut is .U: the value of the lower 5 bits of Pos is used.

BOUB Invert-outputs Word Data Bit-position

Bit output bar
BOUB ([EN], InOut, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1	Specifies the target to output Input status.	.U	.U	.D	.D	—	—	—	—	—
Pos*2	Specifies the target to be held after ON.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Outputs to the Bit in the position of [specified Bit + **Pos**] when a Bit data is specified for **InOut**. Operation across channels is also available.
*2 For the value of **Pos**, specifies 0 to 15 when the target suffix is .U (UINT), and 0 to 31 when .D (UDINT).

●Description of Operation

When Execution condition is TRUE, turns on the Bit of **InOut** that in specified **Pos**, and when is FALSE, off.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Bit Output Instruction".

●Expression sample

- **InOut**: **for 16-bit target**
  BOUB (bEnable, uiTarget, INT#12);
- **InOut**: **for 32-bit target**
  BOUB (iAnalogData > INT#1000, diTarget, INT#28);

Point

When writing a non-constant in **Pos**, if the operation result exceeds the range according to the suffix of **InOut** (0 to 15 for .U, 0 to 31 for .D), the operation is as follows.
If the suffix of **InOut** is .U: the value of the lower 4 bits of **Pos** is used.
If the suffix of **InOut** is .U: the value of the lower 5 bits of **Pos** is used.

BSET Turns On Word Data Bit-position Output

Bit setting
BSET ([EN], InOut, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut[*1]	Specifies the target to be held after ON.	.U	.U	.D	.D	—	—	—	—	—
Pos[*2]	Specifies the Bit position to be turned ON and held.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Outputs to the Bit in the position of [specified Bit + **Pos**] when a Bit data is specified for **InOut**. Operation across channels is also available.
*2 For the value of **Pos**, specifies 0 to 15 when the target suffix is .U (UINT), and 0 to 31 when .D (UDINT).

●Description of Operation

When Execution condition is TRUE, turns on the Bit of **InOut** that in specified **Pos**. Once turned on, it is held in the ON state even if Execution condition becomes FALSE.
(ON target can be turned off by BTES Function)

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Bit Output Instruction".
"BRES function", page 5-12

●Expression sample

- **InOut**: **for 16-bit target**

BSET (bEnable, uiTarget, INT#10);

- **InOut: for 32-bit target**
  BSET (iAnalogData > INT#1000, diTarget, INT#20);

Point

When writing a non-constant in **Pos**, if the operation result exceeds the range according to the suffix of **InOut** (0 to 15 for .U, 0 to 31 for .D), the operation is as follows.
If the suffix of **InOut** is .U: the value of the lower 4 bits of **Pos** is used.
If the suffix of **InOut** is .U: the value of the lower 5 bits of **Pos** is used.

BRES Turns Off Word Data Bit-position Output

Bit resetting
BRES ([EN], InOut, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1	Specifies the target to be turned off.	.U	.U	.D	.D	—	—	—	—	—
Pos*2	Specifies the Bit position to be turned off.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Outputs to the Bit in the position of [specified Bit + **Pos**] when a Bit data is specified for **InOut**. Operation across channels is also available.
*2 For the value of **Pos**, specifies 0 to 15 when the target suffix is .U (UINT), and 0 to 31 when .D (UDINT).

●Description of Operation

When Execution condition is TRUE, turns off the Bit of **InOut** that in specified **Pos**.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Bit Output Instructions".
"BSET function", page 5-11

●Expression sample

- **InOut: for 16-bit target**
  BRES (bEnable, uiTarget, INT#5);
- **InOut: for 32-bit target**
  BRES (iAnalogData > INT#1000, diTarget, INT#30);

Point

When writing a non-constant in Pos, if the operation result exceeds the range according to the suffix of InOut (0 to 15 for .U, 0 to 31 for .D), the operation is as follows.
If the suffix of InOut is .U: the value of the lower 4 bits of Pos is used.
If the suffix of InOut is .U: the value of the lower 5 bits of Pos is used.

Timer/Counter Function

TMR 100 ms 32-bit Down Timer

Timer
TMR ([EN], Timer, In100ms);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition that also serves as a Reset Function (when OFF, resets TMR Function). Omissible. If omitted, resets Timer separately by RES Function.	—	—	—	—	—	—	.B	—	—
Timer	Specifies the Timer. (T0 to T3999)	—	—	—	—	—	—	—	—	—
In100ms *1	Specifies Set value of Timer in 100ms units. (0 to 4294967295)	.D	.D	.D	.D	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When a Bit data is specified to **In100ms**, continuous 32 bits are occupied, and when a Word data, two words.

●Description of Operation

When Execution condition is TRUE, turns on the Timer specified by **Timer**. The value specified in **In100ms** becomes the Current value and is decremented by-1 every 100ms. When Current value becomes 0, the specified Timer (Contact) turns on. When Execution condition is FALSE, turns off the specified Timer and resets Current value.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Timer/Counter Instructions".

●Expression sample

TMR (bEnable, tTimer, INT#100);

Reference The error of TMR is "within ± (100 ms + 1 scan time)".

Point
- **The same Device No. as other TMR/TMH/TMS/TMU Function (instructions) and UDT instruction cannot be used.**
- **Operation expression can not be specified as Set value. The range processable by Operation expression is -2147483648 to +2147483647. Pay attention in the following cases.**
- **If the result value of Operation expression is negative, the value obtained by adding 4294967295 to the result value is set as the Set value.**
- **Pay attention when using in a Box ST or when writing it in a subroutine.**
  "Precautions on using Timer Function in Box ST and IF statement", page 4-18

TMH 10ms32-bit Down Timer

High-speed timer
TMH ([EN], Timer, In10ms);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition that also serves as a Reset Function (when OFF, resets TM Function). Omissible. If omitted, resets Timer separately by RES Function.	—	—	—	—	—	—	.B	—	—
Timer	Specifies the Timer. (T0 to T3999)	—	—	—	—	—	—	—	—	—
In10ms*1	Specifies Set value of Timer in 10ms units. (0 to 4294967295)	.D	.D	.D	.D	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When a Bit data is specified to **In10ms**, continuous 32 bits are occupied, and when a Word data, two words.

●Description of Operation

When Execution condition is TRUE, turns on the Timer specified by **Timer**. The value specified in **In10ms** becomes the Current value and is decremented by-1 every 10ms. When Current value becomes 0, the specified Timer (Contact) turns on. When Execution condition is FALSE, turns off the specified Timer and resets Current value.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Timer/Counter Instructions".

●Expression sample

TMH (bEnable, tTimer, INT#100);

Reference TMH error is "within ± (10ms + 1 scan time)".

Point
- **The same Device No. as other TMR/TMH/TMS/TMU Function (instructions) and UDT instruction cannot be used.**
- **Operation expression can not be specified as Set value. The range processable by Operation expression is -2147483648 to +2147483647. Pay attention in the following cases.**
- **If the result value of Operation expression is negative, the value obtained by adding 4294967295 to the result value is set as the Set value.**

Pay attention when using in a Box ST or when writing it in a subroutine.
"Precautions on using Timer Function in Box ST and IF statement", page 4-18

"TMR function", page 5-13

TMS 1 ms 32-bit Down Timer

High-speed 1ms timer
TMS ([EN], Timer, In1ms);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition that also serves as a Reset Function (when OFF, resets MS Function). Omissible. If omitted, resets Timer separately by RES Function.	—	—	—	—	—	—	.B	—	—
Timer	Specifies the Timer. (T0 to T3999)	—	—	—	—	—	—	—	—	—
In1ms*1	Specifies Set value of Timer in 1ms units. (0 to 4294967295)	.D	.D	.D	.D	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When a Bit data is specified to **In1ms**, continuous 32 bits are occupied, and when a Word data, two words.

●Description of Operation

When Execution condition is TRUE, turns on the Timer specified by **Timer**. The value specified in **In1ms** becomes the Current value and is decremented by-1 every 1ms. When Current value becomes 0, the specified Timer (Contact) turns on. When Execution condition is FALSE, turns off the specified Timer and resets Current value.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Timer/Counter Instructions".

●Expression sample

TMS (bEnable, tTimer, INT#100);

Reference TMS error is "within ± (1ms + 1 scan time)".

Point
- **The same Device No. as other TMR/TMH/TMS/TMU Function (instructions) and UDT instruction cannot be used.**
- **Operation expression can not be specified as Set value. The range processable by Operation expression is -2147483648 to +2147483647. Pay attention in the following cases.**
- **If the result value of Operation expression is negative, the value obtained by adding 4294967295 to the result value is set as the Set value.**

Pay attention when using in a Box ST or when writing it in a subroutine.
"Precautions on using Timer Function in Box ST and IF statement", page 4-18

"TMR function", page 5-13

TMU 10μs32-bit Down Timer

High-speed 10μs timer
TMU ([EN], Timer, In10μs);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition that also serves as a Reset Function (when OFF, resets TMU Function). Omissible. If omitted, resets Timer separately by RES Function.	—	—	—	—	—	—	.B	—	—
Timer	Specifies the Timer. (T0 to T3999)	—	—	—	—	—	—	—	—	—
In10μs*1	Specifies Set value of Timer in 10μs units. (0 to 4294967295)	.D	.D	.D	.D	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When a Bit data is specified to **In10μs**, continuous 32 bits are occupied, and when a Word data, two words.

●Description of Operation

When Execution condition is TRUE, turns on the Timer specified by **Timer**. The value specified in **In10μs** becomes the Current value and is decremented by-1 every 10μs. When Current value becomes 0, the specified Timer (Contact) turns on. When Execution condition is FALSE, turns off the specified Timer and resets Current value.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Timer/Counter Instructions".

●Expression sample

TMU (bEnable, tTimer, INT#100);

Reference TMH error is "within ± (10μs + 1 scan time)".

Point
- **The same Device No. as other TMR/TMH/TMS/TMU Function (instructions) and UDT instruction cannot be used.**
- **Operation expression can not be specified as Set value. The range processable by Operation expression is -2147483648 to +2147483647. Pay attention in the following cases.**
- **If the result value of Operation expression is negative, the value obtained by adding 4294967295 to the result value is set as the Set value.**

  Pay attention when using in a Box ST or when writing it in a subroutine.
  "Precautions on using Timer Function in Box ST and IF statement", page 4-18

"TMR function", page 5-13

CNT 32-bit Up Counter

Counter
CNT ([EN], Counter, PV, CD);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition that also serves as a Reset Function (when OFF, resets CNT Function). Omissible. If omitted, resets Counter separately by RES Function.	—	—	—	—	—	—	.B	—	—
Counter	Specifies the Counter. (C0 to C3999)	—	—	—	—	—	—	—	—	—
PV*1	Specifies Set value of Counter. (0 to 4294967295)	.D	.D	.D	.D	—	—	—	—	○
CD	Specifies the target to be used for Counter input.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When a Bit data is specified to **PV**, continuous 32 bits are occupied, and when a Word data, two words.

●Description of Operation

If the Execution condition of Counter is TRUE, Current value of the specified Counter is added by 1 at the rising edge of Counter input specified by **CD**. When Current value reaches Set value specified in **PV**, **Counter** turns ON. If Execution condition is FALSE, turns off specified Counter and resets Current value.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Timer/Counter Instructions".

●Expression sample

CNT (bEnable, cCounter, INT#100, bDetect);

Point

- **The same Device No. as the OUTC/UDC instructions cannot be used.**
- **Operation expression can not be specified as Set value. The range processable by Operation expression is -2147483648 to +2147483647. Pay attention in the following cases.**
- **If the result value of Operation expression is negative, the value obtained by adding 4294967295 to the result value is set as the Set value.**

CURVAL Gets Timer/Counter Current Value

Timer/Counter current value getting

[R:=] CURVAL (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the target data to read Current value.	—	—	—	—	—	—	—	—	—
R*1	Return value that stores Current value of Timer/Counter. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a Type different from the operation target is specified to **R**, Type conversion is automatically executed during conversion.

●Description of Operation

Reads Current value of Timer/Counter stored in **In** and stores the result in **R**.

●Expression sample

udiValue1 := **CURVAL**(timer);

udiValue2 := **CURVAL**(counter);

*Unavailable to rewrite Current value of Variable directly from ST. Use Timer/Counter Functions.

"About the operation flags (CR2009 to CR2012)", page 1-8

Direct I/O Function

RFSX Input-refreshes

Refresh X
RFSX ([EN], Start, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Start	Specifies the leading of the target to refresh the input status	—	—	—	—	—	—	.B	—	—
Count*1	Specifies the number of Relays to refresh the input status. (1 to 256)*1	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operates with continuous 16 bits.
When specifying Relays other than the leading (R002, R1012, etc.) of a channel, processes with 16 bits across the next channel.
The range of specifiable Relays is 1 to 256.

●Description of Operation

To use the input-refresh operation that can be executed at any time in one scan instead of the batch-refresh operation that is performed after executing END instruction of Ladder program, use this Function.
When Execution condition is TRUE, input-refreshes Bit data status for the number of Relays that specified by **Count**, starting with **Start**.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Direct I/O Instructions".

●Expression sample

RFSX (bFlag1, R30000, INT#128);

RFSY Output-refreshes

Refresh Y
RFSY ([EN], Start, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Start	Specifies the leading of the target to refresh the output status.	—	—	—	—	—	—	.B	—	—
Count*1	Specifies the number of relays to refresh the output status. (1 to 256)*1	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operates with continuous 16 bits.
When specifying Relays other than the leading (R002, R1012, etc.) of a channel, processes with 16 bits across the next channel.
The range of specifiable Relays is 1 to 256.

●Description of Operation

To use the output-refresh operation that can be executed at any time in one scan instead of the batch-refresh operation that is performed after executing END instruction of Ladder program, use this Function.
When Execution condition is TRUE, output-refreshes Bit data status for the number of Relays that specified by **Count**, starting with **Start**.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Direct I/O Instructions".

●Expression sample

RFSY (bFlag1, R30200, INT#32);

6

Application Functions

Flow Function 6-2
Function Block Function 6-4
Index Register Function 6-7
Indirect Designation Function 6-9
File Register Function 6-13
Unit Program Function 6-15
Buffer Memory Function. 6-21
Unit Type Identification Function 6-23
Array Element Amount Get Function 6-24
Word Size Get Function 6-25

Flow Function

CALL Starts Specified Subroutine

Calling
CALL ([EN], In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In	Specifies the sub-routine No. to startup. (0 to 99)*1	—	—	—	—	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 In is specified within the range of 0 to 99. Cannot be specified by 16# (HEX).

●Description of Operation

When Execution condition is TRUE, turns on the subroutine specified by In.
For details, refer to "Instruction Reference Manual" and "Flow Instructions".

●Expression sample

CALL (bFlag, INT#10);

Reference Subroutines can be nested up to 16 layers (nested structure).

Point
- **Only Constants can be specified as the sub-routine No. Devices or Expressions containing devices cannot be used.**
- **Sub-routine No. constants cannot be specified by 16# (HEX).**
- **SBN/RET instructions are not available in ST Language Function. Program them in Ladder.**

ECALL Calls Inter-Module Subroutine

ECALL
ECALL ([EN], Name, No);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Name	Specifies Module name with String.	—	—	—	—	—	—	—	—	○*1
No	Specifies the subroutine No. to call. (0 to 99)*2	—	—	—	—	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 Enter Module name set in "KV STUDIO" by enclosing it in ' '.
*2 No is specified within the range of 0 to 99. Cannot be specified by 16# (HEX).

●Description of Operation

When Execution condition is TRUE, turns on the sub-routine program specified by Module No that specified by Name.
All types of Module (every scan, initialization, standby, fixed period) can be specified except for Function/Function block/Macro.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Flow Instructions".

●Expression sample

ECALL (bFlag, 'NewModule', INT#1);

Point

- **Sub-routine No. constants cannot be specified by 16# (HEX).**
- **SBN/RET instructions are not available in ST Language Function. Program them in Ladder.**

Function Block Function

FBSTRT Starts Function Block

Function block starting
FBSTRT ([EN], 'FuncBlock', Instance, Argument);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
FuncBlock	Specifies Function block name with String.	—	—	—	—	—	—	—	—	○*2
Instance	Specifies Instance with String.	—	—	—	—	—	—	—	—	○
Argument*1	Specifies the Argument.	—	—	—	—	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 Argument amount and attributes are set in the Argument settings that set in the executed Function block.
"Instruction Reference Manual" "Function Block Instructions"
*2 Enter Function block name set in "KV STUDIO" by enclosing it in ' '.

●Description of Operation

When Execution condition is TRUE, executes the Instance specified by **Instance** of Function block that specified by **FuncBlock**. Instance operation continues even if Execution condition is turned off. When FEND instruction is executed in Function block, the operation stops.
Delivers Argument specified by **Argument** to Function block. Function block processes Arguments according to the data type set in Argument settings.

For details, refer to "Instruction Reference Manual" and "Function Block Instructions".

●Expression sample

FBSTRT (bEnable, 'NewFunctionBlock', FunctionA, uiValue);

Reference

Writing with named Arguments eliminates the need to change the order in which Arguments are specified even if their order changes in Function block argument settings. Write named Arguments as follows.
FBSTRT(R3000, 'Function block A', Instance B, Input1:=1, OutPut1=>DM10, ENO=>MR000)
IN/IN-OUT Arguments are specified with :=, while OUT Arguments are specified with =>. ENO can be omitted if not used.
Unit arguments cannot be written in named Arguments. Use Ladder program to write the Function block in which the Unit argument has been set.

Point

- **Strings other than Function block names enclosed by ' ' cannot be written as Function block name.**
- **Instance no need to be enclosed by ' '.**
- **It is also possible to call Function block using Function block instance. For information on how to call a Function block, refer to "Calling a Function block" (pages 2-27).**
  Reference example) FunctionA(uiValue);
- **When calling a Function, specify the Function name to call it directly. For information on how to call a Function refer to "Calling a Function block" (pages 2-27).**
  Reference example) NewFunction(uiValue);
- **For details,** refer to **"KV-8000 Series User's Manual" and "Function Block Instructions".**

FEND Ends Function Block

Function block ending
FEND ([EN]);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When Execution condition is TRUE, stops the Function block in which the FEND function has been written.

For details, refer to "Instruction Reference Manual" and "Function Block Instructions".

●Expression sample

FEND (bFlag);

Reference 「FBSTRT function」(page 6-4)

Point **FEND Function can only be used within a Function block.**

MDSTRT Starts Module Execution

Module starting
MDSTRT ([EN], Name);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Name	Specifies Module name with String.	—	—	—	—	—	—	—	—	○*1
R	None	—	—	—	—	—	—	—	—	—

*1 1Enter Module name set in "KV STUDIO" by enclosing it in ' '.

●Description of Operation

When Execution condition is TRUE, turns on the Module specified by **Name**.
The Module turned on operates from next scan.
No operation is executed if the Module is already running.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Module Instructions".

●Expression sample

MDSTRT (bEnable, 'NewModule');

Point

- **At the start of the module execution, the differential execution type instruction initial state turns on when the rising edge is detected, and turns off when the falling edge is detected. The differential will not be detected immediately at the start of execution.**
- **If MDSTRT/MDSTOP Function (Instruction) is executed multiple times in the same scan for the same Module, the Module instruction executed last will be followed.**
- **Execution of an initialized Module cannot be stopped or started.**

MDSTOP Stops Module Execution

Module stopping

MDSTOP ([EN], Name);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Name	Specifies Module name with String.	—	—	—	—	—	—	—	—	○*1
R	None	—	—	—	—	—	—	—	—	—

*1 Enter Module name set in "KV STUDIO" by enclosing it in ' '.

●Description of Operation

When Execution condition is TRUE, turns off the Module specified by **Name**.
No operation is executed if the Module is already stopped.
Local module can be also stopped.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Module Instructions".

●Expression sample

MDSTOP (bDisable, 'NewModule');

Point

- **If MDSTRT/MDSTOP Function (Instruction) is executed multiple times in the same scan for the same Module, the Module instruction executed last will be followed.**
- **Do not stop all Modules. If you have stopped all Modules, use one of the following methods to restore them. (Unrecoverable unless you stop the program once.)**
  - **Turn off the power of the CPU unit and then turn it on again.**
  - **Switch the "RUN-PROG selector" of the CPU unit to "PROG mode" once, and then switch to RUN mode again.**
  - **Turn off the program RUN and then turn it on again on "KV STUDIO".**
- **Execution of an initialized Module cannot be stopped or started.**

Index Register Function

ZPUSH Batch-saves Index Register

Z pushing
ZPUSH ([EN], Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Dst*1	Specifies the leading of Save destination data block.	S*2	S*2	.L*3	.L*3	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 T/C/CR/Z cannot be used.

*2 When the save destination type is specified in .S:
- Occupies consecutive 13 words. When nesting functions, occupies "number of saves x 12 words + 1" words.
- Occupies only the lower 16 bits.

*3 When the type of Save destination is specified in .L:
Occupies consecutive 25 words. When nesting, occupies "number of saves x 24 words + 1" words.

●Description of Operation

When Execution condition is TRUE, stores (saves) the index register values of Z1 to Z10 in the data block specified by **Dst**. After executing the Function, increments the value of [**Dst**+(0)] by 1.

Point
- **Set Dst + (0) to 0 before executing the "ZPUSH function" for the first time. Not required when executing the "ZPUSH function" from the second time onward.**
- **Use "ZPUSH Function" and "ZPOP Function" in pairs.**
- **Do not write in the range occupied by "ZPUSH Function" with other Functions, Instructions, etc.**
- **If the range of the data block exceeds the Device range or the Variable setting range , the Function will not be executed.**

「About the operation flags (CR2009 to CR2012)」(page 1-8)

Reference
"The "ZPUSH function" and ZPOP function" are programmed as the nested structures. When nesting, specify the same beginning.

Point
Z (Index register) is processed as .L type (signed 32-bit). If a 16-bit suffix (.S/.U) is specified for Dst, only the lower 16 bits of all Z are stored in the save destination, and upper 16 bits are discarded.
In this case, the saved value cannot be restored even if the ZPOP Function (instruction) is used.
To use a program string the .S range (-32768 to +32767) exceeded values in Z, do not specify .S for Dst.

●Expression sample

ZPUSH (bEnable, auiValue[0]);

For details, refer to "Instruction Reference Manual" and "Index Register Instructions".

ZPOP Batch-restores Index Register

Z Popping
ZPOP ([EN], Src);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1	Specifies the leading of Save destination data block.	S*2	S*2	.L*3	.L*3	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 T/C/CR/Z cannot be used.

*2 When the save destination type is specified in .S:
- occupies consecutive 13 words. When nesting functions, occupies "number of saves x 12 words + 1" words.
- Occupies only the lower 16 bits of Z.

*3 When the type of Save destination is specified in .L:
- occupies consecutive 25 words. When nesting, occupies "number of saves x 24 words + 1" words.

●Description of Operation

When Execution condition is TRUE, reads values of the data block (Index register stored in "ZPUSH Function") specified by Src into the Index registers of Z1 to Z10. After executing the Function, decrements the value of Src+(0) by -1.

Point
- **Use "ZPUSH Function" and "ZPOP Function" in pairs.**
- **The Function will not be executed if any of the following conditions are TURE:**
  - **When [Src+0] is 0**
  - **When the range of the data block exceeds Device range or Variable setting range**

「About the operation flags (CR2009 to CR2012)」(page 1-8)

Point
Z (Index register) is processed as .L type (signed 32-bit). If a 16-bit suffix (.S/.U) is specified for Src, only the lower 16 bits of all Z are stored in the save destination, and upper 16 bits are discarded.
In this case, the saved value cannot be restored even if the ZPOP Function (instruction) is used.
To use a program storing the .S range (-32768 to +32767) exceeded values in Z, do not specify .S for Src.

●Expression sample

ZPOP (bEnable, auiValue[0]);

For details, refer to "Instruction Reference Manual" and "Index Register Instructions".

Indirect Designation Function

ADRSET Stores Indirect Designation Address

Address setting
ADRSET ([EN], Src, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src	Specifies Indirect designation object.	.U	.S	.D	.D	.F	.DF	.B	.T	—
Dst*1*2	Specifies Address storage destination.	—	—	.D	.D	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Consecutive 2 words are used.
*2 CM cannot be specified.

●Description of Operation

When Execution condition is TRUE, stores the **Src** address in consecutive 2 words starting with **Dst**. After storage, if you write with Indirect designation (*Storage destination), the Indirect designation result will be the same as Direct designation.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Indirect Designation Instructions".

●Expression sample

ADRSET (bEnable, R0, udiAddress);

Point
- **The address storage destination always occupies consecutive 2 words starting with the target No.**
- **Before using Indirect designation (*Storage destination), be sure to set the Indirect designation address as the storage destination with ADRSET Function.**
- **Be sure to use the ADRSET, ADRINC, ADRDEC, ADRADD, and ADRSUB Functions to generate and operate addresses.**

ADRINC Increments Storage Address By 1

Address incrementing
ADRINC ([EN], InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1*2	Specifies Address storage destination for Indirect designation.	—	—	.D	.D	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Consecutive 2 words are used.
*2 CM cannot be specified.

●Description of Operation

When Execution condition is TRUE, sets the Indirect designation address stored in **InOut** to the next address. Indicates the address of the next word data when a Word data address is stored, and the address of the next bit data when a Bit data address is stored.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Indirect Designation Instructions".

●Expression sample

ADRINC (bEnable, udiAddress);

Point

- **The address storage object always occupies consecutive 2 words.**
- **Before using ADRINC Function, be sure to set the Indirect designation address as the address storage target with ADRSET Function.**
- **Be sure to use the ADRSET, ADRINC, ADRDEC, ADRADD, and ADRSUB Functions to generate and operate addresses.**
- **The Function will not be executed if any of the following conditions are TURE:**
  - **When the address stored in InOut is incorrect.**
  - **When the incremented address is incorrect.**

ADRDEC Decrements Storage Address By 1

Address decrementing
ADRDEC ([EN], InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1*2	Specifies Address storage destination for Indirect designation.	—	—	.D	.D	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Consecutive 2 words are used.
*2 CM cannot be specified.

●Description of Operation

When Execution condition is TRUE, sets the Indirect designation address stored in **InOut** to the previous address. Indicates the address of the previous word data when a Word data address is stored, and the address of the previous bit data when a Bit data address is stored.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Indirect Designation Instructions".

●Expression sample

ADRDEC (bEnable, udiAddress);

Point

- **The address storage object always occupies consecutive 2 words.**
- **Before using ADRDEC Function, be sure to set the Indirect designation address as the address storage target with ADRSET Function.**
- **Be sure to use the ADRSET, ADRINC, ADRDEC, ADRADD, and ADRSUB Functions to generate and operate addresses.**
- **The Function will not be executed if any of the following conditions are TURE:**
  - **When the address stored in InOut is incorrect.**
  - **When the decremented address is incorrect.**

ADRADD Adds Storage Address

Address adding

ADRADD ([EN], Offset, InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Offset	Specifies Address add data or its Storage destination.	.S	.S	.L	.L	—	—	—	—	○
InOut*1	Specifies Address storage destination for Indirect designation.	—	—	.D	.D	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 CM cannot be specified.

●Description of Operation

When Execution condition is TRUE, sets the address that obtained by adding Indirect designation address stored in **InOut** by the value specified by **Offset**. Indicates the Word data address obtained by adding only the address add value when a Word data address is stored, and the Bit data address obtained by adding only the address add value when a Bit data address is stored.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Indirect Designation Instructions".

●Expression sample

ADRADD (bEnable,INT#10, udiAddress);

Point

- **The address storage object always occupies consecutive 2 words.**
- **Before using ADRADD Function, be sure to set the Indirect designation address as the address storage target with ADRSET Function.**
- **Be sure to use the ADRSET, ADRINC, ADRDEC, ADRADD, and ADRSUB Functions to generate and operate addresses.**
- **The Function will not be executed if any of the following conditions are TURE:**
  - **When the address stored in InOut is incorrect.**
  - **When the incremented address is incorrect.**

ADRSUB Subtracts Storage Address

Address subtracting
ADRSUB ([EN], Offset, InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Offset	Specifies the address subtract data or its storage destination.	.S	.S	.L	.L	—	—	—	—	○
InOut*1	Specifies Address storage destination for Indirect designation.	—	—	.D	.D	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 CM cannot be specified.

●Description of Operation

When Execution condition is TRUE, sets the address that obtained by subtract Indirect designation address stored in **InOut** by the value specified by **Offset**. Indicates the Word data address obtained by subtracting only the address subtract value when a Word data address is stored, and the Bit data address obtained by subtracting only the address subtract value when a Bit data address is stored.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Indirect Designation Instructions".

●Expression sample

ADRSUB (bEnable,INT#10, udiAddress);

Point

- **The address storage object always occupies consecutive 2 words.**
- **Before using ADRSUB Function, be sure to set the Indirect designation address as the address storage target with ADRSET Function.**
- **Be sure to use the ADRSET, ADRINC, ADRDEC, ADRADD, and ADRSUB Functions to generate and operate addresses.**
- **The Function will not be executed if any of the following conditions are TURE:**
  - **When the address stored in InOut is incorrect.**
  - **When the decremented address is incorrect.**

File Register Function

FRSET Switches File Register Bank

File register bank switching
FRSET ([EN], In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In	Specifies switched Current bank No. (0 to 15)	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When Execution condition is TRUE, switches the Current bank No. of File register to the number specified by **In**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "File Register Instructions".

●Expression sample

FRSET (bEnable, INT#2);

FRSTM Batch-stores File Register

File register batch-storing
[R:=] FRSTM (Drive, Bank, File, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Drive	Specifies Drive No. to save. (0 to 1)	.U	.U	.U	.U	—	—	—	—	○
Bank	Specifies Bank No. of File register to be batch-saved . (0 to 15)	.U	.U	.U	.U	—	—	—	—	○
File	Specifies File No. when saving data. (0 to 999)	.U	.U	.U	.U	—	—	—	—	○
Result*1	Specifies the leading of Bit data to notify Save result.	—	—	—	—	—	—	.B	—	—
R	Return value. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, Operation result is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified to **Result**, consecutive 2 bits are used as the notify bits, and when Word data is specified, the each lowest bit of consecutive 2 words is used as the notify bit data.

●Description of Operation

Batch-stores the data in File register **Bank** with the **Drive** No. into the memory card in binary format by using the file name specified by **File**.
The write completion notification and abnormal end alarm notification are stored in **Result**.
Turns Return value on if FRSTM Function starts normally, off if fails.

Reference

- File name saved in Memory card or CPU memory is created as "\FR\FR (3 digits for File No.) .FRU".
  (Example) If 10 is specified for **File**, "\ FR \ FR010.FRU" will be created as its name on Memory card or CPU memory.
- The notification bit of **Result** has the following meanings.
  (Example) When MR2000 is specified

Notification bit		ON (1, TRUE)	OFF (0, FALSE)
MR2000	Completion notification	Completed	Uncompleted
MR2001	Abnormal notification	Abnormal completion	Normal completion

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "File Register Instructions".

●Expression sample

bFlag := **FRSTM** (INT#1, INT#10, INT#2, abFlag[0]);

FRLDM Batch-reads File Register

File register batch-reading
[R:=] FRLDM (Drive, Bank, File, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Drive	Specifies Drive No. to batch-read. (0 to 1)	.U	.U	.U	.U	—	—	—	—	○
Bank	Specifies Bank No. of File register to be batch-read. (0 to 15)	.U	.U	.U	.U	—	—	—	—	○
File	Specifies File No. when reading the data. (0 to 999)	.U	.U	.U	.U	—	—	—	—	○
Result[*1]	Specifies the leading of Bit data to notify Read result.	—	—	—	—	—	—	.B	—	—
R	Return value. Omissible. If omitted, Operation result is stored in Internal register. ON if the Function starts successfully, OFF if fails.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified to **Result**, consecutive 2 bits are used as the notify bits, and when Word data is specified, the each lowest bit of consecutive 2 words is used as the notify bit data.

●Description of Operation

Reads the data specified in **File** (\FR\FR (3 digits for File No.) .FRB) saved in **Drive** and stores it in File register of **Bank**.
The write completion notification and abnormal end alarm notification are stored in **Result**.
Turns Return value on if FRLDM Function starts normally, off if fails.

Reference

- File name to be read from Memory card or CPU memory is created as "\FR\FR (3 digits for File No.) .FRU".
  (Example) If 10 is specified for **File**, "\ FR \ FR010.FRU" will be read from Memory card or CPU memory.
- The notification bit of Read result storage target specified by **Result** has the following meanings.
  (Example) When MR2000 is specified

Notification bit		ON (1, TRUE)	OFF (1, FALSE)
MR2000	Completion notification	Completed	Uncompleted
MR2001	Abnormal notification	Abnormal completion	Normal completion

●Expression sample

bFlag := **FRLDM** (INT#1, INT#10, INT#2, abFlag[0]);

Unit Program Function

UPSTRT Starts Unit Program

Unit program starting
UPSTRT (EN, Unit, Detail, Slot, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. or its Storage destination. (1 to 48)	.U	.U	.U	.U	—	—	—	—	○
Detail	Specifies the leading to store Unit program details that requests execution.	.U	.U	.U	.U	—	—	—	—	—
Slot	Specifies the storage destination of started slot No.	.U	.U	.U	.U	—	—	—	—	—
Result	Specifies the leading to store the result.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.

●Description of Operation

At the rising edge of **EN**, starts the Unit program specified by **Detail** of the **Unit** No. unit according to the Parameter specified by **Detail**+(2).
If the 0th bit of **Detail**+(3) is OFF, uses the automatically assigned empty slot.
If the 0th bit of **Detail**+(3) is ON, uses the slot specified by **Detail**+(1).
When the Function execution is completed, stores the used slot No. in **Slot** and Execution result in **Result**.

Detail ~ : Request details

Word offset			Details
Detail	Program No.		Specifies Unit program No. to start.
Detail＋(1)	Slot No.		If the 0th bit of **Detail**+(3) is ON, specifies the slot No to use.
Detail＋(2)	Parameter		Specifies Unit program parameter to start. For Flow: specifies the first block No. to execute.
Detail＋(3)	0 bit	Slot specify method	ON ：Uses the slot specified by**Detail**+(1). OFF ：Use an empty slot automatically assigned.

Result ~ : Execution result

Word offset			Details
Result	0 bit	Completion bit	OFF when executing Function, ON when completed. Always ON regardless of whether Function ends normally or abnormally.
	1 bit	Execution failure bit	If Function execution fails, turns on simultaneously with the completion bit. Turns OFF when executing Function.
Result＋(1)	Completion code		Stores the completion code when Function is complete. 0: Started normally. 1: The specified Unit program does not exist. 2: No empty slot. 3: Operation enable is OFF. 4: Unit program error occurred. 5: The specified slot does not exist. 6: The specified slot is in use. 7: The Unit program is already being executed. 8: The specified block does not exist. 9: The Unit program is already being executed. 10: The specified Unit program is not Flow. 100: An error has occurred on the unit. 1000: The specified Unit program does not exist. 1001: The specified unit is invalid.

If Unit Program Start Function is executed at the rising edge of Execution condition, the process will continue even if the input condition is turned off at the next scan.
The completion bit and execution failure bit of the result storage destination are turned on when completed and turned off while executing.
For details, refer to "Instruction Reference Manual" and "Unit Program Instructions".

●Expression sample

UPSTRT (bEnable,uiUnitNo,auiValue[0],uiValue, auiFlag[0]);

UPSTOP Force-stops Unit Program

Unit program force-stopping
UPSTOP (EN, Unit, Slot, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. or its Storage destination. (1 to 48)	.U	.U	.U	.U	—	—	—	—	○
Slot	Specifies the Storage destination of the slot No. to be forcibly terminated.	.U	.U	.U	.U	—	—	—	—	○
Result	Specifies the leading to store the result.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.

●Description of Operation

At the rising edge of **EN**, force-stops the Unit program specified by **Slot** of the **Unit** No. unit, and empties the slot.
When the Function execution is completed, stores Execution result in **Result**.

Reference 「UPSTRT function」(page 6-15)

For details, refer to "Instruction Reference Manual" and "Unit Program Instructions".

Result ~ : Execution result storage destination

Word offset			Details
Result	0 bit	Completion bit	OFF when executing Function, ON when completed. Always ON regardless of whether Function ends normally or abnormally.
	1 bit	Execution failure bit	If Function execution fails, turns on simultaneously with the completion bit. Turns OFF when executing Function.
Result＋(1)	Completion code		Stores the completion code when Function is complete. 0: Started normally. 1: The specified Unit program does not exist. 2: No empty slot. 3: Operation enable is OFF. 4: Unit program error occurred. 5: The specified slot does not exist. 6: The specified slot is in use. 7: The Unit program is already being executed. 8: The specified block does not exist. 9: The Unit program is already being executed. 10: The specified Unit program is not Flow. 100: An error has occurred on the unit. 1000: The specified Unit program does not exist. 1001 : The specified unit is invalid.

●Expression sample

UPSTOP (bEnable, uiUnitNo, INT#1, auResult);

UFSUS Suspends Flow

Flow suspending

UFSUS (EN, Unit, Slot, Detail, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. or its Storage destination. (1 to 48)	.U	.U	.U	.U	—	—	—	—	○
Slot	Specifies the Storage destination of the slot No. to be suspended.	.U	.U	.U	.U	—	—	—	—	○
Detail	Specifies the target to store the suspend method.	.U	.U	.U	.U	—	—	—	—	○
Result	Specifies the leading to store the result.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.

●Description of Operation

At the rising edge of **EN**, suspends the Flow running in the slot specified by **Slot** of the **Unit** No. unit according to the method specified by **Detail**.

When the Function execution is completed, stores Execution result in **Result**.

Result ~: Execution result storage destination

Word offset			Details
Detail	0 bit	Completion bit	OFF when executing Function, ON when completed. Always ON regardless of whether Function ends normally or abnormally.
	1 bit	Execution failure bit	If Function execution fails, turns on simultaneously with the completion bit. Turns OFF when executing Function.
Result＋(1)	Completion code		Stores the completion code when Function is complete. 0: Started normally. 1: The specified Unit program does not exist. 2: No empty slot. 3: Operation enable is OFF. 4: Unit program error occurred. 5: The specified slot does not exist. 6: The specified slot is in use. 7: The Unit program is already being executed. 8: The specified block does not exist. 9: The Unit program is already being executed. 10: The specified Unit program is not Flow. 100: An error has occurred on the unit. 1000: The specified Unit program does not exist. 1001 : The specified unit is invalid.

Detail: Suspend method

Word offset			Details
Detail	0 bit	Suspend method	ON : Forcibly terminates the Block in progress, and suspends the Flow. OFF : Suspends the Flow when the running Block is completed.

For details, refer to "Instruction Reference Manual" and "Unit Program Instructions".

●Expression sample

UFSUS (bEnable, uiUnitNo, uiValue1,uiValue2, auiValue[0]);

UFRSM Resumes Flow

Flow resuming
UFRSM (EN, Unit, Slot, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. or its Storage destination. (1 to 48)	.U	.U	.U	.U	—	—	—	—	○
Slot	Specifies the storage destination of the slot No. to be resumed.	.U	.U	.U	.U	—	—	—	—	○
Result	Specifies the leading to store the result.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.

●Description of Operation

At the rising edge of **EN**, resumes the suspended Flow in the slot specified by **Slot** of the **Unit** No. unit.
When the Instruction execution is completed, stores Execution result in **Result**.

Reference 「UFSUS function」(page 6-17)

For details, refer to "Instruction Reference Manual" and "Unit Program Instructions".

Result ~ : Execution result storage destination

Word offset			Details
Result	0 bit	Completion bit	OFF when executing Function, ON when completed. Always ON regardless of whether Function ends normally or abnormally.
	1 bit	Execution failure bit	If Function execution fails, turns on simultaneously with the completion bit. Turns OFF when executing Function.

Word offset		Details
Result＋(1)	Completion code	Stores the completion code when Function is complete. 0: Started normally. 1: The specified Unit program does not exist. 2: No empty slot. 3: Operation enable is OFF. 4: Unit program error occurred. 5: The specified slot does not exist. 6: The specified slot is in use. 7: The Unit program is already being executed. 8: The specified block does not exist. 9: The Unit program is already being executed. 10: The specified Unit program is not Flow. 100: An error has occurred on the unit. 1000: The specified Unit program does not exist. 1001 : The specified unit is invalid.

●Expression sample

UFRSM (bEnable, uiUnitNo, uiValue, auiValue[0]);

UMALLOC Allocates Buffer Memory

Buffer memory allocating
UMALLOC (EN, Unit, Count, Result, Start);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. or its Storage destination. (1 to 48)	.U	.U	.U	.U	—	—	—	—	○
Count	Specifies the word amount storage destination to allocate.	.U	.U	.U	.U	—	—	—	—	○
Result	Specifies the leading to store the result.	.U	.U	.U	.U	—	—	—	—	○
Start	Specifies Storage destination of the leading address in allocated area.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.

●Description of Operation

At the rising edge of **EN**, allocates the word amount area specified by **Count** from Buffer memory of the **Unit** No. unit. Stores the leading address of the allocated area in **Start**, and Execution result in **Result**.
If the allocation fails, 0 is stored in **Start**.

Reference
- Area is automatically allocated.
- The value of the allocated area can be freely read/written .
- When Operation enable relay of the unit is OFF, the allocated area is deallocated.

Result ~ : Execution result

Word offset			Details
Result	0 bit	Completion bit	OFF when executing Function, ON when completed. Always ON regardless of whether Function ends normally or abnormally.
	1 bit	Execution failure bit	If the area cannot be allocated, turns on simultaneously with the completion bit. Turns OFF when executing Function.

For details, refer to "Instruction Reference Manual" and "Unit Program Instructions".

●Expression sample

UMALLOC (bEnable, uiUnitNo, uiSize, uiResult, uiAddress);

Buffer Memory Function

UREAD Reads Buffer Memory

Buffer memory reading
UREAD ([EN], Unit, Src, Dst, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	–	–	–	–	–	–	.B	–	–
Unit*1	Specifies Unit No. of Expansion special unit/CPU unit to read Buffer memory data. (0 to 48)	.U	.U	.U	.U	–	–	–	–	○
Src	Specifies the leading address of Buffer memory to read the data. (0 to 59999)	.U	.U	.U	.U	–	–	–	–	○
Dst*2	Specifies the leading to store the read the data.	.U	.S	.D	.L	.F	–	–	.T	–
Count	Specifies the data amount to be read from Buffer memory. (1 to 60000)	.U	.U	.U	.U	–	–	–	–	○
R	None	–	–	–	–	–	–	–	–	–

*1 Cannot be specified by 16# (HEX).
*2 Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, from the address specified by **Src** of the Buffer memory mounted on **Unit** No. Expansion special unit/CPU unit connected to CPU unit, reads the data amount (16-bit word data) specified by **Count**, and stores it in **Dst**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Buffer Memory Instruction".

●Expression sample

UREAD (bEnable, uiUnitNo, uiAddress, auiResult, uiCount);

UWRIT Writes Buffer Memory

Buffer memory writing
UWRIT ([EN], Unit, Dst, Src, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	–	–	–	–	–	–	.B	–	–
Unit*1	Specifies Unit No. of Expansion special unit/CPU unit to write to Buffer memory. (0 to 48)	.U	.U	.U	.U	–	–	–	–	○
Dst*2	Specifies the leading address of Buffer memory to write the data. (0 to 59999)	.U	.U	.U	.U	–	–	–	–	○
Src	Specifies the leading of the write data. String operation result can be also specified. Operations (DM0 + DM2, etc.) other than String operation cannot be written.	.U	.S	.D	.L	.F	–	–	.T	○
Count	Specifies the data amount to be written to Buffer memory. (1 to 60000)	.U	.U	.U	.U	–	–	–	–	○
R	None	–	–	–	–	–	–	–	–	–

*1 Cannot be specified by 16# (HEX).
*2 Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, writes the data amount specified by **Count** and stored in **Src**, to the address specified by **Dst** of the Buffer memory mounted on **Unit** No. Expansion special unit/CPU unit connected to CPU unit.

Reference
- If a Constant is specified in **Src**, the UWRIT Function behaves the same as the UFILL Function.
  「UFILL function」(page 6-22)

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Buffer Memory Instruction".

●Expression sample

UWRIT (bEnable, uiUnitNo, uiAddress, auiData[0], uiCount);

UFILL Fills Same Value to Buffer Memory Data String

Buffer Memory filling
UFILL ([EN], Unit, Dst, In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. of Expansion special unit/CPU unit to write to Buffer memory. (0 to 48)	.U	.U	.U	.U	—	—	—	—	○
Dst	Specifies the leading address of Buffer memory to write the data. (0 to 59999)	.U	.U	.U	.U	—	—	—	—	○
In	Specifies the data to be written, or the data storage destination.	.U	.S	.D	.L	.F	—	—	.T	○
Count	Specifies the data amount to be written to Buffer memory. (1 to 60000)	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 Cannot be specified by 16# (HEX).
*2 Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, from the address specified by **Dst** of the Buffer memory mounted on **Unit** No. Expansion special unit/CPU unit connected to CPU unit, batch-writes the data stored in **In** to the areas specified by **Count**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Buffer Memory Instruction".

●Expression sample

UFILL (bEnable, uiUnitNo, uiAddress, uiValue, uiCount);

Unit Type Identification Function

IS_□□□ Identifies Unit Type

Unit type identifying
IS_□□□ ([EN], Unit, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. or its Storage destination. (1 to 48)	.U	.U	.U	.U	—	—	—	—	○
Result*2	Specifies the storage destination of the result notification.	.B	.B	.B	.B	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 Cannot be specified by 16# (HEX).

*2 When Word data is specified, the lowest bit is used as the notification bit data. No change except for the lowest bit.

●Description of Operation

At the rising edge of Execution condition, checks the unit type of the **Unit** No. unit, and turns on **Result** when it matches the unit type corresponding to the Instruction. Turns **Result** off when mismatching or disconnected.

Instruction and Unit type

Replace "IS_" in the instruction with "KV-" to find the corresponding unit type.

Example)

(Instruction) (Unit type)

IS_XH16ML → KV-XH16ML

●Expression sample

IS_XH16ML (bEnable, INT#1, uiFlag);

Array Element Amount Get Function

ACNT Gets Array Element Amount

Array element amount getting
[R:=] ACNT (In, Dim);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies Array type variable to get the number of elements.* 1	—	—	—	—	—	—	—	—	—
Dim	Specifies the number of dimensions.	.U	—	—	—	—	—	—	—	○
R	ACNT Function has .D type identification*2	.U	.S	.D	.L	.F	.DF*3	—	—	—

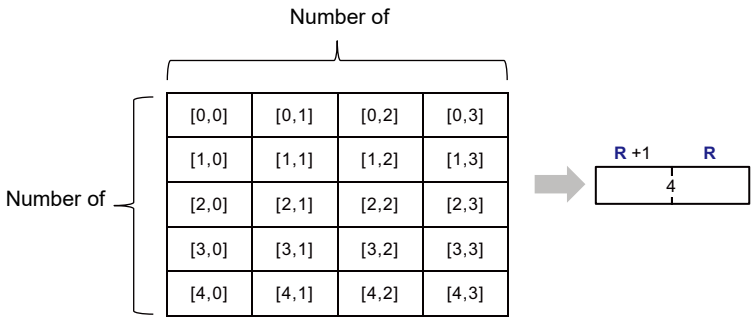
*1 Variables other than Array type cannot be specified. Also, Variable-length array cannot be specified.
*2 If a type other than .D is specified for Return value, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Counts the number of elements in the **Dim** dimension of **In**, and stores the result in **R** as 32-bit BIN data.

Example

When **In**=Array_A(2-dimensional array of 5×4), and **Dim**=2



「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Array Element Amount Get".

Point

ACNT instruction gets the number of elements, but not Word size. For example, if UDINT (unsigned 2-word integer) specifies 4 dimensions, then 4 is stored in R.

●Expression sample

udiSizeOfDimension := **ACNT** (auiData, INT#2);

Word Size Get Function

WSIZE Gets Word Size

Word size getting
[R:=] WSIZE (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies a Variable to get Word size.*1	—	—	—	—	—	—	—	—	—
R*2	Return value. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	—	—	—

*1 The following Variables cannot be specified.
- ·TIMER variable
- ·COUNTER variable
- ·Function block variable
- ·Variable-length array
- ·Variable-length string

Other Arrays can be specified.

*2 If a type other than .D is specified for Return value, type conversion is automatically performed during conversion.

*3 If Device Z is specified, the ".DF" suffix cannot be specified.

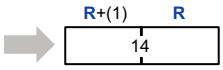
●Description of Operation

Stores the **In** word size in **R** as 32-bit BIN data.

Example

In= When the following structure type variables are specified

Member name	Data type
member1	UINT
member2	STRING[20]
member3	REAL



→The number of words in STRING[20] is 11 after (20+1)/2 is rounded up, so 1 + 11 + 2 = 14.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Word Size Get".

Point

- **WSIZE instruction gets Word size. For example, when an Array type variable or Structure type variable with 4 UDINT type data (unsigned 2-word integer) is specified, 8 is stored in R as 2 words × 4 = 8 words. It does not store the number of elements or the number of members.**
- **The Word size including Offset is stored in R. For details on Offset, refer to the User's Manual of the used CPU unit.**
- **The result rounded to the Word size is stored in R. For example, if BOOL [33], 3 is stored.**

●Expression sample

udiSizeOfTag := **WSIZE** (tagAxis_A);

7

Arithmetic Functions

Data Transfer Function 7-2
Arithmetic Function 7-6
Logical Operation Function 7-9
Data Shift Function 7-14
Data Control Function 7-22
Data Convert Function 7-29
Floating Point Function 7-44
Character Process Function 7-59

Data Transfer Function

BMOV Batch-transfers Word Block Data

Block moving
BMOV ([EN], Src, Dst, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1	Specifies the leading of Source data block.	.U	.U	.D	.D	.D	—	—	—	—
Dst*1	Specifies the leading of Destination data block.	Types same as Src.					—	—	—	—
Count	Specifies the Transfer data amount, or its Save destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 If Device T/C are specified, Transfer destination/source is Current value.

*2 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, block-transfers the Word block data amount of **Count** x (data type words) starting with **Src**, to the Word block amount of **Count** x (number of words by data type) starting with **Dst**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Transfer Instructions".

●Expression sample

BMOV (bEnable, auiSendSrc[0], auiSendDst[0], INT#10);

Point **If Count=0, Function will not be executed.**

Reference
- Source/Destination block (Bit) will be transferred correctly, even if they overlap across channels.

FMOV Fills Same Value to Data Column

Fill moving
FMOV ([EN], Src, Dst, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1	Specifies the Transfer source data, or its Storage destination.	.U	.U	.D	.D	.D	.DF*2	—	—	○
Dst*1	Specifies the leading of Destination data block.	Types same as Src.					—	—	—	—
Count	Specifies the Transfer data amount, or its Save destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 If Device T/C are specified, Transfer destination/source is Current value.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

When Execution condition is TRUE, block-transfers the data stored in **Src**, to the Word block amount of **Count** x (number of words by data type) starting with **Dst**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Transfer Instructions".

●Expression sample

FMOV (bEnable, uiSendSrc, auiSendDst[0], INT#10);

Point **If Count=0, Function will not be executed.**

Reference
- Source/Destination block (Bit) will be transferred correctly, even if they overlap across channels.

PMOV Moves Continuous Bit Info to Specified Position

Partial moving
PMOV ([EN], Src, SrcPos, Dst, DstPos, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1*2	Specifies the Transfer source data storage destination.	.U	.U	.D	.D	.D	—	—	—	○
SrcPos*3*4	Specifies the Transfer source leading bit position, or its storage destination.	.U	.U	.U	.U	—	—	—	—	○
Dst*1*2	Specifies the Transfer destination.	Types same as Src.					—	—	—	—
DstPos*3*4	Specifies the Transfer source leading bit position, or its storage destination.	.U	.U	.U	.U	—	—	—	—	○
Count	Specifies the Transfer data amount, or its storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, occupies consecutive 16 bits (if .U)/32 bits (if .D).
*2 The same target can be specified by **Src** or **Dst**.
*3 For example, if the suffix is .U, DM1000 is specified to **SrcPos**(**DstPos**) and the value is 20 (binary: 0000 0000 0001 0100), the value of the lower 4 bits is 4 (0100), so 4 is used as the value of **SrcPos**(**DstPos**).
*4 Specifies 0 to 15 if the **Src** transfer source is 16 bits (.U/.S), and 0 to 31 if 32 bits (.D/.L).

●Description of Operation

When Execution condition is TRUE, Block (Bit)-transfers Bits (for the number of Bits specified by **Count**) after the position specified by **SrcPos** of 16-bit/32-bit data stored in **Src**, to Bits after the Bit position specified by **DstPos** of **Dst**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Transfer Instructions".

Reference The original value is retained for Bits that are not transferred (can not be transferred).

●Expression sample

PMOV (bEnable, uiSendData, uiBitPos, uiRcvData, INT#3, INT#10);

Point

The Function will not be executed if any of the following conditions are TURE:

- **In the case of Src suffix is .U and Count = 0, or 16 <SrcPos + Count, or 16 <DstPos + Count.**
- **In the case of Src suffix is .U and Count = 0, or 32 <SrcPos + Count, or 32 <DstPos + Count.**

BYBMOV Transfers Continuous Byte String Info to Specified Position (H→L)

BYBMOV
BYBMOV ([EN], Src, SrcOffset, Dst, DstOffset, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1	Specifies the leading of Transfer source byte string data.	.U	.S	.D	.L	.F	.DF*2	.B	.T	—
SrcOffset	Specifies the position from Transfer source byte string leading, or its Storage destination. (0 to 65535)。	.U	.U	.U	.U	—	—	—	—	○
Dst*1	Specifies the leading of Transfer destination byte string.	.U	.S	.D	.L	.F	.DF*2	.B	.T	—
DstOffset	Specifies the position from Transfer source byte string leading, or its Storage destination. (0 to 65535)	.U	.U	.U	.U	—	—	—	—	○
Count	Specifies the Transfer data amount, or its storage destination. (1 to 65535)	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 For Bit data, specify the leading of the channel.
•The same target can be specified by Src or SrcOffset.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

When Execution condition is TRUE, transfers Bits for Count from the SrcOffset byte (counting from upper byte) of Src, to (DstOffset+Count－(1)) byte from the DstOffset byte (counting from upper byte) of Dst. The original value is retained for bytes that are not transferred.

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Data Transfer Instructions".

●Expression sample

BYBMOV (bEnable, auiData[0], uiOffset1,auiDstData[0],uiOffset2,uiCount);

Point

If Count=0, Function will not be executed.

BYLMOV Transfers Continuous Byte String Info to Specified Position (L→H)

BYLMOV
BYLMOV ([EN], Src, SrcOffset, Dst, DstOffset, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1	Specifies the leading of Transfer source byte string data.	.U	.S	.D	.L	.F	.DF*2	.B	.T	—
SrcOffset	Specifies the position from Transfer source byte string leading, or its Storage destination.(0 to 65535)	.U	.U	.U	.U	—	—	—	—	○
Dst*1	Specifies the leading of Transfer destination byte string.	.U	.S	.D	.L	.F	.DF*2	.B	.T	—
DstOffset	Specifies the position from Transfer source byte string leading, or its Storage destination.(0 to 65535)	.U	.U	.U	.U	—	—	—	—	○
Count	Specifies the Transfer data amount, or its storage destination. (1 to 65535)	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 •For Bit data, specify the leading of the channel.
•The same target can be specified by **Src** or **SrcOffset**.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

When Execution condition is TRUE, transfers Bits for **Count** from the **SrcOffset** byte (counting from lower byte) of **Src**, to (**DstOffset**+**Count**－(-1)) byte from the **DstOffset** byte (counting from lower byte) of **Dst**. The original value is retained for bytes that are not transferred.

"About the operation flags (CR2009 to CR2012)", page 1-8
For details, refer to "Instruction Reference Manual" and "Data Transfer Instructions".

●Expression sample

BYLMOV (auiSrc[0], uiSrcOffset, auiDst[0], uiDstOffset, uiCount);

Point **If Count=0, Function will not be executed.**

Arithmetic Function

INC Increments By 1

Increment Memory
INC ([EN], InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1	Specifies the target to increment by 1.	.U	.S	.D	.L	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

Increments **InOut** data by 1 and stores the result in **InOut**.

- **InOut: If the increment target is unsigned 16-bit data (.U)**
  Overflow occurs if Operation result is greater than 65535.
  (Overflow occurs if the count before operation is 65535 and the result after operation becomes 0.)
- **InOut: If the increment target is signed 16-bit data (.S)**
  Overflow occurs if Operation result is greater than +32767.
  (Overflow occurs if the count before operation is +32767 and the result after operation becomes -32768.)
- **InOut: If the increment target is unsigned 32-bit data (.D)**
  Overflow occurs if Operation result is greater than 4294967295.
  (Overflow occurs if the count before operation is 4294967295 and the result after operation becomes 0.)
- **InOut: If the increment target is signed 32-bit data (.L)**
  Overflow occurs if Operation result is greater than +2147483647.
  (Overflow occurs if the count before operation is +2147483647 and the result after operation becomes -2147483648.)

Point **Even if an overflow occurs, it will be calculated correctly.**

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Arithmetic/Comparison Instructions".

●Expression sample

INC (bEnable, uiData);

DEC Decrements By 1

Decrement Memory
DEC ([EN], InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1	Specifies the target to decrement by 1.	.U	.S	.D	.L	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

Decrements **InOut** data by 1 and stores the result in **InOut**.

- **InOut: If the decrement target is unsigned 16-bit data (.U)**
  Underflow occurs if Operation result is less than 0.
  (Underflow occurs if the count before operation is 0 and the result after operation becomes 65535.)
- **InOut: If the decrement target is signed 16-bit data (.S)**
  Underflow occurs if Operation result is less than -32768.
  (Underflow occurs if the count before operation is -32768 and the result after operation becomes +32767.)
- **InOut: If the decrement target is unsigned 32-bit data (.U)**
  Underflow occurs if Operation result is less than 0.
  (Underflow occurs if the count before operation is 0 and the result after operation becomes 4294967295.)
- **InOut: If the decrement target is signed 32-bit data (.S)**
  Underflow occurs if Operation result is less than -2147483648.
  (Underflow occurs if the count before operation is -2147483648 and the result after operation becomes +2147483647.)

Point **Even if underflow occurs, it will be calculated correctly.**

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Arithmetic/Comparison Instructions".

●Expression sample

DEC (bEnable, uiData);

ROOT Square Root

ROOT
[R:=] ROOT (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.U	.D	.L	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Calculates the square root of the data stored in **In**, and stores the result in **R**.

●Expression sample

udiResult := **ROOT** (diXCoord ** 2+ diYCoord ** 2);

Calculates the square root of the sum for diXCoord square and diYCoord square, and stores the result in udiResult.

Reference

When an integer value is specified for Operation target data:

- The square root of 16-bit/32-bit binary conversion target data are respectively calculated as the expanded 32-bit/64-bit binary result data, and the results are respectively stored again as 16-bit/32-bit BIN data.
- The value after decimal point of Operation result is truncated.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Transfer Instructions".

SQRT Square Root

Square root
[R:=] SQRT (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Calculates the square root of the data stored in **In**, and stores the result in **R**.

●Expression sample

rResult := **SQRT** (rXCoord ** 2+ rYCoord ** 2);

Calculates the square root of the sum for rXCoord square and rYCoord square, and stores the result in rResult.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Arithmetic/Comparison Instructions".

Logical Operation Function

ANDA Logical AND Operation

ANDA
[R:=] ANDA (In1, In2);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
In2*1	Specifies the Logical AND data, or its Storage destination	Types same as In1.				–	–	–	–	○
R*2	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Performs Logical AND operation for the data stored in In1 and BIN data stored in In2.

●Expression sample

- **In1: for 16-bit (.U/.S) operation target**

uiValue := **ANDA** (uiValue1, uiValue2);

- **In1: for 32-bit (.D/.L) operation target**

udiValue := **ANDA** (udiValue1, udiValue2);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Logical Operation Instructions".

●Logical AND operation truth table

Operation target	0	0	1	1
Operation data	0	1	0	1
Return value	0	0	0	1

ORA Logical OR Operation

ORA
[R:=] ORA (In1, In2);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
In2*1	Specifies the Logical OR data, or its Storage destination.	Types same as In1.				–	–	–	–	○
R*2	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 1When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

*2 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Performs Logical OR operation for the data stored in In1 and BIN data stored in In2.

●Expression sample

- **In1: for 16-bit (.U/.S) operation target**

uiValue := **ORA** (uiValue1, uiValue2);

- **In1: for 32-bit (.D/.L) operation target**

udiValue := **ORA** (udiValue1, udiValue2);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Logical Operation Instructions".

●Logical OR operation truth table

Operation target	0	0	1	1
Operation data	0	1	0	1
Return value	0	1	1	1

EORA Exclusive OR Operation

Exclusive ORA
[R:=] EORA (In1, In2);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
In2*1	Specifies the Logical EOR data, or its Storage destination.	Types same as In1.				–	–	–	–	○
R*2	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Performs Logical EOR operation for the data stored in In1 and BIN data stored in In2.

●Expression sample

- **In1: for 16-bit (.U/.S) operation target**

uiValue := **EORA** (uiValue1, uiValue2);

- **In1: for 32-bit (.D/.L) operation target**

udiValue := **EORA** (udiValue1, udiValue2);

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Logical Operation Instructions".

●Logical EOR operation truth table

Operation target	0	0	1	1
Operation data	0	1	0	1
Return value	0	1	1	0

ENRA Exclusive NOR Operation

Exclusive NORA

[R:=] ENRA (In1, In2);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
In2*1	Specifies the Logical exclusive NOR data, or its Storage destination.	Types same as In1.				–	–	–	–	○
R*2	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

*2 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.

*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Performs Logical exclusive NOR operation for the data stored in In1 and BIN data stored in In2, and stores the result in R.

●Expression sample

- **In1: for 16-bit (.U/.S) operation target**

uiValue := **ENRA** (uiValue1, uiValue2);

- **In1: for 32-bit (.D/.L) operation target**

udiValue := **ENRA** (udiValue1, udiValue2);

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Logical Operation Instructions".

●Logical exclusive NOR operation truth table

Operation target	0	0	1	1
Operation data	0	1	0	1
Return value	1	0	0	1

COM Inverts Bit

Complement
[R:=] COM (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
R*2	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Inverts each bit of BIN data stored in In.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Logical Operation Instructions".

●Expression sample

uiValue := **COM** (uiValue1);

NEG Inverts Sign

Negative
[R:=] NEG (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.S	.D	.L	.F	.DF*2	–	–	○
R*1	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Inverts the sign of BIN data stored in In (2's complement operation).

For details, refer to "Instruction Reference Manual" and "Logical Operation Instructions".

●Expression sample

uiValue := **NEG** (uiValue1);

ABS Absolute Value

Absolute value
[R:=] ABS (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.S	.D	.L	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Stores Absolute value of BIN data stored in In in R.

●Expression sample

uiValue := **ABS** (iADConvData);

Point

If the following data is specified, it will not operate properly.

- **Numbers exceeding the positive integer range of INT (-32768)**
- **Numbers exceeding the positive integer range of DINT (-2147483648)**

Example:

- **When udiVar: = ABS (iVar); // iVar: = -32768, -32768 is stored in udiVar.**
- **When lrVar: = ABS (udiVar); // udiVar: =-2147486348, -2147486348 is stored in lrVar.**

Data Shift Function

SHR Right-shifts (in bits)

Shifting to Right
[R:=] SHR (In, Num);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
Num	Specifies the Right-shift amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Shifts the data stored in In to the left by the number of bits specified by Num, and stores 0 in the number of bits Left-shifted from the lower bit.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

●Expression sample

uiValue := **SHR** (uiValue1, INT#4);

Point
- **If Num is 0, the value of In is stored in R.**
- **If In is .U (UINT) and Num ≥ 17, or In is .D (UDINT) and Num ≥ 33, the Operation result will be 0.**

SHL Left-shifts (in bits)

Shifting to Left
[R:=] SHL (In, Num);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
Num	Specifies the Left-shift amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Shifts the data stored in In to the left by the number of bits specified by Num, and stores 0 in the number of bits Left-shifted from the lower bit.

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

●Expression sample

uiValue := **SHL** (uiValue1, INT#5);

Point
- **If Num is 0, the value of In is stored in R.**
- **Operation result is 0 if In is .U (UINT) and Num ≥ 17, or In is .D (UDINT) and Num ≥ 33.**

ASRA Right-shifts Arithmetic (in bits)

Arithmetic right-shifting
[R:=] ASRA (In, Offset);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.S	.D	.L	–	–	–	–	○
Offset	Specifies the Right-shift amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Shifts the data stored in In to the right by the number of bits specified by Offset, and stores "0" (if In≧0) or "1" (if In＜0) in the number of bits Right-shifted from the upper bit.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

●Expression sample

uiValue := **ASRA** (uiValue1, INT#5);

Point
- **If Offset is 0, the value of In is stored in R.**
- **Operation result is 0 if In is .U (UINT) and Offset ≥ 17, or In is .D (UDINT) and Offset ≥ 33.**

ASLA Left-shifts Arithmetic (in bits)

Arithmetic left-shifting
[R:=] ASLA (In, Offset);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.S	.D	.L	–	–	–	–	○
Offset	Specifies the Left-shift amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Shifts the data stored in In to the left by the number of bits specified by Offset, and stores 0 in the number of bits Left-shifted from the lower bit.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

●Expression sample

uiValue := **ASLA** (uiValue1, INT#5);

Point
- **If Offset is 0, the value of In is stored in R.**
- **Operation result is 0 if In is .U (UINT) and Offset ≥ 17, or In is .D (UDINT) and Offset ≥ 33.**

RRA Right-rotates (in bits, with Carry)

Right-rotating
[R:=] RRA (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
Count	Specifies the Right-rotate amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register. The type of RRA Function is the same as the 1st Argument.*1	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Rotates the data stored in In to the right by the number of bits specified by Count including Carry flag (CR2009).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

●Expression sample

uiValue := **RRA** (uiValue1, INT#5);

Point **When rotated to the right, the lowest bit is stored in Carry flag (CR2009).**

RLA Left-rotates (in bits, with Carry)

Left-rotating
[R:=] RLA (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	–	–	–	–	○
Count	Specifies the Left-rotate amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Rotates the data stored in In to the left by the number of bits specified by Count including Carry flag (CR2009).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

●Expression sample

uiValue := **RLA** (uiValue1, INT#5);

Point **When rotated to the left, the most upper bit is stored in Carry flag (CR2009).**

ROR Right-rotates (in bits, without Carry)

Right-rotating without Carry
[R:=] ROR (In, Num);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target storage destination.	.U	.U	.D	.D	–	–	–	–	○
Num*1	Specifies the Right-rotate amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register. The type of ROR Function is the same as the 1st Argument.* 2	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 If the target is specified for **In** and the **Num** value is 16 or more (the Operation target is .U)/32 or more (the Operation target is .D), the result is as follows:
For .U: The remainder of the **Num** value divided by 16 is the number of bits to be rotated.
For .D: The remainder of the **Num** value divided by 32 is the number of bits to be rotated.
*2 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Rotates the data stored in **In** to the right by the number of bits specified by **Num** without Carry, and stores the result in **R**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

Reference The state of the rotate result lowest bit is stored in Carry flag (CR2009).

●Expression sample

uiValue := **ROR** (uiValue1, INT#5);

ROL Left-rotates (in bits, without Carry)

Left-rotating without Carry
[R:=] ROL (In, Num);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target storage destination.	.U	.U	.D	.D	–	–	–	–	○
Offset[*1]	Specifies the Left-rotate amount, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R[*2]	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF[*3]	–	–	–

*1 If the target is specified for **In** and the **Num** value is 16 or more (the Operation target is .U)/32 or more (the Operation target is .D), the result is as follows:
For .U: The remainder of the **Num** value divided by 16 is the number of bits to be rotated.
For .D: The remainder of the **Num** value divided by 32 is the number of bits to be rotated.
*2 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Rotates the data stored in **In** to the left by the number of bits specified by **Num** without Carry, and stores the result in **R**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

Reference The state of the rotate result most upper bit is stored in Carry flag (CR2009).

●Expression sample

uiValue := **ROL** (uiValue1, INT#5);

WSR Right-shifts Word Device

Right-shifting word
WSR ([EN], InOut, Count, Offset);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	.D	—	—	—	—
Count*1	Specifies the Right-shift range data amount, and its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Offset	Specifies the Right-shift amount, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, starting with **InOut**, shifts the data (16 bits/32 bits) stored in the data amount range specified by **Count**, to the right side (the side with larger Device No./Array subscript) for the amount specified by **Offset**.
0 is stored in the area of the shift source range that does not overlap with the shift destination range.

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

Reference
If Operation target is 32-bit data, the Operation range data amount and the Shift amount are the specified data amount x 2 words.

●Expression sample

WSR (bFlag, uiValue, INT#32, INT#10);

Point
- **If Count=0, Function will not be executed.**
- **If Offset＝0, Operation result is the same as before operation.**

WSL Left-shifts Word Device

Left-shifting Word
WSL ([EN], InOut, Count, Offset);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1	Specifies the leading of Left-shift range.	.U	.U	.D	.D	.D	—	—	—	—
Count*1	Specifies the Left-shift range data amount, and its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Offset	Specifies the Left-shift amount, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 1When Bit data is specified, operation is performed with consecutive 16/32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, starting with **InOut**, shifts the data (16 bits/32 bits) stored in the data amount range specified by **Count**, to the left side (the side with smaller Device No./Array subscript) for the amount specified by **Offset**.
0 is stored in the area of the shift source range that does not overlap with the shift destination range.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

Reference
If Operation target is 32-bit data, the Operation range data amount and the Shift amount are the specified data amount x 2 words.

●Expression sample

WSL (bFlag, uiValue, INT#32, INT#10);

Point
- **If Count=0, Function will not be executed.**
- **If Offset＝0, Operation result is the same as before operation.**

BSR Right-shifts Bit Device

Right-shifting Bit
BSR ([EN], InOut, Count, Offset);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1*2	Specifies the leading of Right-shift range.	—	—	—	—	—	—	.B	—	—
Count	Specifies the Right-shift range target amount, and its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Offset	Specifies the Right-shift amount, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 If Bit data is specified, operates correctly across channels.
If Word data is specified, the Shift target is the number of Word data lowest bits specified by **Count**. No change except for the lowest bit.
*2 T/C/CR/Z/CM cannot be used.

●Description of Operation

When Execution condition is TRUE, starting with **InOut**, shifts the data amount specified by **Count**, to the right side (the side with larger Device No./Array subscript) for the amount specified by **Offset**. If **InOut** is Word data, only the lowest bit of each Word data is the Shift target. (No change except for the lowest bit)
OFF is stored in the position (the lowest bit in the case of Word data) where Shift source range does not overlap with Shift destination range.

●Expression sample

BSR (bFlag, abFlag[0], INT#3, INT#4);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

Point

- **If Offset＝0, Operation result is the same as before operation.**
- **The Function will not be executed if any of the following conditions are TURE:**
  - **When Shift destination range exceeds Device/Variable setting range**
  - **When Count is 0.**

BSL Left-shifts Bit Device

Left-shifting Bit

BSL ([EN], InOut, Count, Offset);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
InOut*1*2	Specifies the leading of Left-shift range.	—	—	—	—	—	—	.B	—	—
Count	Specifies the Left-shift range target amount, and its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Offset	Specifies the Left-shift amount, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 If Bit data is specified, operates correctly across channels.
If Word data is specified, the Shift target is the number of Word data lowest bits specified by **Count**. No change except for the lowest bit.

*2 T/C/CR/Z/CM cannot be used.

●Description of Operation

When Execution condition is TRUE, starting with **InOut**, shifts the data amount specified by **Count**, to the left side (the side with smaller Device No./Array subscript) for the amount specified by **Offset**. If **InOut** is Word data, only the lowest bit of each Word data is the Shift target. (No change except for the lowest bit)
OFF is stored in the position (the lowest bit in the case of Word data) where Shift source range does not overlap with Shift destination range.
0 is stored in the area of the shift source range that does not overlap with the shift destination range.

●Expression sample

BSL (bFlag, abFlag[0], INT#3, INT#4);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Shift Instructions".

Point

- **If Offset＝0, Operation result is the same as before operation.**
- **The Function will not be executed if any of the following conditions are TURE:**
  - **If InOut－Count＞0.**
  - **When Count is 0.**

Data Control Function

LIMIT Upper/Lower Limit Control

Limit
[R:=] LIMIT (MN, In, MX);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
MN	Specifies the Lower limit value.	.U	.S	.D	.L	.F	.DF*2	—	—	○
In	Specifies the Control target.	.U	.S	.U	.L	.F	.DF*2	—	—	○
MX	Specifies the Upper limit value.	.U	.S	.U	.L	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Limits In by MN/MX and stores the result in R.

In value	R value
In < MN	MN
MN ≦ In ≦ MX	In
MX < In	MX

●Expression sample

rValue := **LIMIT** (rMinValue, rInput, rMaxValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Control Instructions".

BANDC Dead Zone Control

Dead zone control
[R:=] BANDC (LoLim, HiLim, In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
LoLim*1	Specifies the Lower limit value, or its Storage destination.	.S	.S	.L	.L	—	—	—	—	○
HiLim*1	Specifies the Upper limit value, or its Storage destination.	Types same as LoLim.				—	—	—	—	○
In*1	Specifies the Dead zone control input value, or its Storage destination.	Types same as LoLim.				—	—	—	—	○
R*2	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Limits **In** by **HiLim**/**LoLim** and stores the result in **R**.

In value	R value
In < LoLim	LoLim
LoLim ≦ In ≦ HiLim	In
HiLim < In	HiLim

●Expression sample

uiValue := **BANDC** (uiValue1, uiValue2, uiValue3);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Control Instructions".

ZONE Zone Control

[R:=] ZONE (NegBias, PosBias, In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
NegBias*1	Specifies the Negative bias value that added to Input value, or its Storage destination.	.S	.S	.L	.L	–	–	–	–	○
PosBias*1	Specifies the Positive bias value that added to Input value, or its Storage destination.	Types same as NegBias.				–	–	–	–	○
In*1	Specifies the Zone control input value, or its Storage destination.	Types same as NegBias.				–	–	–	–	○
R	Return value. Omissible. If omitted, it is stored in Internal register. The type of RZONE Function is the same as the 1st Argument.*2	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Limits **In** by **PosBias**/**NegBias** and stores the result in **R**.

In value	R value
In < NegBias	NegBias
NegBias ≦ In ≦ PosBias	In
PosBias < In	PosBias

●Expression sample

uiValue := **ZONE** (uiValue1, uiValue2, uiValue3);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Control Instructions".

APR Broken Line Approximation (Scaling)

Broken line approximation

[R:=] APR (Src, Table, In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Src*1	Specifies the Broken line approximation input value, or its Storage destination.	.U	.S	.D	.L	.F	.DF*4	–	–	○
Table*2	Specifies the leading that stores Broken line data table.	Types same as Src.						–	–	–
In	Specifies the number of Broken line data records, or its Storage destination.	.U	.U	.U	.U	–	–	–	–	○
R*3	Return value. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*4	–	–	–

*1 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
• When T and C are specified, Current value is used for operation.

*2 If the Src type is .F or .DF, T and C cannot be specified.

*3 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.

*4 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

- **Src: for 16-bit (.U/.S) input value**
  For the "Input value" of the Broken line approximation stored in Src, operates it according to the "Line chart data table" that stores records of In starting with Table.
  Broken line approximation operation searches the Src position in Line chart data table, and performs Broken line approximation by the two points before and after. Operation result is stored in R.
  Set the Line chart data table X value in ascending order. ($X_0 < X_1 \ldots < X_k \ldots < X_n$)
  Store the Line chart data table in the order of (X_0, Y_0), (X_1, Y_1)…(X_{n-1}, Y_{n-1}), (X_n, Y_n) from Table.
  Store the number of Line chart data table records in In. In addition, set In to a value of 2 or greater.

  If Src (Input value) <X_0, Y_0 is stored in R.
  If Src (Input value) >X_n, Y_n is stored in R.
  If Src (Input value) =X_k, Y_k is stored in R.

- **Src: for 32-bit (.D/.L) input value**
  For the "Input value" of Broken line approximation stored in [Src/Src + (1)], operates it according to the "Line chart data table" that stores records of [In] starting with [Table/Table + (1)].
  Broken line approximation operation searches the [Src/Src+(1)] position in Line chart data table, and performs Broken line approximation by the two points before and after.
  Operation result is stored in [R/R+(1)].
  Store the Line chart data table in the order of (X_0, Y_0), (X_1, Y_1)…(X_{n-1}, Y_{n-1}), (X_n, Y_n) from [Table/Table+(1)].
  Store the number of records in the Line chart data table in [In] (number of records).
  In addition, set the number of [In] to 2 or more.

  [If [Src/Src + (1)] <X_0, Y_0 is stored in [R/R + (1)].
  If [Src/Src + (1)]> X_n, Y_n is stored in [R/R + (1)].
  If [Src/Src + (1)] = X_k, Y_k is stored in [R/R + (1)] (k = 1 to n).

- **Src: for 32-bit input value (.F)**
  For the "Input value" of Broken line approximation stored in [Src/Src + (1)], performs approximate calculation (Single precision floating point type) according to the "Line chart data table" that stores records of [In] starting with [Table/Table + (1)].
  Broken line approximation operation searches the [Src/Src+(1)] position in Line chart data table, and performs Broken line approximation by the two points before and after.
  Operation result is stored in [R/R+(1)].

Store the Line chart data table in the order of (X_0, Y_0), (X_1, Y_1)…(X_{n-1}, Y_{n-1}), (X_n, Y_n) from [Table/Table+(1)].

Set the Line chart data table X value in ascending order. ($X_0 < X_1 \ldots < X_k \ldots < X_n$)
Store the number of Line chart data table records in [In].
In addition, set the number of [In] to 2 or more.

[If [Src/Src + (1)] <X_0, Y_0 is stored in [R/R + (1)].
If [Src/Src + (1)]> X_n, Y_n is stored in [R/R + (1)].
If [Src/Src + (1)] = X_k, Y_k is stored in [R/R + (1)] (k = 1 to n).

- **Src: for 64-bit input value (.F)**
  For the "Input value" of Broken line approximation stored in [Src/Src+(1)/Src+(2)/Src+(3)], performs approximate calculation (Double precision floating point type) according to the "Line chart data table" that stores records of [In] starting with [Table+(1)/Table+(2)/Table+(3)].

  Broken line approximation operation searches the [Src/Src+(1)/Src+(2)/Src+(3)] position in Line chart data table, and performs Broken line approximation by the two points before and after.
  Operation result is stored in [R/R+(1)/R+(2)/R+(3)].
  Store the Line chart data table in the order of (X_0, Y_0), (X_1, Y_1)…(X_{n-1}, Y_{n-1}), (X_n, Y_n) from [Table/Table+(1)/Table+(2)/Table+(3)].
  Set the Line chart data table X value in ascending order. ($X_0 < X_1 \ldots < X_k \ldots < X_n$)

  Store the number of Line chart data table records in [In].
  In addition, set the number of [In] to 2 or more.

  If [Src/Src+(1)/Src+(2)/Src+(3)]<X_0, Y_0 is stored in [R/R+(1)/R+(2)/R+(3)].
  If [Src/Src+(1)/Src+(2)/Src+(3)]>X_n, Y_n is stored in [R/R+(1)/R+(2)/R+(3)].
  If [Src/Src+(1)/Src+(2)/Src+(3)]=X_k, Y_k is stored in [R/R+(1)/R+(2)/R+(3)](k=1 to n)。

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Control Instructions".

●Expression sample

uiValue := **APR** (uiValue1, auiData[0],uiCount);

RAMP Ramp Signal

Ramp signal
RAMP ([EN], In, Ratio, TimeUnit, InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the Input value, or its Storage destination. (－32768 to 32767)	.S	.S	.S	.S	—	—	—	—	○
Ratio*1	Specifies the Change rate in time, or its Storage destination. (0 to 65535)	.U	.U	.U	.U	—	—	—	—	○
TimeUnit*1*2	Specifies the Time unit. (0 to 2)	.U	.U	.U	.U	—	—	—	—	○
InOut*1	Stores the result after adding Ramp amount to Input value at the specified interval.	.S	.S	.S	.S	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

• T/C/Z process the lower 16 bits.

*2 Specify the time unit setting in the range of 0 to 2.

Description of time unit settings (**TimeUnit**)

Value	Time unit
0	1 second
1	1 minute
2	1 hour

●Description of Operation

When Execution condition is TRUE, approaches the **InOut** value by the specified Ramp (**Ratio**/**TimeUnit**) until it matches **In**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Control Instructions".

●Expression sample

RAMP (bFlag, uiValue1, uiValue2, INT#1, uiValue);

Point **If TimeUnit ≥ 3, the Function will not be executed.**

TPOUT Time Proportional Output

Time proportional output
TPOUT ([EN], Duty, Cycle, InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Duty *1	Specifies the Output relay ON time in ms. (0 to 65535)	.U	.U	.U	.U	—	—	—	—	○
Cycle *1	Specifies the Output relay ON/OFF cycle in ms. (1 to 65535)	.U	.U	.U	.U	—	—	—	—	○
InOut *1	Specifies the ON/OFF target.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
• T/C/Z process the lower 16 bits.

*2 • When Word data is specified, the result is stored in the lowest bit. No change except for the lowest bit.
• When a position other than the channel leading such as R1002 is specified, operates correctly across the next channel.
• Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, turns on **InOut** that specified only by **Duty** (ms) in **Cycle** (ms).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Control Instructions".

●Expression sample

TPOUT (bFlag, uiValue1, uiValue2, bControl);

Point

- **The Function will not be executed if any of the following conditions are TURE:**
  - When **Cycle**＜**Duty**
  - When **Cycle**=0
- **An error within ±10μs+1 scan time occurs.**
- **When 0 is specified for Duty, the Function does not turn on for even one scan.**
- **When performing Writing During RUN, the cycle (Cycle) is initialized and the ON time (Duty) is measured starting form the Writing During RUN ending.**
- **It cannot be used in Interrupt program, or used for Initialization module.**
- **Pay attention when using it in Subroutine programs, and between STP to STE, STG to JM/ENDS, and CJ (NCJ) to LABEL instructions.**
  Refer to "Precautions on Timer" in the user's manual of the used CPU unit.
- **Cautions are required when performing Writing During RUN.**
  Refer to "About Writing During RUN" in the user's manual of the used CPU unit.

LLFLT Lead/Lag Filter

Lead/Lag filter
LLFLT ([EN], In, Param, InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1*2	Specifies the Filter input value, or its Storage destination. (−32768 to 32767)	.S	.S	.S	.S	—	—	—	—	○
Param*1*3	Specifies the leading to store Parameter.	.U	.U	.U	.U	—	—	—	—	—
InOut*1	Store the Output value storage destination. (−32768 to 32767)	.S	.S	.S	.S	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 T/C/Z process the lower 16 bits.
*3 • When Bit data is specified, occupies consecutive 48 bits.
• For Bit data, specify the leading of the channel.
• When Word data is specified, occupies consecutive 3 bits.

●Description of Operation

When Execution condition is TRUE, executes Lead/Lag operation for each **Param** for **In**, and stores it in **InOut**.
When the Function is executed at the rising edge of Execution condition, **InOut** is initialized with the value of **In**.

Param: Parameter details

Word offset		Description
Param	Sampling cycle (T_S)	Specifies the Cycle time for executing the instruction. (1 to 60000ms) Specify a smaller value than Lead/Lag time.
Param+(1)	Lag time(T1)	Specifies the Time constant for Lag element. (0 to 65535ms) Lag is disabled if 0 is specified.
Param+(2)	Lead time(T2)	Specifies the Time constant for Lead element. (0 to 65535ms) Lead is disabled if 0 is specified.

For details on Lead/Lag operation, refer to "Instruction Reference Manual" and "LLFLT Instructions".
「About the operation flags (CR2009 to CR2012)」(page 1-8)

●Expression sample

LLFLT (bFlag, uiValue, auiData[0], iControl);

Point
- **When performing Writing During RUN, the Sampling cycle (Param+(1)) is initialized and Output value (InOut) is initialized with Input value (In).**
- **It cannot be used in Interrupt program, or used for Initialization module.**
- **Pay attention when using it in Subroutine programs, and between STP to STE, STG to JM/ENDS, and CJ (NCJ) to LABEL instructions.**
  Refer to "Precautions on Timer" in the user's manual of the used CPU unit.
- **Cautions are required when performing Writing During RUN.**
  Refer to "About Writing During RUN" in the user's manual of the used CPU unit.

Data Convert Function

TBCD Converts BIN Data to BCD Data

BCD transferring
[R:=] TBCD (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	.U	.U	.D	.D	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts 16-bit BIN data stored in In to 4-digit BCD (16-bit) data, or 32-bit BIN data to 8-digit BCD (32-bit) data, and stores it in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

uiValueBCD := **TBCD** (uiValue);

Point **When exceeding the range of 0 to 9999 for 16-bit In (or 0 to 99999999, for 32-bit), the value of In is stored in R.**

TBIN Converting BCD Data to BIN Data

BIN transferring
[R:=] TBIN (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	.U	.U	.D	.D	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts 4-digit BCD (16-bit) data stored in In to16-bit BIN data, or 8-digit BCD (32-bit) data to 32-bit BIN data, and stores it in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

uiValue := **TBIN** (uiValueBCD);

Point **When In is not BCD data, the value of In is stored in R.**

MPX Decodes 4-bit Data

Multiplex
[R:=] MPX (In, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*2*3	Specifies the Convert target or Convert target data.	.U	.U	.D	.D	–	–	–	–	–
Pos*1	Specifies the Digit convert position. (0 to 3)	–	–	–	–	–	–	–	–	○
R*4	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	–	–	–

*1 In **Pos**, specify the number of digits from 0 to 3 (4 bits/1 digit) with a constant. (No target can be specified)
*2 When 32-bit data is specified to **In**, the Convert target (the effective digit) is the lower 16 bits.
*3 Even if a Convert target other than the Bit data channel leading such as R1002 is specified, operates correctly across the next channel.
*4 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Digitizes (0 to 15) the 4 bits (0 to 3 digits) specified by **Pos** of the 16-bit data stored in **In**. Stores the result that turns on only the bits indicated by the digitized value in **R**.

For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

uiValueB := **MPX** (uiValue, INT#2);

DMX Encodes 4-bit Data

Demultiplex
[R:=] DMX (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2	Specifies the Convert target or Convert target data.	.U	.U	.U	.U	–	–	–	–	○
R*3	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*4	–	–	–

*1 When 32-bit data is specified to **In**, the Convert target (the effective digit) is the lower 16 bits.
*2 Even if a Convert target other than the Bit data channel leading such as R1002 is specified, operates correctly across the next channel.
*3 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.

*4 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Stores the most upper bit position (0 to 15) of the ON bits for the 16-bit data stored in **In**, in **R**, as the 16-bit (stored in the lower 4 bits) data.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

uiCount := **DMX** (uiValue);

Point **If the value of In is 0, 0 is stored in R.**

GRY Converts BIN Data to Gray Code

Gray code converting
[R:=] GRY (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	.U	.U	.D	.D	—	—	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts 16-bit/32-bit BIN data stored in **In** to Gray code, and stores it in **R**.
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

uiValueGRY := **GRY** (uiValue);
Gray code table

DEC	bit															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
to																
31	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
to																
255	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
256	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
to																
65535	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RGRY Converts Gray Code to BIN Data

Gray code inverse-converting
[R:=] RGRY (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	.U	.U	.D	.D	—	—	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts Gray code stored in In to 16-bit/32-bit BIN data, and stores it in R.

For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

uiValue := **RGRY** (uiValueGRY);

Gray code table

DEC	bit															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
to																
31	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
to																
255	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
256	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
to																
65535	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DISN Disperses Nibble

Nibble dispersing
DISN ([EN], In, Dst, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1*3	Specifies the Nibble disperse data storage destination.	.U	.U	.D	.D	—	—	.B	—	○
Dst*1*3	Specifies the leading to store the Nibble disperse data.	Types same as In.					—	—	—	○
Count*2	Specifies the Nibble disperse data amount.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, **In** processes with continuous 16/32 bits, and **Dst** processes with continuous 64/128 bits. When a position other than the channel leading such as R002 is specified, the operation will be processed across the next channel.
When Word data is specified, **In** processes consecutive 1 bits. **Dst** processes 4/8 word.
*2 If the suffix of the data specified by **In** is .U or .B, specify it in the range of 0 to 4, and if it is .D, 0 to 8.
*3 Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, disperses the data specified by **In** into the amount specified by **Count** in nibble (4-bit) units, and stores in **Dst**.
0 is stored in the position upper than 4-bit.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

DISN (uiValue, auiDst[0], uiCount);

Point **If Count is 0, Function will not be executed.**

UNIN Unites Nibble

Nibble uniting
UNIN ([EN], Src, InOut, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1*3	Specifies the Nibble unite data storage destination.	.U	.U	.D	.D	—	—	.B	—	—
InOut*1*3	Specifies the leading to store the Nibble unite data.	Types same as Src.					—	—	—	—
Count*2	Specifies the Nibble unite data amount.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, **Src** processes with continuous 64/128 bits, and **InOut** processes with continuous 16/32 bits. When a position other than the channel leading such as R002 is specified, the operation will be processed across the next channel.
When Word data is specified, **Src** processes consecutive 4/8 words. **InOut** processes consecutive 1/2 words.
*2 If the suffix of the data specified by **Src** is .U or .B, specify it in the range of 0 to 4, and if it is .D, 0 to 8.
*3 Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, unites the lower 4 bits of the Unite data starting with **Src**, to the BIN data by the amount specified by **Count**, and stores in **InOut**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

UNIN (auiValue[0], uiDst, uiCount);

Point **If Count is 0, Function will not be executed.**

DISB Disperses Byte

Byte dispersing
DISB ([EN], In, InOut, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1*3	Specifies the Byte disperse data storage destination.	.U	.U	.D	.D	—	—	.B	—	—
InOut*1*3	Specifies the leading to store the Byte disperse data.	Types same as In.					—	—	—	—
Count*2	Specifies the Byte disperse data amount.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, In processes with continuous 16/32 bits, and InOut processes with continuous 32/64 bits. When a position other than the channel leading such as R002 is specified, the operation will be processed across the next channel.
When Word data is specified, In processes consecutive 1 bits. InOut processes consecutive 2/4 words.
*2 If the suffix of the data specified by In is .U or .B, specify it in the range of 0 to 2, and if it is .D, 0 to 4.
*3 Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, disperses the data specified by In into the amount specified by Count in byte (8-bit) units, and stores in InOut.
0 is stored in the position upper than 8-bit.
「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

DISB (uiValue, auiDst[0], uiCount);

Point **If Count is 0, Function will not be executed.**

UNIB Unites Byte

Byte uniting
UNIB ([EN], Src, InOut, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1*3	Specifies the Byte unite data storage destination.	.U	.U	.D	.D	—	—	.B	—	—
InOut*1*3	Specifies the leading to store the Byte unite data.	Types same as Src.					—	—	—	—
Count*2	Specifies the Byte unite data amount.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 1When Bit data is specified, Src processes with continuous 32/64 bits, and InOut processes with continuous 16/32 bits. When a position other than the channel leading such as R002 is specified, the operation will be processed across the next channel.
When Word data is specified, Src processes consecutive 2/4 words. InOut processes consecutive 1/2 words.
*2 If the suffix of the data specified by Src is .U or .B, specify it in the range of 0 to 2, and if it is .D, 0 to 4.
*3 Z cannot be specified.

●Description of Operation

When Execution condition is TRUE, unites the lower 8 bits of the Unite data starting with **Src**, to the BIN data by the amount specified by **Count**, and stores in **InOut**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

UNIB (auiValue[0], uiDst, uiCount);

Point **If Count is 0, Function will not be executed.**

SWAP Swaps Upper/Lower Byte

Swapping
[R:=] SWAP (InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
InOut	Specifies the Swap target or Swap target data.	.U	.U	.D	.D	–	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Swaps values of the upper byte (upper 8 bits) and lower byte (lower 8 bits) for the 16-bit data stored in **InOut**, and stores the result in **R**.

For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

uiValue := **SWAP** (uiValue);

BSWAP Block-swaps

Block-swapping
BSWAP ([EN], InOut, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	–	–	–	–	–	–	.B	–	–
InOut	Specifies the Swap target.	.U	.U	.U	.U	–	–	.B	–	–
Count	Specifies the Swap amount.	.U	.U	.U	.U	–	–	–	–	○
R	None	–	–	–	–	–	–	–	–	–

●Description of Operation

When Execution condition is TRUE, swaps the upper byte (upper 8 bits) and lower byte (lower 8 bits) of the data amount specified by **Count** starting with **InOut**.

For details, refer to 📖 "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

BSWAP(auiValue, INT#4);

Point **If Count=0, Function will not be executed.**

DECO Decodes 8-bit

8-bit decoding
DECO ([EN], Src, Dst, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src	Specifies the leading of Convert range.	.U	.U	.U	.U	—	—	—	—	—
Dst*1*2	Specifies the Decode result storage destination.	—	—	—	—		—	.B	—	—
Count	Specifies the number of Convert ranges (number of bits: n) (1 to 8).	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 If Word data is specified, the lowest bit of continuous 2^n word data becomes the Decode result storage destination data. No change except for the lowest bit.

*2 Z cannot be used.

●Description of Operation

When Execution condition is TRUE, digitizes (0 to 255) the number of states (number of bits: n) specified by **Count** starting with the Bit data specified by **Src**. Turns on the bit of the leading No. + numerical value (No.) in the continuous 2^n bit data starting with **Dst**, and turns off all other bits.
When **Src** is a Word data, **Count** indicates the number of bits to convert to numerical values, and when the Storage destination is a Word data, turns on the lowest bit of the Word data of the leading No. + numerical value (No.) in the continuous 2^n Word data starting with the specified word data, and turns off all lowest bits of other Word data. (No change except for the lowest bit)

📖「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to 📖 "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

DECO(uiValue, abResult[0], uiCount);

Point **If Count=0, Function will not be executed.**

ENCO Encodes 8-bit

8-bit encoding
[R:=] ENCO (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2	Specifies the leading of Convert range.	—	—	—	—	—	—	.B	—	—
Count	Specifies the number of Convert ranges (number of bits: n). (1 to 256)	.U	.U	.U	.U	—	—	—	—	○
R*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*4	—	—	—

*1 When Word data is specified, the Encode target is the number of Word data lowest bits specified by **Count**.
*2 Z cannot be used.
*3 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.
*4 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

As the 16-bit data, stores the lowest bit position (0 to 255) of the ON bits in the state of the number (number of digits: n) specified by **Count** starting from the bit data specified by **In** to **R**.
When **In** is Word data, **Count** indicates the number of Words, and the lowest bit position (0 to 255, in ON state) of Word within the Convert target amount range is stored in **R** as 16-bit data.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

ENCO (abData[0], INT#8);

Point **If Count=0, Function will not be executed.**

CPMSET Creates CIP Message

CIP message creating
CPMSET ([EN], Src, Param, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1	Specifies the leading of the data to be added.	.U	.S	.D	.L	.F	.DF	.B	.T	—
Param	Specifies the Data type/Data size (in bytes) of the send data to be added, or their Storage destination.*1	.U	.U	.U	.U	—	—	—	—	○
Dst*1	Specifies the leading of the Send data to be created.	.U	.S	.D	.L	.F	.DF	.B	.T	—
R	None	—	—	—	—	—	—	—	—	—

*1 When specifying Bit data, specify the leading of the channel.

●Description of Operation

When Execution condition is TRUE, with the data type specified by the upper 4 bits of **Param**, adds the lower 12-bit data size (in bytes) stored in **Src** to the area below the data size stored in **Dst**.
When adding data for the first time, store 0 in **Dst** and then execute the Function.

CPMSET Function is used for creating CIP string data and numeric data.
For CIP data type of , refer to "KV EtherNet/IP Function User's Manual".

Upper 4 bits	Lower 12 bits
0：Non-string	1 to 4095(001H to FFFH): Specifies data size
1：String	0 to 1999(000H to 7CFH): Specifies the number of characters to be converted to CIP string type data. If 0 is specified, up to NUL(00H) will be converted. If the specified number of characters contains NUL(00H), 00H is added to the data following NUL(00H).
2 or more: System-reserved	—

For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

CPMSET (auiValue[0], uiParam, auiDst[0]);

CPMGET Gets CIP Message

CIP message getting
CPMGET ([EN], Src, Param, Offset, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1	Specifies the leading of the Receive data storage destination.	.U	.S	.D	.L	.F	.DF	.B	.T	—
Param	Specifies the Data type/Data size (in bytes) of Receive data to be obtained, or their Storage destination.* 1	.U	.U	.U	.U	—	—	—	—	○
Offset*1	Specifies the Byte offset from the leading to specify the Receive data get position, or its Storage destination.	.U	.S	.D	.L	—	—	—	—	—
Dst*1	Specifies the leading of Get data storage destination.	.U	.S	.D	.L	.F	.DF	.B	.T	—
R	None	—	—	—	—	—	—	—	—	—

*1 When specifying Bit data, specify the leading of the channel.

●Description of Operation

When Execution condition is TRUE, stores the Receive data (that stored in **Src** and under the Offset position (specified by **Offset**)) in **Dst**, with the data type specified by the upper 4 bits of **Param**, and the data size (in bytes) specified by the lower 12 bits of **Param**. After storing the data, **Offset** is updated.
CPMGET Function is used for getting the Receive CIP string data and numeric data.
For CIP data type of , refer to "KV EtherNet/IP Function User's Manual".

Upper 4 bits	Lower 12 bits
0：Non-string	1 to 4095(001H to FFFH): Specifies data size
1：String	0 to 1999(000H to 7CFH): Specifies the number of characters to be converted to CIP string type data. If 0 is specified, up to NUL(00H) will be converted. If the specified number of characters contains NUL(00H), 00H is added to the data following NUL(00H).
2 or more: System-reserved	—

For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

●Expression sample

CPMGET(auiValue[0], uiParam, uiOffset, auiDst[0]);

*_TO_** Converts Data Type (Specifies Data Type)

Data type converting
[R:=] *_TO_** (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*2	Specifies the Convert target data.	Data types specified by *						—	—	○
R*1*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts the data of data type * stored in In to the data type **, and stores the result in R.

* And ** are one of UINT/INT/UDINT/DINT/REAL/LREAL.

In \ R	UINT	INT	UDINT	DINT	REAL	LREAL
UINT		D1	A	A	A	A
INT	D1		D1	A	A	A
UDINT	D2	D2		D1	B	A
DINT	D2	D2	D1		B	A
REAL	C	C	C	C		B
LREAL	C	C	C	C	B	

Legend:
A: Convert value as is
B: If the data stored in In exceeds the Valid digit amount range of the data type **, the precision will deteriorate.
Example)(DINT)1234567890→(REAL)1.234567E09
Example)(LREAL)3.1415926535→(REAL)3.1415926
C: Set the value after decimal point to an integer according to the following rules.

The value after decimal point	Processing	Example
Less than 0.5	Round down	REAL_TO_INT(1.4)→1 REAL_TO_INT(-1.4)→-1
0.5	Round down to the closest even integer	REAL_TO_INT(1.5)→2 REAL_TO_INT(2.5)→2 REAL_TO_INT(-1.5)→-2 REAL_TO_INT(-2.5)→-2
Greater than 0.5	Round up	REAL_TO_INT(1.6)→2 REAL_TO_INT(-1.6)→-2

If the converted value is out of the data type range, R is an invariant value.
D1: If the data stored in In is outside the range of data type **, the value is stored as it is without conversion.
D2: Converts the lower 16 bits of the data stored in In.

●Expression sample

diValue := **REAL_TO_DINT** (rValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

TO_** Converts Data Type (Overloads)

Data type converting
[R:=] TO_ (In);**

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*2	Specifies the Convert target data.	.U	.S	.D	.L	.F	.DF	—	—	○
R*1*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts the data stored in In to the data type **, and stores the result in R.
** is one of UINT/INT/UDINT/DINT/REAL/LREAL.

For the operation during conversion, 「*_TO_**function」(page 7-39)

●Expression sample

diValue := **TO_DINT** (rValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

*_TRUNC_** Rounds Down Decimal (Specifies Data Type)

Decimal rounding down
[R:=] *_TRUNC_ (In);**

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*2	Specifies the Convert target data.	Data types specified by *						—	—	○
R*1*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Round down the numbers after the decimal point of the data type * stored in In, converts it to an integer data of data type **, and stores the result in R.
** is one of REAL/LREAL.
** is one of UINT/INT/UDINT/DINT.
If the data stored in In is outside the Floating-point real number range, R is an indefinite value.

D1: If the data stored in In is outside the range of data type **, the value is stored as it is without conversion.

●Expression sample

diValue := **REAL_TRUNC_DINT** (rValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

TRUNC_** Rounds Down Decimal (Overloads)

Decimal rounding down
[R:=] TRUNC_ (In);**

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*2	Specifies the Convert target data.	.F	.F	.F	.F	.F	.DF	—	—	○
R*1*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Round down the numbers after the decimal point of the data type * stored in In, converts it to an integer data of data type **, and stores the result in R.
** is one of UINT/INT/UDINT/DINT.
If the data stored in In is outside the Floating-point real number range, R is an indefinite value.
D1: If the data stored in In is outside the range of data type **, the value is stored as it is without conversion.

For details on operation during conversion, "*_TRUNC_** function"(page 7-40)

●Expression sample

diValue := **TRUNC_DINT** (rValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

*_TO_BCD_** Converts BCD Data Type (Specifies Data Type)

Data type converting
[R:=] *_TO_BCD_ (In);**

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target data.	Data types specified by *				—	—	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.

●Description of Operation

Converts the data of data type * stored in In to the data type **, and stores the result in R.
* and ** is one of UINT / INT / UDINT / DINT.
If the data stored in In is out of range, R is an indefinite value.

●Expression sample

diValue := **INT_TO_BCD_DINT** (iValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

TO_BCD_** Converts BCD Data Type (Overloads)

Data type converting
[R:=] TO_BCD_ (In);**

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target data.	.U	.S	.D	.L	—	—	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.

●Description of Operation

Converts the data stored in In to BCD data of data type **, and stores the result in R.
The data type of In is one of UINT/INT/UDINT/DINT.
** is one of UINT/INT/UDINT/DINT.

For details on operation during convertion, "*_TO_BCD_** function"(page 7-42)

●Expression sample

diValue := **TO_BCD_DINT** (iValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

*_BCD_TO_** Converts Data Type (Specifies Data Type)

Data type converting
[R:=] *_BCD_TO_ (In);**

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target data.	Data types specified by *				—	—	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.

●Description of Operation

Converts BCD data of data type * stored in In to an integer data of data type **, and stores the result in R.
* And ** are one of UINT/INT/UDINT/DINT.
If BCD data stored in In is outside the range, R is an indefinite value.

●Expression sample

iValue := **DINT_BCD_TO_INT** (diValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

BCD_TO_** Converts Data Type (Overloads)

Data type converting
[R:=] BCD_TO_ (In);**

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target data.	.U	.S	.D	.L	—	—	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type different from the data type specified by ** is specified to R, the type conversion is automatically executed during conversion.

●Description of Operation

Converts BCD data stored in In to an integer data of data type **, and stores the result in R.
The data type of In is one of UINT/INT/UDINT/DINT.
** is one of UINT/INT/UDINT/DINT.
For details on operation during conversion, "*_BCD_TO_** function"(page 7-43)

●Expression sample

diValue := **BCD_TO_DINT** (iValue);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Convert Instructions".

Floating Point Function

FLOAT Converts BIN Data to Single precision floating point real number data

Single precision floating point converting
[R:=] FLOAT (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	.U	.S	.D	.L	–	.DF*2	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type other than .F is specified to R, type conversion is automatically performed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts BIN data (16-bit/32-bit (unsigned / signed) and Double precision floating point real number) stored in In to Single precision floating point real number data, and stores the result in R.

For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

rValue := **FLOAT** (uiValue);

Point **If the integer value of In exceeds the range of -16777216 to 16777215 (24-bit BIN), the 25th and subsequent bits are rounded.**

DFLOAT Converts BIN Data to Double precision floating point real number data

Double precision floating point converting
[R:=] DFLOAT (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target data, or its Storage destination.	.U	.S	.D	.L	.F	–	–	–	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in the 64-bit internal register.	.U	.S	.D	.L	.F	.DF*2	–	–	–

*1 If a type other than .DF is specified to R, type conversion is automatically performed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts BIN data (16-bit/32-bit (unsigned / signed) and Single precision floating point real number) stored in In to Double precision floating point real number data, and stores the result in R.

For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

lValue := **DFLOAT** (uiValue);

INTG Converts Single/Double precision floating point real number data to BIN Data

Integer converting
[R:=] INTG (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	–	–	–	–	.F	.DF*3	–	–	○
R*1*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 C/T/Z cannot be used.
*2 If a type other than .L is specified to R, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

Point

- **The Single precision floating point real number range is as follows:**

  −3.4E+38 ≦ N ≦ −1.4E−45
  N = 0
  1.4E−45 ≦ N ≦ 3.4E+38
  (Number of effective digits: about 7 digits)

- **The Double precision floating point real number range is as follows:**

  −1.79E+308 ≦ N ≦ −2.23E−308
  N = 0
  2.23E−308 ≦ N ≦ 1.79E+308
  (Number of effective digits: about 16 digits)

●Description of Operation

Converts Single/Double precision floating point real number data stored in In to 32-bit BIN data (signed), and stores the result in R.

For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

uiValue := **INTG** (rValue);

Point

- **If In is outside the Single precision floating point real number range, or the converted data is outside the signed 16/32 bit BIN range, the bit pattern of In is stored in R.**
- **When In is outside the Double precision floating point real number range, or the converted data is outside the Signed 16/32-bit BIN range, the lower 32-bit pattern of In is stored in R.**

INT Converts Single/Double precision floating point real number data to BIN Data

Integer converting
[R:=] INT (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	—	—	—	—	.F	.DF*3	—	—	○
R*1*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	—	—	—

*1 C/T/Z cannot be used.
*2 If a type other than .L is specified to R, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

Point
- **For details on Single/Double precision floating point real number ranges, refer to "INTG function"(page 7-45).**

●Description of Operation

This Function is similar to "INTG Function" except the Function name is different.

DISF Disperses Mantissa/Exponent of Single precision floating point real number

F dispersing
DISF ([EN], In, InOut1, InOut2);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1*3	Specifies the Single precision floating point real number to be Mantissa/Exponent dispersed.	—	—	—	—	.F	—	—	—	○*2
InOut1*1	Specifies the separated mantissa storage destination.	—	—	.L	.L	—	—	—	—	—
InOut2	Specifies the separated exponent storage destination.	.S	.S	—	—	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 32 bits. When a position other than the channel leading such as R002 is specified, the operation will be processed across the next channel.
*2 Set Constant with Single precision floating point real number range.
*3 C/T/Z cannot be used.

Point
For details on the Single precision floating point real number range, refer to "INTG function"(page 7-45).

●Description of Operation

When Execution condition is TRUE, disperses the mantissa/exponent of 32-bit Single precision floating point real number specified by **In**, stores the mantissa part as 32-bit BIN data (signed) in the specified **InOut1**, and the exponent part as 16-bit BIN data (signed) in specified **Out2**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

DISF (bEnable, rValue, diValue, iValue);

Point

- **Errors may occur due to the DISF function converts Single precision floating point real number (BIN) to the mantissa part (DEC) and the exponent part (DEC).**
- **If the data to be dispersed is outside the Single precision floating point real number range, the Function will not be executed.**

UNIF Unites Mantissa/Exponent to Single precision floating point real number

F uniting

[R:=] UNIF (In1, In2);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1	Specifies the Unite target mantissa storage destination.	—	—	.L	.L	—	—	—	—	—
In2	Specifies the Unite target exponent storage destination.	.S	.S	.S	.S	—	—	—	—	—
R*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 32 bits. When a position other than the channel leading such as R002 is specified, the operation will be processed across the next channel.

*2 If a type other than .F is specified to R, type conversion is automatically performed during conversion.

*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Stores the united 32-bit Single precision floating point real number value into R, with the signed 32-bit BIN data specified by In1 as the mantissa, and the signed 16-bit BIN data specified by In2 as the exponent.

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

rValue := **UNIF** (diValue, iValue);

Point

- **Errors may occur due to the UNIF function converts the mantissa part (DEC) and the exponent part (DEC) to Single precision floating point real number (BIN).**
- **If the Operation result is outside the Single precision floating point real number range, 0 is stored in R.**

EXP Operates Exponent Function

Exponent
[R:=] EXP (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 EXP Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point
- **For details on Single/Double precision floating point real number ranges, refer to "INTG function"(page 7-45).**

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the exponential function value with Single precision floating point real number stored in In as an exponent.
  The calculation result is stored in R as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Exponent Function value with the Double precision floating point real number stored in In as the exponent.
  The calculation result is stored in R as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

rValue := **EXP** (rValue1);

Point
If the value to be operated or the Operation result is outside the Single precision floating point real number range, the value of In is stored in R.

EXPT Exponentiation

Exponentiation
[R:=] EXPT (In, Pwr);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
Pwr	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 EXPT Function has .F identification type when Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Multiplies the data stored in **In** by the power of **Pwr**, and stores the result in **R**.

Point
- **If In= 0, and Pwr= 0, 1 is stored in R.**

●Expression sample

rValue1 := **EXPT** (rValue2, rValue3);

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

LN Operates Natural Logarithm

[R:=] LN (In); ($\log_e$)

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 1EXP Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to **R**, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point
- **For details on Single/Double precision floating point real number ranges, refer to "INTG function"(page 7-45).**

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the natural logarithm of Single precision floating point real number stored in **In**.
  The calculation result is stored in **R** as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Natural logarithm of Double precision floating point real number stored in **In**.
  The calculation result is stored in **R** as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

rValue := **LN** (rValue1);

Point
If the value to be calculated is 0 or negative, or the Operation result is outside the Floating-point real number range, the value of In is stored in R.

LOG Operates Constant Logarithm

log10
[R:=] LOG (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 LOG Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point
- **For details on Single/Double precision floating point real number ranges, refer to "INTG function"(page 7-45).**

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the Constant logarithm of Single precision floating point real number stored in In.
  The calculation result is stored in R as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Constant logarithm of Single precision floating point real number stored in In.
  The calculation result is stored in R as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

●Expression sample

rValue := **LOG** (rValue1);

RAD Converts Degree (°) to Radian Unit

Radian
[R:=] RAD (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 RAD Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point
- **For details on Single/Double precision floating point real number ranges, refer "INTG function"(page 7-45).**

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Converts the Angle (in degrees) of Single precision floating point real number stored in In to Radian.

The calculation result is stored in **R** as a 32-bit Single precision floating point real number.

- **In: When the suffix of the Convert target is .DF**
  Converts the Angle (in degrees) of Double precision floating point real number stored in **In** to Radian.
  The calculation result is stored in **R** as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference

Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

rRadian := **RAD** (uiDegree);

Point

If the value to be calculated is outside the Floating point real number range, the value of In is stored in R.

DEG Converts Radian to Degree (°) Unit

Degree (°)
[R:=] DEG (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target or Convert target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 DEG Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to **R**, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point

- **For details on Single/Double precision floating point real number ranges, refer to "INTG function"(page 7-45).**

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Converts the Angle (in degrees) of Single precision floating point real number stored in **In** to Degree (°) unit.
  The calculation result is stored in **R** as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Converts the Angle (in degrees) of Double precision floating point real number stored in **In** to Degree (°) unit.
  The calculation result is stored in **R** as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference

Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

iDegree := **DEG** (rRadian);

Point **If the value to be calculated is outside the Floating point real number range, the value of In is stored in R.**

SIN Converts Angle (Radian) to Sine Value

[R:=] SIN (In);

Sine

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 SIN Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point

- **The Single precision floating point real number range that can be processed by this Function is**

  −1.6777216E+7 ≦ N ≦ −1.4E−45
  N = 0
  1.4E−45 ≦ N ≦ 1.6777216E+7

  (Number of effective digits: about 7 digits).
- **The Double precision floating point real number range that can be processed by this Function is**

  −9.007199254740992E+15 ≦ N ≦ −2.23E−308
  N = 0
  +2.23E−308 ≦ N ≦ +9.007199254740992E+15

  (Number of effective digits: about 16 digits).

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the Sine (SIN) value of the Angle (in radians) of Single precision floating point real number stored in **n**.

  The calculation result is stored in **R** as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Sine (SIN) value of the Angle (in radians) of Double precision floating point real number stored in **n**.
  The calculation result is stored in **R** as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

rValue := **SIN** (rRadian);

Point

If the value to be calculated is outside the Floating point real number range that can be processed by the SIN Function, the value of In is stored in R.

COS Converts Angle (Radian) to Cosine Value

Cosine

[R:=] COS (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 COS Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified for Return value, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point

- **The Single precision floating point real number range that can be processed by this Function is**

  −1.6777216E+7 ≦ N ≦ −1.4E−45
  N = 0
  1.4E−45 ≦ N ≦ 1.6777216E+7

  (Number of effective digits: about 7 digits).
- **The Double precision floating point real number range that can be processed by this Function is**

  −9.007199254740992E+15 ≦ N ≦ −2.23E−308
  N = 0
  +2.23E−308 ≦ N ≦ +9.007199254740992E+15

  (Number of effective digits: about 16 digits).

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the Cosine (COS) value of the Angle (in radians) of Single precision floating point real number stored in **n**.
  The calculation result is stored in **R** as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Cosine (COS) value of the Angle (in radians) of Double precision floating point real number stored in **n**.
  The calculation result is stored in **R** as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference

Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

rValue := **COS** (rRadian);

Point

If the value to be calculated is outside the Floating point real number range that can be processed by the COS Function, the value of In is stored in R.

TAN Converts Angle (Radian) to Tangent Value

Tangent
[R:=] TAN (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 TAN Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point

- **The Single precision floating point real number range that can be processed by this Function is**

  −1.6777216E+7 ≦ N ≦ −1.4E−45
  N = 0
  1.4E−45 ≦ N ≦ 1.6777216E+7

  (Number of effective digits: about 7 digits).
- **The Double precision floating point real number range that can be processed by this Function is**

  −9.007199254740992E+15 ≦ N ≦ −2.23E−308
  N = 0
  +2.23E−308 ≦ N ≦ +9.007199254740992E+15

  (Number of effective digits: about 16 digits).

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the Tangent (TAN) value of the Angle (in radians) of Single precision floating point real number stored in n.
  The calculation result is stored in R as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Tangent (TAN) value of the Angle (in radians) of Double precision floating point real number stored in n.
  The calculation result is stored in R as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference

Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

rValue := **TAN** (rRadian);

Point

If the value to be calculated is outside the Floating point real number range that can be processed by the TAN Function, the value of In is stored in R.

ASIN Converts Sine Value to Angle (Radian)

Arc sign
[R:=] ASIN (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 ASIN Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

Point
- **For details on Single/Double precision floating point real number ranges, refer to "INTG function"(page 7-45).**

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the Angle (in radians) from the Sine (SIN) value of Single precision floating point real number stored in n
  .
  The calculation result is stored in R as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Angle (in radians) from the Sine (SIN) value of Double precision floating point real number stored in n.
  The calculation result is stored in R as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference
Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

rRadian := **ASIN** (rValue);

Point
If the value to be calculated is outside the range of −1.0 to 1.0, the value of In is stored in R.

ACOS Converts Cosine Value to Angle (Radian)

Arc cosine
[R:=] ACOS (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 ACOS Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the Angle (in radians) from the Cosine (COS) value of Single precision floating point real number stored in In.
  The calculation result is stored in R as a 32-bit Single precision floating point real number.
- **In: When the suffix of the Convert target is .DF**
  Calculates the Angle (in radians) from the Cosine (COS) value of Double precision floating point real number stored in In.
  The calculation result is stored in R as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference

Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

rRadian := **ACOS** (rValue);

Point

If the value to be calculated is outside the range of −1.0 to 1.0, the value of In is stored in R.

ATAN Converts Tangent Value to Angle (Radian)

Arc tangent
[R:=] ATAN (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 ATAN Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

- **In: When the suffix of the Convert target is .U/.S/.D/.L/.F**
  Calculates the Angle (in radians) from the Tangent (TAN) value of Single precision floating point real number stored in **In**.
  The calculation result is stored in **R** as a 32-bit Single precision floating point real number.

- **In: When the suffix of the Convert target is .DF**
  Calculates the Angle (in radians) from the Tangent (TAN) value of Double precision floating point real number stored in **In**.
  The calculation result is stored in **R** as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference

Angle (°) and (rad) are calculated as follows:

$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

●Expression sample

rRadian := **ATAN** (rValue);

Point

If the value to be calculated is outside the Floating point real number range, the value of In is stored in R.

ATAN2 Converts Coordinate to Angle (Radian)

Arc tangent2
[R:=] ATAN2 (Y, X);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Y	Specifies the Operation target 1 or Operation target data 1.	.F	.F	.F	.F	.F	.DF*2	—	—	○
X	Specifies the Operation target 2 or Operation target data 2.	.F	.F	.F	.F	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 ATAN2 Function has .F identification type when the Argument is U/S/D/L/F, and the .DF type when DF. If a type different from the type of the Function is specified to **R**, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

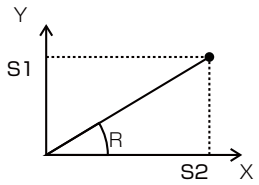
●Description of Operation

- **When the suffix of the Convert target 1(Y) and Convert target 2 (Y) is U/S/D/L/F**
  From the Y coordinate obtained by converting the value specified by **Y** to Single precision floating-point real number and the X coordinate obtained by converting the value specified by **X** to Single precision floating point real number, calculates the Angle (in radians) relative to X axis of Single precision floating point real number.
  The calculation result is stored in **R** as a 32-bit Single precision floating point real number.
- **When the suffix of the Convert target 1(Y) and Convert target 2 (Y) is DF**
  From the Y coordinate obtained by converting the value specified by **Y** to Double precision floating-point real number and the X coordinate obtained by converting the value specified by **X** to Double precision floating point real number, calculates the Angle (in radians) relative to X axis of Single precision floating point real number.
  The calculation result is stored in **R** as a 64-bit Double precision floating point real number.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Floating Point Instructions".

Reference

ATAN instruction calculates the Angle up to $-\pi/2$(rad) < **R** < $\pi/2$(rad) by the Tangent (TAN) value, whereas the ATAN2 instruction calculates the Angle up to $-\pi$(rad) < **R** ≦ π(rad) by Coordinates (**X**, **Y**)



$$1^\circ = \frac{\pi}{180} \fallingdotseq 0.01745\ (\text{rad}),\quad 1\ (\text{rad}) = \frac{180}{\pi} \fallingdotseq 57.23^\circ$$

If S1=0 and S2≧0, R(Return value)=0°=0(rad).

If S1=0 and S2<0, R(Return value)=180°=π(rad).

If S1>0 and S2=0, R(Return value)=90°=π/2(rad)).

If S1<0 and S2=0, R(Return value)=-90°=-π/2(rad).

●Expression sample

rRadian := **ATAN2** (rValueY, rValueX);

Point

If the value to be calculated is outside the Single precision floating point real number range, the value of R does not change.

Character Process Function

ASC Converts BIN Data to HEX ASCII Code

HEX ASCII code converting
[R:=] ASC (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the HEX ASCII character code storage destination of the Convert target, or the Convert target data.	.U	.U	.U	.U	–	–	–	–	○
R*2	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 Even if a Convert target other than the Bit data channel leading such as R1002 is specified, operates correctly across the next channel.
*2 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts the lower 8 bits of BIN data stored in In to the HEX ASCII character code, and stores the 16-bit data in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **ASC** (uiData);

RASC Converts HEX ASCII Code to BIN Data

HEX ASCII code reverse-converting
[R:=] RASC (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2	Specifies the BIN data storage destination of the Convert target, or the Convert target data.	.U	.U	.U	.U	–	–	–	–	○
R*3	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*4	–	–	–

*1 32-bit data can also be used to In, while the Convert target is only the lower 16 bits.
*2 Even if a Convert target other than the Bit data channel leading such as R1002 is specified to In, operates correctly across the next channel.
*3 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*4 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts the HEX ASCII character code (two characters) of the 16 bits stored in In to BIN data, and stores the 16-bit data (lower 8 bits are the Convert value, and the upper 8 bits are 0) of the result in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)

「ASCII Code Table」(page A-10)

For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

Point

- **In must store the ASCII code values with the codes 0 to 9 (30H to 39H) indicating both upper 8 bits and lower 8 bits or A to F (41H to 46H), or the codes NUL (00H) indicating upper 8 bits and 0 to 9 (30H to 39H) indicating lower 8 bits or A to F (41H to 46H).**
  When either of the upper 8 bits or lower 8 bits are outside this range, the Function is not executed.
- **If there is an ASCII code that cannot be processed, the value of In is stored in R.**

●Expression sample

uiValue := **RASC** (uiData);

DASC Converts BIN Data to DEC ASCII String

DEC ASCII code converting

[R:=] DASC (In, Param);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2	Specifies the BIN data storage destination of the Convert target, or the Convert target data.	.U	.S	.D	.L	—	—	—	—	○*3
Param*4	Specifies the Convert Parameter or the Convert parameter storage destination. (0 to 3)	.U	.U	.U	.U	—	—	—	—	○
R*2*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

*2 For Bit data, specify the leading of the channel.

*3 Constants cannot be specified by 16# (HEX).

*4 Specify the Convert parameter within the range 0 to 3.
Description of Convert parameter **Param**

Value	Details	CR2814	CR2815
0	Zero suppress OFF, Sign omission OFF	0	0
1	Zero suppress ON, Sign omission OFF	1	0
2	Zero suppress OFF, Sign omission ON	0	1
3	Zero suppress ON, Sign omission ON	1	1

Point

The Function may not operate correctly when a value outside of the range is specified to the Convert parameter.

*5 T/C/CR/Z cannot be used.

●Description of Operation

Converts the BIN data stored in **In** to DEC ASCII code string according to the Convert parameter specified with **Param**, and stores the result in **R**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **DASC** (uiValue, INT#3);

Point

- **When Constant is used for the Convert target data, 16# (HEX specifying) is unavailable.**
- **Only 1-byte characters (ASCII code) can be specified as the Convert target data.**
- **Pay attention when changing or referencing CR2814/CR2815 by the interrupt program.**
- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored in R.**
  - **When Bit data is specified for R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for R by Indirect designation.**

STR Converts Numerical Value Data to DEC ASCII String

DEC ASCII code converting

[R:=] STR (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*3	Specifies the BIN data storage destination of the Convert target, or the Convert target data.	.U	.S	.D	.L	.F	—	—	—	○
R*2*4	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 For Bit data, specify the leading of the channel.
*3 Constants cannot be specified by 16# (HEX).
*4 T/C/CR/Z cannot be used.

●Description of Operation

Performs DEC ASCII string conversion or Single precision floating point string conversion according to the Argument type specified by In.

For details on operation, refer to "DASC function"(page 7-60).
「About the operation flags (CR2009 to CR2012)」(page 1-8)

Point

- **When Constant is used for the Convert target data, 16# (HEX specifying) is unavailable.**
- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored in R.**
  - **When Bit data is specified for R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for R by Indirect designation.**

●Expression sample

strString := **STR** (uiValue);

RDASC Converts DEC ASCII String to BIN Data

DEC ASCII code reverse-converting
[R:=] RDASC (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2	Specifies the HEX ASCII string storage destination of the Convert target, or the HEX ASCII string.	—	—	—	—	—	—	—	.T	○
R*3*4	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 T/C/CR/Z cannot be used.
*3 When Bit data is specified, operation is performed with consecutive 32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*4 If a type other than .L is specified to R, type conversion is automatically performed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts the stored DEC ASCII string starting with In to 32-bit BIN data (signed), and stores the result in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **RDASC** (strString);

Point

- **Specify a string consists of "16# (HEX specifying) 30 to 39 ('0' to '9') and 16# (HEX specifying) 2B/2D ('±' sign: only for signed conversion)" to the DEC ASCII string specified by the Convert target data.**
- **The Function will not be executed if an ASCII character string other than the above is specified for In.**
- **If the ASCII string specified by In does not have NUL (00H), the Function converts a string of up to 11 characters.**
- **When any of the following conditions is TRUE, 0 is stored in R.**
  - **When there is an invalid character in In.**
  - **When the converted value is outside the signed 32-bit BIN data range.**
  - **When Bit data is specified for In and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In by Indirect designation.**

HASC Converts BIN Data to HEX ASCII String

HEX ASCII code converting
[R:=] HASC (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the BIN data storage destination of the Convert target, or the Convert target data. (unsigned BIN data)	.U	.U	.D	.D	–	–	–	–	○
Count*4	Specifies the number of numerical value (HEX) digits for the Convert result storage. (0 to 8)	.U	.U	.U	.U	–	–	–	–	○*2
R*3*5	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register. HASC Function has .T identification type.	–	–	–	–	–	–	–	.T	–

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits.
When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 Specify the Constant by 16# (HEX specifying).
*3 It is necessary to specify the .T suffix for the destination to store **R**.
When specifying Bit data to **R**, specify the leading of the channel.
*4 Specify the number of Convert result storage digits (lower digits) in the range of 0 to 8 for the number of Convert digits.
(If 0 is specified, the number of effective digits is stored with zero suppression)
*5 T/C/CR/Z cannot be used.

●Description of Operation

Converts the BIN data stored in **In** to a HEX ASCII code string with the terminal code NUL (00H) added, and stores the result in **R** for the specified **Count**. (When **Count** = 0, all **In** data is zero-suppressed and converted)

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **HASC** (uiValue, INT#3);

Point
- **When using Constant for the Convert data, DEC specifying is unavailable.**
- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored in R.**
  - **When Bit data is specified for R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for R by Indirect designation.**
  - **In the case of 5 or more is specified when Count is .U, or 9 or more is specified when Count is .D.**

RHASC Converts HEX ASCII String to BIN Data

HEX ASCII code reverse-converting
[R:=] RHASC (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2	Specifies the HEX ASCII string storage destination of the Convert target, or the HEX ASCII string.	—	—	—	—	—	—	—	.T	○
R*3*4	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 T/C/CR/Z cannot be used.
*3 When Bit data is specified, operation is performed with consecutive 32 bits.
When a position other than the channel leading such as R002 is specified, 32/32 bits will be processed across the next channel.
*4 If a type other than .D is specified to R, type conversion is automatically performed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts the stored HEX ASCII string starting with In to 32-bit BIN data (unsigned), and stores the result in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

udiValue := **RHASC** (strString);

Point

- **In the HEX ASCII string specified in the Convert target data In, the value of the range of 16# (HEX specifying) 30 to 39 ('0' to '9') or 16# (HEX specifying) 41 to 46 (A' to 'F') must be stored.**
- **Use the Uppercase A to F (41H to 46H) in HEX ASCII string. Lowercase is not supported.**
- **When any of the following conditions is TRUE, 0 is stored in R.**
  - **When there is an invalid character in In.**
  - **When the converted value is outside the unsigned 32-bit BIN data range.**
  - **When Bit data is specified for In and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In by Indirect designation.**

FASC Converts Single precision floating point real number to String

Single precision floating point real number converting

[R:=] FASC (Fmt, FullCount, SubDecCount, In, Param);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Fmt	Specifies the display specification format (decimal point format, exponential format).	.U	.U	.U	.U	—	—	—	.T	○
FullCount	Specifies the total number of digits.	.U	.U	.U	.U	—	—	—	—	○
SubDecCount	Specifies the number of decimal digits.	.U	.U	.U	.U	—	—	—	—	○
In*1*2*4	Specifies the Single precision floating point real number storage destination of the Convert target, or the Single precision floating point real number data.	—	—	—	—	.F	—	—	—	○
Param	Specifies Zero suppress and Sign omission.	.U	.U	.U	.U	—	—	—	—	○
R*3*4	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 When Bit data is specified, occupies consecutive 32 bits. When a position other than the channel leading such as R002 is specified, 32 bits will be processed across the next channel.

*2 For details on the Single precision floating point real number range, refer to "INTG function"(page 7-45).

*3 For Bit data, specify the leading of the channel.

*4 T/C/CR/Z cannot be used.

●Description of Operation

Converts the Single precision floating point real number stored in **In** to the string added the end code NUL(00H) according to the Convert parameter stored in **Fmt** to **SubDecCount** and **Param**, and stores the result in **R**.

Specify the Convert parameter as follows:

Fmt	Display specification format	Specifies "0" when converting to a decimal point format string, and "1" when converting to an exponent format string.
FullCount	Total number of digits	Specify within the range 1 to 24 digits.
SubDecCount	Number of decimal digits	Specify within the range 0 to 7 digits.

The content to be specified in the Convert parameter is as follows.

Param value	Details	CR2814	CR2815
0	Zero suppress OFF, Sign omission OFF	0	0
1	Zero suppress ON, Sign omission OFF	1	0
2	Zero suppress OFF, Sign omission ON	0	1
3	Zero suppress ON, Sign omission ON	1	1

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **FASC** (INT#0, INT#10, INT#3, rValue, uiParam);

Point

- When Constant is used for the Convert target data, 16# (HEX specifying) is unavailable.
- Pay attention when changing or referencing CR2814/CR2815 by the interrupt program.
- When any of the following conditions is TRUE, the end code (NUL(00H)) is stored in R.
  - When the Argument value is out of range
  - When you specify the value outside the Single precision floating point real number range.
  - When the number of string digits to be stored has exceeded the total number of digits
  - When Bit data is specified to R and a position other than the channel leading is specified.
  - When Timer/Counter is specified to R by Indirect designation.
  - When the total/decimal digits do not meet the following conditions.

	Decimal point format	Exponent format
SubDecCount = 0	1 ≦ FullCount ≦ 24	5 ≦ FullCount ≦ 24
0 < SubDecCount ≦ 7	SubDecCount+ 2 ≦ FullCount 3 ≦ FullCount ≦ 24	SubDecCount + 6 ≦ FullCount 7 ≦ FullCount ≦ 24

RFASC Converts String to Single precision floating point real number

Single precision floating point real number reverse-converting

[R:=] RFASC (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*3*4	Specifies the Single precision floating point string storage destination of the Convert target, or the Single precision floating point string.	—	—	—	—	—	—	—	.T	○
R*2*4*5	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*6	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 When Bit data is specified, it occupies consecutive 32 bits. When a position other than the channel leading such as R002 is specified, 32 bits will be processed across the next channel.
*3 A NUL (00H) is required at the end of the Single precision floating point string specified by the Convert target.
*4 T/C/CR/Z cannot be used.
*5 If a type other than .F is specified to R, type conversion is automatically performed during conversion.
*6 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts the stored Single precision floating point string starting with In to the Single precision floating point real number data, and stores the result in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

rValue := **RFASC** (strString);

Point

- **The Single precision floating point string specified by the Convert target data should only stores 16# (HEX specifying) 30 to 39 ('0' to '9')/2B ('+')/2C ('-')/45 ('E') and 2E ('.').**
- **When any of the following conditions is TRUE, 0 is stored in R.**
  - **When the string length is 0**
  - **When the string format is incorrect**
  - **When the converted numerical value exceeds the Single precision floating point real number range**
  - **When the number of string digits to be stored has exceeded the total number of digits**
  - **When Bit data is specified for In and a position other than the channel leading is specified.**

VAL Converts String to Single precision floating point real number

Single precision floating point real number reverse-converting

[R:=] VAL (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2*4	Specifies the Single precision floating point string storage destination of the Convert target, or the Single precision floating point string.	—	—	—	—	—	—	—	.T	○
R*3*5	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*6	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 A NUL (00H) is required at the end of the Single precision floating point string specified by the Convert target.
*3 When Bit data is specified, occupies consecutive 32 bits. When a position other than the channel leading such as R002 is specified, 32 bits will be processed across the next channel.
*4 T/C/CR/Z cannot be used.
*5 If a type other than .F is specified to R, type conversion is automatically performed during conversion.
*6 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

This Function is similar to "RFASC Function" except the Function name is different.
For details on operation, refer to "RFASC function"(page 7-66).

LEN Detects String Length

String length detecting

[R:=] LEN (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2*4	Specifies the Storage destination to store the target string for calculating string length, or the target string.	—	—	—	—	—	—	—	.T	○*1
R*3*4	The Return value that stores the Convert result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 To specify a string using Constant, enclose with " ".
*2 For Bit data, specify the leading of the channel.
*3 T/C/CR/Z cannot be used.
*4 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Counts the number of stored strings starting with **In**, and stores the result in **R** as 16-bit BIN data.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **LEN** (strString);

Point

- **2-byte characters are processed as two characters.**
- **End code (NUL (00H)) cannot be counted.**
- **When any of the following conditions is TRUE, 0 is stored in R.**
  - **When there is no end code (NUL (00H))**
  - **When the string including the end code (NUL (00H)) exceeds 2000 characters**
  - **When Bit data is specified for In by Indirect designation, and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In by Indirect designation.**

SRGHT Right-substrings

Right-substring
[R:=] SRGHT (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Substring target string storage destination, or the Substring target string.	—	—	—	—	—	—	—	.T	○*2
Count*4	Specifies the number of Substring characters (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For **Count**, specify the number of characters (bytes) from the right end (end) in the range of 0 to 1999.
When 0 is specified, only one character of NUL (00H) is stored in the Return value.
*5 T/C/CR/Z cannot be used.

●Description of Operation

Performs substring from the right end for the stored string starting with **In** by the number of characters (bytes) specified by **Count**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **SRGHT** (strAllString, INT#40);

Point

- When any of the following conditions is TRUE, the end code (NUL(00H)) is stored.
  - When there is no end code (NUL (00H)) in the string specified by In.
  - When the string including the end code (NUL (00H)) exceeds 2000 characters
  - When the string stored after R exceeds the Device/Variable range.
  - When Bit data is specified for In/R and a position other than the channel leading is specified.
  - When Timer/Counter is specified for In/R by Indirect designation.

RIGHT Right-substrings

Right-substring
[R:=] RIGHT (In, L);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Substring target string storage destination, or the Substring target string.	—	—	—	—	—	—	—	.T	○*2
L*4	Specifies the number of Substring characters (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For L, specify the number of characters (bytes) from the right end (end) in the range of 0 to 1999.
When 0 is specified, only one character of NUL (00H) is stored in the Return value.
*5 T/C/CR/Z cannot be used.

●Description of Operation

This Function is similar to "SRGHT Function" except the Function name is different.
For details on operation, refer to "SRGHT function"(page 7-68).

SLEFT Left-substrings

Left-substring
[R:=] SLEFT (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Substring target string storage destination, or the Substring target string.	—	—	—	—	—	—	—	.T	○*2
Count*4	Specifies the number of Substring characters (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.

*4 For **Count**, specify the number of characters (bytes) from the right end (end) in the range of 0 to 1999.
When 0 is specified, only one character of NUL (00H) is stored in the Return value.

*5 T/C/CR/Z cannot be used.

●Description of Operation

Performs substring from the left end for the stored string starting with **In** by the number of characters (bytes) specified by **Count**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **SLEFT** (strAllString, INT#40);

Point

- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored.**
  - **When there is no end code (NUL (00H)) in the string specified by In.**
  - **When the string including the end code (NUL (00H)) exceeds 2000 characters**
  - **When the string stored after R exceeds the Device/Variable range.**
  - **When Bit data is specified for In/R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In/R by Indirect designation.**

LEFT Left-substrings

Left-substring
[R:=] LEFT (In, L);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Substring target string storage destination, or the Substring target string.	—	—	—	—	—	—	—	.T	○*2
L*4	Specifies the number of Substring characters (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.

*2 To specify a string using Constant, enclose with " ".

*3 16# (HEX specifying) cannot be used.

*4 For **L**, specify the number of characters (bytes) from the right end (end) in the range of 0 to 1999.
When 0 is specified, only one character of NUL (00H) is stored in the Return value.

*5 T/C/CR/Z cannot be used.

●Description of Operation

This Function is similar to "SLEFT Function" except the Function name is different.
For details on operation, refer to "SLEFT function"(page 7-69).

SMID Mid-substrings

Mid-substring
[R:=] SMID (In, Pos, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Substring target string storage destination, or the Substring target string.	—	—	—	—	—	—	—	.T	○*2
Pos*6	Specifies the substring position (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
Count*4	Specifies the number of Substring characters (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For **Count**, specify the number of characters (bytes) from the right end (end) in the range of 0 to 1999.
When 0 is specified, only one character of NUL (00H) is stored in the Return value.
*5 T/C/CR/Z cannot be used.
*6 The leading character of the string specified by **In** is 0.

●Description of Operation

Performs substring from the position (bytes) specified by **Pos** starting with **In** by the number of characters (bytes) specified by **Count**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **SMID** (strAllString, INT#8, INT#16);

Point

- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored.**
  - **When there is no end code (NUL (00H)) in the string specified by In.**
  - **When the string including the end code (NUL (00H)) exceeds 2000 characters**
  - **When the string length of In＜Pos.**
  - **When the string stored after R exceeds the Device/Variable range.**
  - **When Bit data is specified for In/R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In/R by Indirect designation.**

MID Mid-substrings

Mid-substring
[R:=] MID (In, L, P);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*4	Specifies the Substring target string.	—	—	—	—	—	—	—	.T	○*2
L*3	Specifies the number of Substring characters (bytes), or its storage destination. (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○
P*5	Specifies the Substring start position (bytes), or its storage destination data. (1 to 1999)	.U	.U	.U	.U	—	—	—	—	○
R*1*4	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 Specify the leading of the channel by Bit data.
*2 To specify a string using Constant, enclose with " ".
*3 For L, specify the number of characters (bytes) from the Substring target characters in the range of 0 to 1999.
When 0 is specified, only one character of NUL (00H) is stored in the Return value.
*4 T/C/CR/Z cannot be used.
*5 The leading character of the string specified by In is 1.

●Description of Operation

Performs substring for the string stored in In from the position (bytes) specified by P by the number of characters (bytes starting with 1) specified by L, and stores the result in R after adding the code NUL (00H).

●Expression sample

strResult := **MID** (strValue, uiLength, uiPosition);

SRPLC Replaces String

String replacing
[R:=] SRPLC (In, ByStr, Pos, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Replace target string storage destination, or the Replace target string.	—	—	—	—	—	—	—	.T	○*2
ByStr*1*5	Specifies the Replace string storage destination, or the Replace string.	—	—	—	—	—	—	—	.T	○*2
Pos*6	Specifies the Replace start position (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
Count*4	Specifies the number of characters (bytes) to replace. (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	Stores the value of the result processed by the Function. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For Count, specify the number of characters (bytes) from the Replace start position in the range of 0 to 1999.
*5 T/C/CR/Z cannot be used.
*6 The leading character of the string specified by In is 0.

●Description of Operation

Replaces the stored strings starting with **In** with the stored string starting with **ByStr**, from the position (bytes) specified by **Pos**, by the number of characters (bytes) specified by **Count**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **SRPLC** (strAllString, strReplace, INT#10, INT#20);

Point

- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored.**
  - **When there is no end code (NUL (00H)) in the string specified by In/ByStr.**
  - **When the string including the end code (NUL (00H)) exceeds 2000 characters**
  - **When the string length of In < Pos.**
  - **When the replaced string length exceeds 2000 characters including the end code (NUL (00H)).**
  - **When Bit data is specified for In/ByStr/R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In/ByStr/R by Indirect designation.**

REPLACE Replaces String

String replacing
[R:=] REPLACE (In1, In2, L, P);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1*5	Specifies the Replace target string storage destination, or the Replace target string.	—	—	—	—	—	—	—	.T	○*2
In2*1*5	Specifies the Replace string storage destination, or the Replace string.	—	—	—	—	—	—	—	.T	○*2
L*4	Specifies the number of characters (bytes) to replace. (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
P	Specifies the Replace start position (bytes). (1 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	Return value that stores the result value processed by the Function. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For L, specify the number of characters (bytes) from the Replace start position in the range of 0 to 1999.
*5 T/C/CR/Z cannot be used.

●Description of Operation

Replaces the string stored in **In1** with the Replace string **In2**, from the position (bytes starting with 1) specified by **P**, by the number of characters (bytes) specified by **L**, and stores the result in **R** after adding the code NUL (00H).

This Function differs from the "SRPLC Function" in the order of arguments and values that P can get, but the operation of the Function is same.
For details on operation, refer to "SRPLC function"(page 7-72).

●Expression sample

strString := **REPLACE** (strAllString, strReplace, uiLength, uiPosition);

SINS Inserts String

String inserting
[R:=] SINS (In, ByStr, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Insert target string storage destination, or the Insert target string.	—	—	—	—	—	—	—	.T	○*2
ByStr*1*5	Specifies the Insert string storage destination, or the Insert string.	—	—	—	—	—	—	—	.T	○*2
Pos*6	Specifies the Insert start position (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*3*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For Pos, specify the number of characters (bytes) from the left end in the range of 0 to 1999 as the Insert position.
*5 T/C/CR/Z cannot be used.
*6 The leading character of the string specified by In is 0.

●Description of Operation

Inserts the stored string starting with ByStr to the position (bytes) specified by Pos of the stored strings starting with In, and stores the result in R after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **SINS** (strAllString, strInsert, INT#10);

Point

- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored.**
  - **When there is no end code (NUL (00H)) in the string specified by In/ByStr.**
  - **When the string including the end code (NUL (00H)) exceeds 2000 characters**
  - **When the string length of In＜Pos.**
  - **When the inserted string length exceeds 2000 characters including the end code (NUL (00H)).**
  - **When Bit data is specified for In/ByStr/R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In/ByStr/R by Indirect designation.**

INSERT Inserts String

String inserting
[R:=] INSERT (In1, In2, P);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1*5	Specifies the Insert target string storage destination, or the Insert target string.	—	—	—	—	—	—	—	.T	○*2
In2*1*5	Specifies the Insert string storage destination, or the Insert string.	—	—	—	—	—	—	—	.T	○*2
P*6	Specifies the Insert start position (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*1*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For **P**, specify the number of characters (bytes) from the left end in the range of 0 to 1999 as the Insert position.
*5 T/C/CR/Z cannot be used.
*6 The leading character of the string specified by In1 is 0.

●Description of Operation

This Function is similar to "SINS Function" except the Function name is different.
For details on operation, refer to "SINS function"(page 7-74).

SDEL Partially Deletes String

String deleting
[R:=] SDEL (In, Pos, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*6	Specifies the Delete target string storage destination, or the Delete target string.	—	—	—	—	—	—	—	.T	○*2
Pos*3*4	Specifies the Delete start position (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○
Count*3*5	Specifies the number of characters (bytes) to delete. (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○
R*1*6	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For **Pos**, specify the number of characters (bytes) from the left end in the range of 0 to 1999 as the Delete position.
*5 For **Count**, specify the number of characters (bytes) from **Pos** in the range of 0 to 1999.
*6 T/C/CR/Z cannot be used.

●Description of Operation

Deletes the number of characters (bytes) specified by **Count** from the position (bytes starting with 0) specified by **Pos** of the stored strings starting with **In**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **SDEL** (strAllString, INT#10, INT#20);

Point

- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored.**
  - **When there is no end code (NUL (00H)) in the string specified by In.**
  - **When the string including the end code (NUL (00H)) exceeds 2000 characters**
  - **When the string length of In＜Pos.**
  - **When the string stored after R exceeds the Device/Variable range.**
  - **When Bit data is specified for In/R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In/R by Indirect designation.**

DELETE Partially Deletes String

String deleting

[R:=] DELETE (In, L, P);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Delete target string storage destination, or the Delete target string.	—	—	—	—	—	—	—	.T	○*2
L*3*4	Specifies the number of characters (bytes) to delete. (1 to 1999)	.U	.U	.U	.U	—	—	—	—	○
P*3	Specifies the Delete start position (bytes). (1 to 1999)	.U	.U	.U	.U	—	—	—	—	○
R*1*5	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	○

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For L, specify the number of characters (bytes) from the Delete start position in the range of 0 to 1999.
*5 T/C/Z cannot be used.

●Description of Operation

Deletes the string stored in In by the number of characters (bytes) specified by L from the position (bytes starting with 1) specified by P, and stores the result in R.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **DELETE** (strAllString, INT#10, INT#20);

Point

- **When any of the following conditions is TRUE, the end code (NUL(00H)) is stored.**
  - **When there is no end code (NUL (00H)) in the string specified by In.**
  - **When the string including the end code (NUL (00H)) exceeds 2000 characters**
  - **When the string length of In＜L.**
  - **When the string stored after R exceeds the Device/Variable range.**
  - **When Bit data is specified for In/R and a position other than the channel leading is specified.**
  - **When Timer/Counter is specified for In/R by Indirect designation.**

TRIM Trims 1-byte Space/Tab at String End

Trimming

[R:=] TRIM (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the Trim target string storage destination, or the Trim target string.	–	–	–	–	–	–	–	.T	○*2
R*1*3	Return value that stores the Trimmed string. Omissible. If omitted, Conversion result is stored in Internal register.	–	–	–	–	–	–	–	.T	–

*1 For Bit data, specify the leading of the channel.

*2 To specify a string using Constant, enclose with " ".

*3 T/C/Z cannot be used.

●Description of Operation

Deletes all 1-byte spaces (20H) and tabs (09H) at both ends (start and end) of the stored string starting with **In**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)

For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **TRIM** (strAllString);

LTRIM Left-trims 1-byte Space/Tab

Left-trimming
[R:=] LTRIM (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the Trim target string storage destination, or the Trim target string.	—	—	—	—	—	—	—	.T	○*2
R*1*3	Return value that stores the Trimmed string. Omissible. If omitted, Conversion result is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 T/C/Z cannot be used.

●Description of Operation

Deletes 1-byte spaces (20H) and tabs (09H) at the left end (string leading) of the stored string starting with **In**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **LTRIM** (strAllString);

RTRIM Right-trims 1-byte Space/Tab

Right-trimming
[R:=] RTRIM (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the Trim target string storage destination, or the Trim target string.	—	—	—	—	—	—	—	.T	○*2
R*1*3	Return value that stores the Trimmed string. Omissible. If omitted, Conversion result is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 T/C/Z cannot be used.

●Description of Operation

Deletes 1-byte spaces (20H) and tabs (09H) at the right end (string leading) of the stored string starting with **In**, and stores the result in **R** after adding the code NUL (00H).

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **RTRIM** (strAllString);

STRIM Trims 1-byte Space/Tab from Specified String Position

Specified position trimming
[R:=] STRIM (In, Param);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the Trim target string storage destination, or the Trim target string.	—	—	—	—	—	—	—	.T	○*2
Param	Specifies the Trim target position. (1 to 3)	.U	.U	.U	.U	—	—	—	—	○
R*1*3	Return value that stores the Trimmed string. Omissible. If omitted, Conversion result is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 T/C/Z cannot be used.

●Description of Operation

Deletes all 1-byte spaces and tabs at the position specified by **Param** from the stored string starting with **In**, and stores the result in **R** after adding the code NUL (00H).
The content to be specified by the Trim position specify parameter is as follows.

Param value	Details
1	Deletes all 1-byte spaces and tabs on the left side of the string. (Same operation as LTRIM Function)
2	Deletes all 1-byte spaces and tabs on the right side of the string. (Same operation as RTRIM Function)
3	Deletes all 1-byte spaces and tabs on both sides of the string. (Same operation as TRIM Function)

Point **If a parameter other than the above is specified, In is stored in R without executing the Function.**

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strString := **STRIM** (strAllString, INT#3);

SPLIT Splits String

Splitting
[R:=] SPLIT (Src, Delimiter, InOut1, InOut2, InOutN);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Src*1	Specifies the Split source string storage destination, or the Split source string.	—	—	—	—	—	—	—	.T	○*2
Delimiter*1	Specifies the Delimiter string storage destination, or the Delimiter string.	—	—	—	—	—	—	—	.T	○*2
InOut1*1	Specifies the Split string 1 storage destination.	—	—	—	—	—	—	—	.T	—
InOut2*1	Specifies the Split string 2 storage destination.	—	—	—	—	—	—	—	.T	—
⋮	⋮	—	—	—	—	—	—	—	.T	—
InOutN*1*3	Specifies the Split string n storage destination.	—	—	—	—	—	—	—	.T	—
R*1*4*5	The Return value that stores the number of splits. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 The number of arguments n is specified in the range of 1 to 10.
*4 T/C/Z cannot be used.
*5 If a type other than .U is specified to R, type conversion is automatically performed during conversion.

●Description of Operation

Splits Src by Delimiter, and stores the result in the Split string InOut1 to InOutN.
The number of Split strings is stored in R.
When Src is '18,1A,35,E0,C3,EB' and Delimiter is ',' (comma)

Reference

- When the number of splits of R is 0, the Function is not executed.
- When 0＜the number of splits of R＜n, only the end code (NUL) is stored for the excessive argument.
- When the number of splits of R =n, InOutN may contain splits (can be further split)
  To determine if there are remained splits that can be further split, use the SFIND Function to check for delimiters in InOutN.
  If there are remained splits that can be further split, they can be split by specifying InOutN in Src and executing the SPLIT Function again.

「About the operation flags (CR2009 to CR2012)」(page 1-8)

●Expression sample

uiValue := **SPLIT** (strAllString, ',', strString1, strString2, strString3);

SFIND Searches String

String searching
[R:=] SFIND (In, Pattern, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*5	Specifies the Search target string storage destination, or the Search target string.	—	—	—	—	—	—	—	.T	○*2
Pattern*1*5	Specifies the Search string storage destination, or the Search string.	—	—	—	—	—	—	—	.T	○*2
Pos*3*4	Specifies the Search start position (bytes). (0 to 1999)	.U	.U	.U	.U	—	—	—	—	○*3
R*6	The Return value that stores the Convert result. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF*7	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For **Pos**, specify the number of characters (bytes) from the leading of **In** in the range of 0 to 1999 as the Search position.
*5 T/C/Z cannot be used.
*6 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.
*7 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

When Execution condition is TRUE, searches the stored Search string starting with **Pattern** from the position specified by **Pos** of the stored Search target string starting with **In**.
As the search result, the first character position (number of bytes from **In**) of the Search string found for the first time after **Pos** is stored in **R**.
If the string length of **Pattern** is 0 and the specified string is not found, or the string length of **Pattern** is 0, 16# (HEX specifying) FFFF is stored in **R**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **SFIND** (strAllString, strString, INT#8);

Point

When any of the following conditions is TRUE, 0 is stored in R.

- **When there is no end code (NUL (00H)) in the string specified by In/Pattern.**
- **When the string including the end code (NUL (00H)) exceeds 2000 characters**
- **When (the string length of In) < (Pos + the string length of Pattern) (the end code (NUL (00H)) is not included in the string length).**
- **When Bit data is specified for In/Pattern and a position other than the channel leading is specified.**
- **When Timer/Counter is specified for In/Pattern by Indirect designation.**

SFINDN Searches String

String searching N
[R:=] SFINDN (In, Pattern, InLen, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*3	Specifies the Search target string storage destination, or the Search target string.	–	–	–	–	–	–	–	.T	○*2
Pattern*1*3	Specifies the Search string storage destination, or the Search string.	–	–	–	–	–	–	–	.T	○*2
InLen	Specifies the Search target string length (bytes).	.U	.U	.U	.U	–	–	–	–	○
Pos	Specifies the Search start byte position.	.U	.U	.U	.U	–	–	–	–	○
R*4	The Return value that stores the Convert result. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	–	.T	–

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 T/C/Z cannot be used.
*4 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

When Execution condition is TRUE, searches the stored string starting with **Pattern** from **Pos** of the number of Search target bytes starting with **In**.
In **R**, the start position (number of bytes) of the string specified by **Pattern** that is found for the first time after the position specified by **Pos** in the character string specified by **InLen**, is stored.
If the string specified by **Pattern** is not found, or the string length of **Pattern** is 0.
16# (HEX specifying) FFFF is stored in **R**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **SFINDN** (strAllString, strString, INT#50 , INT#4);

INSTR Searches String

String searching N
[R:=] INSTR (Pos, In, Pattern);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Pos*4	Specifies the Search start position (bytes). (0 to 1999)	.U	.U	.U	.U	–	–	–	–	○*3
In*1*5	Specifies the Search target string storage destination, or the Search target string.	–	–	–	–	–	–	–	.T	○*2
Pattern*1*5	Specifies the Search string storage destination, or the Search string.	–	–	–	–	–	–	–	.T	○*2
R*6	Return value that stores Operation result. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF*7	–	–	–

*1 For Bit data, specify the leading of the channel.

*2 To specify a string using Constant, enclose with " ".
*3 16# (HEX specifying) cannot be used.
*4 For **Pos**, specify the number of characters (bytes) from the leading of **In** in the range of 0 to 1999 as the Search position.
*5 T/C/Z cannot be used.
*6 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.
*7 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

When Execution condition is TRUE, searches the stored Search string starting with **Pattern** from the position specified by **Pos** of the stored Search target string starting with **In**.
As the search result, the first character position (number of bytes of **In**) of the Search string found for the first time after **Pos** is stored in **R**.
If the string specified by **Pattern** is not found, or the string length of **Pattern** is 0.
16# (HEX specifying) FFFF is stored in **R**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **INSTR** (INT#8, strAllString, strString);

Point

When any of the following conditions is TRUE, 0 is stored in R.

- **When there is no end code (NUL (00H)) in the string specified by In/Pattern.**
- **When the string including the end code (NUL (00H)) exceeds 2000 characters**
- **When (the string length of In) < (Pos + the string length of Pattern) (the end code (NUL (00H)) is not included in the string length).**
- **When Bit data is specified for In/Pattern and a position other than the channel leading is specified.**
- **When Timer/Counter is specified for In/Pattern by Indirect designation.**

FIND Finds String

String finding
[R:=] FIND (In1, In2);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1*3	Specifies the Search target string storage destination, or the Search target string.	—	—	—	—	—	—	—	.T	○*2
In2*1*3	Specifies the Search string storage destination, or the Search string.	—	—	—	—	—	—	—	.T	○*2
R*4	Return value that stores Operation result. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " " (up to 31 1-byte characters).
*3 T/C/Z cannot be used.
*4 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.

●Description of Operation

From the strings stored in **In1**, in order from the beginning, finds the string that matches the string specified in **In2**, and stores the start position of the matches (number of bytes in the original string starting from 1) in **R**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **FIND**(strString, strFind);

Point

When any of the following conditions is TRUE, 0 is stored in R.

- **When there is no end code (NUL (00H)) in the string specified by In1/In2.**
- **When the string including the end code (NUL (00H)) exceeds 2000 characters**
- **When (the string length of In1)＜(the string length of In2).**
- **When Bit data is specified for In1/In2 and a position other than the channel leading is specified.**
- **When Timer/Counter is specified for In1/In2 by Indirect designation.**
- **If the string length of In2 is 0, or the specified string is not found, 65535 is stored in R.**

DISS Disperses String

String dispersing
DISS ([EN], Src, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1*3	Specifies the Disperse target string storage destination, or the Disperse target string.	—	—	—	—	—	—	—	.T	○*2
Dst*1*4	Specifies the Dispersed string (in words) storage destination.	.U	.U	.U	.U	—	—	—	—	
R	None	—	—	—	—	—	—	—	—	

*1 For Bit data, specify the leading of the channel.
*2 To specify a string using Constant, enclose with " ".
*3 T/C/CR/Z cannot be used.
*4 T/C/Z cannot be used.

●Description of Operation

When Execution condition is TRUE, disperse and stores the stored strings (in bytes) starting with **Src** to the strings (in words) starting with **Dst**.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

DISS (bEnable , strString, auiData[0]);

Point

When any of the following conditions is TRUE, 0 is stored in R.

- **When there is no end code (NUL (00H)) in the string specified by Src.**
- **When the string including the end code (NUL (00H)) exceeds 2000 characters**
- **When Bit data is specified for Src/Dst and a position other than the channel leading is specified.**
- **When Timer/Counter is specified for Src/Dst by Indirect designation.**

UNIS Unites Byte String

String uniting
UNIS ([EN], Src, Dst, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Src*1*4	Specifies the Unite target string (in words) storage destination, or the Unite target string (in words).	.U	.U	.U	.U	—	—	—	—	—
Dst*1*4	Specifies the United string storage destination.	—	—	—	—	—	—	—	.T	—
Count*1*4	Specifies the number of characters (bytes (in words) to unite. (1 to 1999)	.U	.U	.U	.U	—	—	—	—	○*2
R	None	—	—	—	—	—	—	—	—	—

*1 For Bit data, specify the leading of the channel.
*2 16# (HEX specifying) cannot be used.
*3 For Count, specify the number of characters (in words) to be united in the range of 1 to 1999.
*4 T/C/Z cannot be used.
*5 T/C/CR/Z cannot be used.

●Description of Operation

When Execution condition is TRUE, unites the stored string (in words) starting with Src by the number of characters (words) specified by Count, and stores with Dst leading and end code NUL (00H) added.

「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

UNIS (bEnable, auiData[0], strString, uiValue);

Point

The Function will not be executed if any of the following conditions are TURE:

- **When the value of Count is 0 or exceeds 2000.**
- **When the string including the end code (NUL (00H)) exceeds 2000 characters**
- **When Bit data is specified for Src/Dst and a position other than the channel leading is specified.**
- **When Timer/Counter is specified for Src/Dst by Indirect designation.**

CHR Converts HEX ASCII Code to Character

Character converting
[R:=] CHR (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Convert target ASCII code data, or its Storage destination. (16-bit data)	.U	.U	.U	.U	—	—	—	—	○*1
R*2	Return value that stores Operation result. Has .T identification type. Omissible. If omitted, Conversion result is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 1The Constant value is within the range 0 to 255. Specify the target, and when the value exceeds the range of 0 to 255, use the value with the upper 8 bits rounded.

*2 T/C/CR/Z cannot be used.

●Description of Operation

Converts the the ASCII code data stored in **In** to 1-byte character string with end code NUL (00H) added, and stores the result in **R**.

●Expression sample

strString := **CHR** (uiValue);

CPSASC Converts CIP String

CIP string converting
[**R**:=] **CPSASC** (**In**);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the leading of the Storage destination of the unconverted CIP string data.	.U	.S	.D	.L	.F	.DF	.B	.T	—
R[*1]	The Return value that stores the Convert result. Has .T identification type. Omissible. If omitted, Conversion result is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 1When specifying Bit data, specify the leading of the channel.

●Description of Operation

Converts the stored CIP string data starting with **In** to a string, and stores the result in **R**.
CPSASC Function is used for converting CIP string data and processing it as String.

For CIP data type of , refer to "KV EtherNet/IP Function User's Manual".
When the CPSASC function is executed, the CIP string data is converted and stored in Return value.
For details, refer to "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

strValue := **CPSASC**(auiValue[0]);

RCPSASC Revers-converts CIP String

CIP string revers-converting
[**R**:=] **RCPSASC** (**Src**, **Dst**);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Src	Specifies the leading of the Storage destination of the unconverted string.	—	—	—	—	—	—	—	.T	○[*2]
Dst	Specifies the leading of the Storage destination of the converted CIP string data.	.U	.S	.D	.L	.F	.DF	.B	.T	—
R[*1]	The Return value that stores the number of characters (in bytes) stored in **Dst**. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.
*2 To specify a string using Constant, enclose with " ".

●Description of Operation

Converts the string stored in **Src** to a CIP string data, and stores the result in **Dst**. The number of characters (in bytes) stored in **Dst** after conversion is stored in **R**.
CPSASC Function is used for converting String to CIP string data.
For CIP data type of , refer to 📖 "KV EtherNet/IP Function User's Manual".
When the CPSASC function is executed, the String is converted to CIP string data.

For details, refer to 📖 "Instruction Reference Manual" and "Character Processing Instructions".

●Expression sample

uiValue := **RCPSASC** (strValue, auiDst[0]);

CONCAT Concatenates String

String concatenating
[R:=] CONCAT (In1, In2, … ,InN);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1	Specifies the Concatenate target string.	—	—	—	—	—	—	—	.T	○
In2	Specifies the Concatenate target string.	—	—	—	—	—	—	—	.T	○
:	:									
InN	Specifies the Concatenate target string.	—	—	—	—	—	—	—	.T	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	—	.T	—

*1 If a type different from Operation target is specified to **R** Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Sequentially stores the Concatenate result stored from **In1** to **InN** in **R**.
N can be specified from 2 to 10.

●Expression sample

strAllString := **CONCAT** (strString1, strString2, strString3);

📖「About the operation flags (CR2009 to CR2012)」(page 1-8)
For details, refer to 📖 "Instruction Reference Manual" and "String Processing Instructions".

8

Extension Functions

Data Process Function 8-2
Table Process Function 8-17
Time Process Function. 8-22
High-speed Process Function 8-25
PID Function 8-28
Log Function 8-29
Storage Function. 8-32
Access Window Function. 8-45
FREE RUN Counter Function. 8-47
Sensor Setting Function. 8-48
Cyclic Communication Refresh Function. 8-52

Data Process Function

SEG Decodes 4-bit Data to 7-segment

7-segment decoding
[R:=] SEG (In, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the Decode target data, or its Storage destination.	.U	.U	.D	.D	–	–	–	–	–
Pos*2	Specifies the Decode digit position, or its Storage destination. (0 to 3 for .U, 0 to 7 for .D)	.U	.U	.U	.U	–	–	–	–	–
R*3	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*4	–	–	–

*1 When Bit data is specified as the Decode target, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

*2 When the target is specified in the **Pos** target digit position and the stored value exceeds 0 to 3 or 0 to 7:
If the suffix of the Decode target is .U: the value of the lower 2 bits is used.
If the suffix of the Decode target is .D: the value of the lower 3 bits is used.
For example, if the suffix of the Decode target **In** is .U, DM01000 is specified to **Pos** and the value is 23 (binary: 0000 0000 0001 0111), the value of the lower 2 bits is 3, so 3 is used as the value of **Pos**.

*3 SEG Function uses .U type. If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.

*4 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

From the 16-bit/32-bit data stored in **In**, decodes the 4 bits (1 digit) of the number of digits specified by **Pos** into 7-segment display data (8 bits), and stores the result in **R**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

- **In: When the Decode target is 16 bits (.U/.S)**
  uiValue := **SEG** (uiData, INT#2);

- **In: When the Decode target is 32 bits (.D/.L)**
  uiValue := **SEG** (udiData, INT#5);

BCNT Counts Target ON Bit

Bit counting
[R:=] BCNT (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In	Specifies the Operation target or Operation target data.	.U	.U	.D	.D	—	—	—	—	—
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	—

*1 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Counts the number of ON bits of the (16-bit/32-bit) BIN data stored in In, and stores the result in Return value.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **BCNT** (uiData);

DCNT Counts Detect Data in Specified Block

Data counting
[R:=] DCNT (Pattern, In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Pattern*1	Specifies the Search value storage destination, or the Search value.	.U	.U	.D	.D	.D	—	—	—	○
In*1	Specifies the leading of the Search data block.	Types same as Pattern.					—	—	—	—
Count	Specifies the Storage destination for Search block data amount, or the Search data amount.	.U	.U	.U	.U	—	—	—	—	○
R*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Compares the block data amount specified by Count (starting with In) with the data specified by Pattern, and stores the matched data amount in the Return value R.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **DCNT** (uiData , auiValue[0], uiCount);

Point **If the value of Detect data amount of Count is 0, the value of Detect target of In is stored in the Return value R.**

SER Searches Data in Specified Block

Searching
[R:=] SER (Pattern, In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Pattern*1	Specifies the Search value storage destination, or the Search value.	.U	.U	.D	.D	.D	–	–	–	○
In*1	Specifies the leading of the Search data block.	Types same as Pattern.					–	–	–	–
Count	Specifies the Storage destination for Search block data amount, or the Search data amount.	.U	.U	.U	.U	–	–	–	–	○
R*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	–	–	–

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Compares the block data amount specified by Count (starting with In) with the data specified by Pattern, stores the Device No. where the matched data is stored in R.
If multiple matched data are found, the smallest Device No. is stored.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **SER** (uiData, auiValue[0], INT#40);

Point
- **If no data is found or the specified Search data amount of Count is 0, the value of Search target of Pattern is stored in the Return value R.**
- **If Bit data is specified starting with the Search range, the value converted to DEC based on R (MR, LR) 0000 is stored in the Search result (Return value) instead of 16-bit units.**
  (Example) When the Search result is MR82312: 82312 → 823 × 16 + 12 = 13180 is stored.

DSER Searches Data in Specified Range

Data searching
[R:=] DSER (Pattern, Start, Count, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Pattern*1	Specifies the Search data, or its Storage destination.	.U	.U	.D	.D	.D	—	—	—	○
Start*1	Specifies the leading of Search range.	Types same as Pattern.					—	—	—	—
Count*1	Specifies the Search target amount, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Result*1	Specifies the Search result (Search amount) storage destination.	.U	.U	.U	.U	—	—	—	—	—
R*2	Return value that stores Operation result. Specifies the Storage destination for the Detect position (offset from the Detect range leading). Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*2 If a type other than .U is specified to R, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Searches the Word data range of Count starting from Start by using BIN data stored in Pattern as the keyword.

- The Word data amount that matches the Pattern is stored in Result.
- Stores the position of the first matched word data (position relative to the Pattern) in R.
- If no matched data is found, 0 is stored in Result and R.

●Expression sample

uiValue := **DSER** (uiData, auiValue[0], uiCount, uiResult);

MAX Maximum

Maximum
[R:=] MAX (In1, In2, …, InN);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*3*4	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	—	—	○
In2*3*4	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	—	—	○
:	:									
InN*3*4	Specifies the Select target. (The maximum value of N is 10)	.U	.S	.D	.L	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	-

*1 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.
*3 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
*4 If Data type is .F or .DF, T/C cannot be specified.

●Description of Operation

Searches the largest result from **In1** to **InN** and stores the maximum value in **R**. N can be specified to 2 to 10.

●Expression sample

rValue := **MAX** (rValue1, rValue2, rValue3);

MAX_RANGE Searches Max. Value in Specified Block

Max. value searching

[R:=] MAX_RANGE (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In[*1*2*3]	Specifies the leading of Search range.	.U	.S	.D	.L	.F	.DF[*5]	—	—	—
Count[*1]	Specifies the Storage destination for the data amount in Detect range , or the data amount in Detect range.	.U	.U	.U	.U	—	—	—	—	○
R[*4]	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF[*5]	—	—	—

*1 The range of numerical values that can be processed differs depending on the suffix.
*2 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
*3 If the type of **In** is .F or .DF, T/C cannot be specified.
*4 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Starting with **In**, searches for the Max. value in BIN data (16/32/64 bits) stored in the Word amount range specified by **Count**. (When the Search target is 32 bits data, the Search range is the specified data amount x 2 words, and 64 bits data, the specified data amount x 4 words)
As the Search result, the Max. value is stored in **R**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **MAX_RANGE** (auiValue[0], INT#32);

Point **When the value of Count is 0, 0 is stored in R.**

MIN Minimum

Minimum
[R:=] MIN (In1, In2, …, InN);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*3*4	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	—	—	○
In2*3*4	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	—	—	○
:	:									
InN*3*4	Specifies the Select target. (The maximum value of N is 10)	.U	.S	.D	.L	.F	.DF*2	—	—	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	—	—	-

*1 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.
*3 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
*4 If Data type is .F or .DF, T/C cannot be specified.

●Description of Operation

Searches the largest result from **In1** to **InN** and stores the Min. value in **R**. N can be specified to 2 to 10.

●Expression sample

rValue := **MIN** (rValue1, rValue2, rValue3);

MIN_RANGE Searches Min. Value in Specified Block

Min. value searching
[R:=] MIN_RANGE (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2*3	Specifies the leading of Search range.	.U	.U	.D	.D	.F	.DF*5	—	—	—
Count*1	Specifies the Storage destination for the data amount in Detect range , or the data amount in Detect range.	.U	.U	.U	.U	—	—	—	—	○
R*4	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 The range of numerical values that can be processed differs depending on the suffix.
*2 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
*3 If the type of **In** is .F or .DF, T/C cannot be specified.
*4 If a type different from Operation target is specified to **R**, Type conversion is automatically executed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Starting with **In**, searches for the Min. value in BIN data (16/32/64 bits) stored in the Word amount range specified by **Count**. As the Search result of (when the Search target is 32 bits data, the Search range is the specified data amount x 2 words, and 64 bits data, the specified data amount x 4 words), the Min. value is stored in **R**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)

For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **MIN_RANGE** (auiValue[0], INT#32);

Point **When the value of Count is 0, 0 is stored in R.**

SEL Select

Select

[R:=] SEL (G, In0, In1);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
G	Specifies the Gate.	–	–	–	–	–	–	.B	–	○
In0	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	.B	.T	○
In1	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	.B	.T	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	.B	.T	–

*1 If a type different from In0/In1 is specified to R, Type conversion is automatically executed during conversion.

*2 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

If G is FALSE, stores In0 in R, and if G is TRUE, In1.

●Expression sample

rValue := **SEL** (bFlag, rValue1, rValue2);

MUX Multiplexer

Multiplexer

[R:=] MUX (K, In0, In1, … ,InN);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
K	Specifies the selector. (0 to N)	.U	.U	.U	.U	–	–	–	–	○
In0	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	.B	.T	○
In1	Specifies the Select target.	.U	.S	.D	.L	.F	.DF*2	.B	.T	○
:	:									
InN	Specifies the Select target. (The maximum value of N is 9)	.U	.S	.D	.L	.F	.DF*2	.B	.T	○
R*1	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*2	.B	.T	–

*1 If a type different from In0/In1/…/InN is specified to R, Type conversion is automatically executed during conversion.
*2 If Device Z is specified, the ".DF" suffix cannot be specified.
*3 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
*4 If the type of InN is .F or .DF, T/C cannot be specified.

●Description of Operation

Stores the value of InN in R according to the value (k) specified by K.

●Expression sample

rValue := **MUX** (udiSelector, rValue1, rValue2, rValue3);

Point **When a value other than 1 to N (N is 2 to 10) is specified for K. R is an indefinite value.**

AVG Average Value of Specified Block

Averaging
[R:=] AVG (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2*3	Specifies the leading of Average range.	.U	.U	.D	.D	.F	.DF*5	—	—	—
Count*1	Specifies the data amount in Average range, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
R*4	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 The range of numerical values that can be processed differs depending on the suffix.
*2 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
*3 If the type of In is .F or .DF, T/C cannot be specified.
*4 If a type different from Operation target is specified to R, Type conversion is automatically executed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Starting with In, calculates the Average value of all BIN data (16 bits/32 bits (unsigned/signed)/64 bits) stored in the Word amount range specified by Count. (When the Operation target is 32 bits data, the Operation range is the specified data amount x 2 words, and 64 bits data, the specified data amount x 4 words)
As the Operation result, the Average value is stored in R.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **AVG** (auiValue[0], ACNT(auiValue, INT#1));

Point **When the value of Count is 0, 0 is stored in R.**

WSUM Sum of Specified Block

Word sum
[R:=] WSUM (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2*3	Specifies the leading of Sum range.	.U	.U	.D	.D	.F	.DF*5	—	—	—
Count*1	Specifies the Storage destination for the data amount in Sum range , or the data amount in Sum range.	.U	.U	.U	.U	—	—	—	—	○
R*4	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 The range of numerical values that can be processed differs depending on the suffix.
*2 When Bit data is specified, operation is performed with consecutive 16/32/64 bits. When a position other than the channel leading such as R002 is specified, 16/32/64 bits will be processed across the next channel.
*3 If the data type of In is .F or .DF, T/C cannot be specified.
*4 If the data type of In is .U/.S/.D/.L, and a type other than .L is specified to R, type conversion is automatically performed during conversion. If the data type of In is .F, and a type other than .F is specified to R, type conversion is automatically performed during conversion. If the data type of In is .DF, and a type other than .DF is specified to R, type conversion is automatically performed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Starting with In, calculates the Sum value of all BIN data (16 bits/32 bits (unsigned)/64 bits) stored in the Word amount range specified by Count. As the Operation result of (when the Operation target is 32 bits data, the Operation range is the specified data amount x 2 words, and 64 bits data, the specified data amount x 4 words), the Sum value is stored in R.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **WSUM** (auiValue[0], INT#24);

Reference
If the data type of In is .U/.S/.D/.L and the Operation result exceeds the range of signed 32-bit data (-2147483648 to +2147483647), the lower 32-bit data is stored in Return value, and the upper 32-bit data is stored in TM001 and TM000.

Point
When the value of Count is 0, 0 is stored in R.

BSUM Sum (in bytes) of Specified Block

Byte sum
[R:=] BSUM (In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2*3	Specifies the leading of Sum range.	.U	.U	.U	.U	—	—	—	.U	—
Count*1	Specifies the data amount (in bytes) to sum.	.U	.U	.U	.U	—	—	—	—	○
R*4	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*5	—	—	—

*1 The range of numerical values that can be processed differs depending on the suffix.
*2 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.
*3 T/C/Z cannot be used.
*4 If a type other than .D is specified to **R**, type conversion is automatically performed during conversion.
*5 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Calculates the sum of 8 bits for upper or lower BIN data stored in the specified range starting from **In**, and stores it in **R**. If **Count** is odd, the upper 8 bits of the last word data are calculated. The operation is performed in the order of upper to lower, and when **Count** is odd, the lower 8 bits of the last word data are ignored.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

uiValue := **BSUM** (auiValue[0], uiCount);

Reference: If the Operation result exceeds the range of signed 32-bit data, the lower 32-bit data is stored in Return value, and the upper 32-bit data is stored in TM1 and TM0.

Point: **When any of the following conditions is TRUE, 0 is stored in R.**
- When Timer counter is specified for **In**.
- When Bit data is specified for **In** and a position other than the channel leading is specified.
- When the value of **Count** is 0.

CRC Calculates CRC Value in Specified Block

CRC calculating
[R:=] CRC (Init, CalcType, CacOrder, BitOrder, In, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Init[*4]	Specifies the Operation mode.[*1]	.U	.U	.U	.U	–	–	–	–	○
CalcType[*4]		.U	.U	.U	.U	–	–	–	–	○
CalcOrder[*4]		.U	.U	.U	.U	–	–	–	–	○
BitOrder[*4]		.U	.U	.U	.U	–	–	–	–	○
In[*2*4]	Specifies the leading of the data block to be calculated.	.U	.U	.U	.U	–	–	–	–	–
Count[*3]	Specifies the data amount (1 to 65535).	.U	.U	.U	.U	–	–	–	–	○
R[*4]	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF[*6]	–	–	–

*1 Uses 1 if the specified target or expression contains a value other than 0 or 1.
*2 When specifying Bit data, specify the leading of the channel.
*3 When **CalcOrder** is "0", specify the number in 1-word (16-bit) units.
When **CalcOrder** is "1 (other than 0)", specify the number in 1-byte (8-bit) units.
*4 T/C/Z cannot be used.
*5 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.
*6 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

In the Operation mode specified by **Init** to **BitOrder**, calculates the CRC value of **Count** data starting from **In**, and stores it in **R**.

●Expression sample

uiValue := **CRC** (INT#1, INT#0, INT#0, INT#0, auiValue[0], uiCount);

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

Point

When any of the following conditions is TRUE, the value of Init is stored in R.

- When Word data or Timer/Counter is specified for **Init**.
- When Timer/Counter or Bit data is specified for **CalcType** by Indirect designation, or a position other than the channel leading is specified.
- When the value of **Count** is 0.

ZRES Resets Specified Block (Stores 0)

Zone Resetting
ZRES ([**EN**], **Start**, **Last**);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Start	Specifies the leading of the Reset (0 string) range.	.U	.U	.U	.U	.U	.U*1	.B	.T	—
Last	Specifies the end of the Reset (0 string) range.	Types same as **Start**.								—
R	None	—	—	—	—	—	—	—	—	—

*1 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

When Execution condition is TRUE, resets (stores 0 for) all in the range specified by **Start** and **Last** (stores 0).
*Even if .D .L .F .DF is specified as the suffix, all will be treated as .U, so 0 is stored only for the specified 1 word as the last word.

Reference When Word data is specified, it is reset to 0 in word units, and when Bit data is specified, to OFF in bit units.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

ZRES (bEnable, auiValue[0], auiValue[10]);

Point

If Device types specified for Start and Last are different, ZRES Function will not be executed.

BCMP Compares Data Between Specified Blocks

Data comparing between blocks
[R:=] BCMP (In1, In2, InOut, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1	Specifies the leading of the Compare source data block.	.U	.U	.D	.D	.D	—	—	—	—
In2*1	Specifies the leading of the Compare source data block.	Types same as In 1.					—	—	—	—
InOut*1*2	Specifies the leading to store the Compare result.	—	—	—	—	—	—	.B	—	—
Count	Specifies the Storage destination for Compare block data amount, or the Compare data amount.	.U	.U	.U	.U	—	—	—	—	○
R	The Return value that stores Bit info. ON when all data matched. Otherwise, OFF Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

*2 If Bit data is specified, operates correctly across channels.
If Word data is specified for **InOut**, the Result storage destination is the lowest bit of 16-bit word data specified by **Count**. No change except for the lowest bit.

●Description of Operation

When Execution condition is TRUE, compares the block data (starting with **In1**) which amount is specified by **Count** with the block data (starting with **In2**) of **Count**.
On if Compare result matched, and OFF if unmatched, and is stored in the Bit data for the Compare data amount starting with **InOut**.
When **InOut** is Word data with .B specified, the Compare result is stored in 16-bit word data units, and only the lowest bit of each word data is the Compare target. (No change except for the lowest bit)
As the Compare result, turns **R** on when all data matched. Otherwise, off.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

bFlag := **BCMP** (auiValue[0], auiData[0], abFlag[0], uiCount);

Point **When the value of Count is 0, R turns on.**

BCMPI Compares Data Between Block and Value

Data comparing between blocks
[R:=] BCMPI (In1, In2, InOut, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In1*1	Specifies the leading of the Compare data block.	.U	.U	.D	.D	.D	—	—	—	—
In2*1	Specifies the Compare value storage destination, or the Compare value.	Types same as In 1.					—	—	—	—
InOut*1*2	Specifies the leading to store the Compare result.	—	—	—	—	—	—	.B	—	—
Count	Specifies the Storage destination for Compare block data amount, or the Compare data amount.	.U	.U	.U	.U	—	—	—	—	○
R	Return value that stores Operation result. Omissible. ON when all data matched. Otherwise, OFF If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

*2 If Bit data is specified, operates correctly across channels.
If Word data is specified for **InOut**, the Result storage destination is the lowest bit of 16-bit word data specified by **Count**. No change except for the lowest bit.

●Description of Operation

When Execution condition is TRUE, compares the block data (starting with **In1**) which amount is specified by **Count** with the block data specified by **In2**.
As the Compare result, On if matched, and OFF if unmatched, and is stored in the target for the data amount of **Count** starting with **InOut**.
When **InOut** is Word data, the Compare result is stored in 16-bit word data units, and only the lowest bit of each word data is the Compare target. (No change except for the lowest bit)
As the Compare result, turns **R** on when all data matched. Otherwise, off.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

bFlag := **BCMPI** (auiValue[0], uiValue, abFlag[0], uiCount);

Point **When the value of Count is 0, R turns on.**

RND Generates Random Number

Random
[R:=] RND (In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1	Specifies the Random number generate source (Operation target or Operation target data).	.U	.U	.U	.U	—	—	—	—	—
R*2	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF*3	—	—	—

*1 Even if Convert target other than the Bit data channel leading such as R1002 is specified to **In**, operates correctly across the next channel.

*2 If a type other than .U is specified to **R**, type conversion is automatically performed during conversion.
*3 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Generates a Random number from **In** (Operation target) and stores it in **R**.
The generated random numbers are pseudo-random numbers from 1 to 65535. If the value of **In** is 0, 1 is stored in **R**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Arithmetic/Comparison Instructions".

Point
- **"M Series" is used for the Random number generate algorithm.**
- **The value generated by RND Function is cyclic (for example, 2 is always generated after 1). If the value of the random number cause specified in In does not change, the value of the generated random number also unchanges.**

●Expression sample

uiValue1 := **RND** (uiValue2);

SORT Sorts BIN Data

Sorting
SORT ([**EN**], **Order**, **InOut**, **Count**);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	○
Order	Specifies the Sort order. (0: ascending, 1: descending)	.U	.U	.U	.U	—	—	—	—	○
InOut[*1]	Specifies the leading of Search byte string.	.U	.S	.D	.L	—	—	—	—	—
Count[*2]	Specifies the Sort target data amount.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 If Bit data is specified, the **Count** channels are occupied when the type of **InOut** is .U/.S,
and 2 x **Count** channels, when .D/.L.
If Word data is specified, the **Count** words are occupied when the type of **InOut** is .U/.S, and 2 x **Count** words , when .D/.L.
*2 When Bit data is specified, occupies consecutive 16 bits. When Word data is specified, occupies 1 word.

●Description of Operation

Sorts the BIN data of **Count** from the data table specified by **InOut** in the sort order specified by **Order**.

Point
The SORT Function sorts all data during execution. Therefore, specifying a large value for Order can significantly increase the Scan time.
To adjust the effect on Scan time, use the SORTN Function to divide into multiple scans to sort.
"SORTN function", page 8-16

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

SORT (bEnable, INT#0 , auiValue[0], uiCount);

SORTN Divide-sorts BIN Data

Divide-sorting
[R:=] SORTN (En, Order, CmpCount, InOut, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
En	Executes the sort process when Execution condition is ON.	—	—	—	—	—	—	.B	—	—
Order	Specifies the Sort order. (0: ascending, 1: descending)	.U	.U	.U	.U	—	—	—	—	○
CmpCount*2	Specifies the number of comparisons in one scan. (1 to 65535)	.U	.U	.U	.U	—	—	—	—	○
InOut*1	Specifies the leading of Search byte string.	.U	.S	.D	.L	—	—	—	—	—
Count*2	Specifies the Sort target data amount.	.U	.U	.U	.U	—	—	—	—	○
R	The Return value that stores the Search result. Omissible. OFF during sorting, ON after completed. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 If Word data is specified, the **Count** channels are occupied when the type of **InOut** is .U/.S, and 2 x **Count** channels , when .D/.L.
If Word data is specified, the **Count** words are occupied when the type of **InOut** is .U/.S, and 2 x **Count** words , when .D/.L.

*2 When Bit data is specified, occupies consecutive 16 bits. When Word data is specified, occupies 1 word.

●Description of Operation

Sorts the BIN data of **Count** from the data table specified by **InOu**t in the sort order specified by **Order**.
The number of comparisons per scan is specified by the number of comparisons of **CmpCount**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Data Processing Instructions".

●Expression sample

bFlag := **SORTN** (bEnable, INT#0, uiValue, auiValue[0], uiCount);

Table Process Function

FIFOW Writes to FIFO Table

FIFO writing
FIFOW ([EN], In, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the Storage destination for the Write data in FIFO block.	Types same as Dst.					—	—	—	—
Dst	Specifies the leading of FIFO data table.	.U	.U	.D	.D	.D	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, writes the data (16/32 bits) stored in **In** to the end of the data block specified by **Dst**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Table Processing Instructions".

●Expression sample

FIFOW (bEnable, uiValue, auiValue[0]);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When the Max. storage data amount is 0.
- When the data amount n exceeds the Max. storage data amount "max".
- In the case of Max. storage data amount "max" that causes the data table to exceed Device/Variable range.

Reference

For the 32-bit data table, Single precision floating point real data can also be used as the Write data.

FIFOR Reads from FIFO Table

FIFO reading
FIFOR ([EN], In, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the leading of FIFO data table.	.U	.U	.D	.D	.D	—	—	—	—
Dst*1	Specifies the Storage destination for the Read data from FIFO block.	Types same as In.					—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, reads the leading data stored in the data block specified by **In**, and stores in **Dst**.
After the data is stored, the contents of the data block are moved forward one by one, and the stored data amount is decremented by 1.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Table Processing Instructions".

●Expression sample

FIFOR (bEnable, auiValue[0] , uiValue);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When there is no data in the data table.
- When the data amount n exceeds the Max. storage data amount "max".
- In the case of Max. storage data amount "max" that causes the data table to exceed Device/Variable range.

LIFOW Writes to LIFO Table

LIFO writing
LIFOW ([EN], In, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the Storage destination for the Write data in LIFO block.	Types same as Dst.					—	—	—	—
Dst	Specifies the leading of LIFO data table.	.U	.U	.D	.D	.D	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, writes the data (16/32 bits) stored in **In** to the end of the data block specified by **Dst**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Table Processing Instructions".

●Expression sample

LIFOW (bEnable, uiValue, auiValue[0]);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When the Max. storage data amount is 0.
- When the data amount n exceeds the Max. storage data amount "max".
- In the case of Max. storage data amount "max" that causes the data table to exceed Device/Variable range.

LIFOR Reads from LIFO Table

LIFO reading
LIFOR ([EN], In, Dst);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the leading of LIFO data table.	.U	.U	.D	.D	.D	—	—	—	—
Dst	Specifies the Storage destination for the Read data from LIFO block.	Types same as In.					—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, reads the final data stored in the data block specified by **In**, and stores in **Dst**.
After the data is stored, the contents of the data block are moved forward one by one, and the stored data amount is decremented by 1.
(0 is stored in the final data that was read)

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Table Processing Instructions".

●Expression sample

LIFOR (bEnable, auiValue[0], uiValue);

Point

The Function will not be executed if any of the following conditions is TRUE:
- When there is no data in the data table.
- When the data amount n exceeds the Max. storage data amount "max".
- In the case of Max. storage data amount "max" that causes the data table to exceed Device/Variable range.

FWRIT Overwrites Specified Position in Data Table

Data table overwriting
FWRIT ([EN], In, Dst, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the Overwrite data, or its storage destination.	Types same as Dst.					—	—	—	—
Dst*1	Specifies the leading of Table.	.U	.U	.D	.D	.D	—	—	—	—
Pos	Specifies the Overwrite data block position, or its storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, overwrites the data (16/32 bits) stored in **In** at the position (No.) specified by **Pos** in the data block specified by **Dst**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Table Processing Instructions".

●Expression sample

FWRIT (bEnable, uiValue , auiValue[0] , uiCount);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When the value of **Pos** is 0 or exceeds data amount N.
- When the Max. storage data amount "max' is 0.
- When the data amount N exceeds the Max. storage data amount "max".
- When the Max. storage data amount "max" exceeds Device/Variable range.

FINS Inserts to Specified Position in Data Table

Data table inserting
FINS ([EN], In, Dst, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the Overwrite data, or its storage destination.	Types same as Dst.					—	—	—	—
Dst*1	Specifies the leading of Table.	.U	.U	.D	.D	.D	—	—	—	—
Pos	Specifies the Insert position in data block, or its storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, inserts the data (16/32 bits) stored in **In** to the position (No.) specified by **Pos** in the data block specified by **Dst**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Table Processing Instructions".

●Expression sample

FINS (bEnable, uiValue , auiValue[0] , uiCount);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When the value of **Pos** is 0 or exceeds data amount N.
- When the Max. storage data amount "max' is 0.
- When the data amount N exceeds the Max. storage data amount "max".
- When the Max. storage data amount "max" exceeds Device/Variable range.

FDEL Deletes Specified Position in Data Table

Data table deleting

FDEL ([EN], Dst, Pos);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Dst*1	Specifies the leading of Table.	.U	.U	.D	.D	.D	—	—	—	—
Pos	Specifies the Delete position in data block, or its storage destination.	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16/32 bits will be processed across the next channel.

●Description of Operation

When Execution condition is TRUE, deletes the data at the position (No.) specified by **Pos** in the data block specified by **Dst**.
After the data is deleted, the contents of the data block are moved forward one by one, and the stored data amount is decremented by 1.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Table Processing Instructions".

●Expression sample

FDEL (bEnable ,auiValue[0], uiValue);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When the value of **Pos** is 0 or exceeds data amount N.
- When the Max. storage data amount "max' is 0.
- When the data amount N exceeds the Max. storage data amount "max".
- When the Max. storage data amount "max" exceeds Device/Variable range.

Time Process Function

WTIME Writes Time Data to Calendar Timer

Time writing
WTIME ([EN], Year, Month, Day, Hour, Min, Sec, DayOfWeek);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Year	Year data (0 to 99)	.U	.U	.U	.U	—	—	—	—	○
Month	Month data (1 to 12)	.U	.U	.U	.U	—	—	—	—	○
Day	Day data (1 to 31)	.U	.U	.U	.U	—	—	—	—	○
Hour	Hour data (0 to 23)	.U	.U	.U	.U	—	—	—	—	○
Min	Minute data (0 to 59)	.U	.U	.U	.U	—	—	—	—	○
Sec	Second data (0 to 59)	.U	.U	.U	.U	—	—	—	—	○
DayOfWeek	Week data (0 to 6)	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

Reference When Word data is specified for **Year** to **Sec**, all data will be processed as 16-bit unsigned data. (The specified suffix is ignored).

●Description of Operation

When Execution condition is TRUE, writes the time data stored in **Year** to **Sec** to the built-in calendar timer.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Time Processing Instructions".

●Expression sample

WTIME (bEnable, uiYear, uiMonth, uiDay, uiHour, uiMin, uiSec, uiDayOfWeek);

Point
- **The time data cannot be written separately and directly to Control memory.**
- **When using the WTIME Function, specify the time data (Year/Month/Day/Hour/Minute/Second/Week) correctly. The day of a week is not checked for Year/Month/Day/Hour/Minute/Second.**
- **If Write data is inappropriate as Time data, WTIME Function will not be executed.**

SEC Converts Date/Time Data to Second Format Data

Converting Date/Time to Second

[R:=] SEC (In);

Argument/ Return value	Description	Identification type								Const ant
		.U	.S	.D	.L	.F	.DF	.B	.T	
In*1*2	Specifies the leading to store the Date/Time convert source.	.U	.U	.U	.U	.F	—	—	—	—
R*3	Return value that stores Operation result. Omissible. If omitted, Conversion result is stored in Internal register.	.U	.S	.D	.L	.F	.DF*4	—	—	—

*1 When Bit data is specified, processes Date/Time data with continuous 16/96 bits, and Second data with continuous 32 bits. Available when a position other than the channel leading such as R002 is specified.

*2 T/C/Z cannot be used.

*3 If a type other than .D is specified to R, type conversion is automatically performed during conversion.

*4 If Device Z is specified, the ".DF" suffix cannot be specified.

●Description of Operation

Converts 6 words (96 bits) of stored Date/Time data starting with In to total seconds with 0:00:00 on January 1, 2000 as the reference Date/Time data, and stores in R as 32-bit BIN data (unsigned).

Reference All Date/Time data is specified as 16-bit BIN data, and either unsigned or signed can be used.

"About the operation flags (CR2009 to CR2012)", (page 1-8)

For details, refer to "Instruction Reference Manual" and "Time Processing Instructions".

●Expression sample

udiValue := **SEC** (auiValue[0]);

Point

When any of the following conditions is TRUE, 0 is stored in R.

- When the Date/Time convert data is incorrect.
- When Timer/Counter is specified for In/R by Indirect designation.

RSEC Converts Second Format Data to Date/Time Data

Converting Second to Date/Time

RSEC ([EN], In, InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the Storage destination for Convert source second (total seconds).	—	—	.D	.D	—	—	—	—	—
InOut	Specifies the leading to store the converted Date/Time data.	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When Bit data is specified, processes Date/Time data with continuous 16/96 bits, and Second data with continuous 32 bits. Available when a position other than the channel leading such as R002 is specified.

*2 Timer/Counter cannot be used.

●Description of Operation

Converts total seconds referenced with 0:00:00 on January 1, 2000 specified by **In** to Date/Time data, and stores in 6 words (96 bits) starting with **InOut**.

Reference All Date/Time data is stored as 16-bit BIN data.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Time Processing Instructions".

●Expression sample

RSEC (bEnable, udiValue, auiValue[0]);

Point **The Function will not be executed if any of the following conditions is TRUE:**

- When the Second convert data is incorrect.
- When Timer/Counter is specified for **In**/**InOut** by Indirect designation.

AJST Adjusts Calendar Timer

Adjusting
AJST ([EN]);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When Execution condition is TRUE, sets it to 0 if "second" value of the current time of the built-in calendar timer is 0 to 29, and if "second" value is 30 to 59, raises to "minute" digit to set "second" to 0 and corrects it to the nearest number in minute units (0 seconds).

Before execution		After execution
12:59:15	→	12:59:00
12:59:30	→	13:00:00

For details, refer to "Instruction Reference Manual" and "Time Processing Instructions".

●Expression sample

AJST (bEnable);

High-speed Process Function

DI	**Disables Interrupt**
EI	**Enables Interrupt**

Disable Interrupt
DI ([EN]);
Enable interrupt
EI ([EN]);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

DI Function disables the execution of Interrupt program when Execution condition is TRUE.
EI Function enables the execution of Interrupt program when Execution condition is TRUE.
By EI Function, enables (releases) the execution of Interrupt program temporarily disabled by DI Function.

- Once the EI Function is executed, it enables interrupts until DI Function is executed.
- Interrupt program is always disabled at the start of operation.
- When an interrupt occurs while the DI function is being executed (interrupt disabled), up to 16 layers are stored.
  When EI Function is executed, the memorized interrupt is executed.
- To execute another Interrupt program while an Interrupt program is being executed, up to 16 layers are available by executing the EI Function in the Interrupt program.

For details on Interrupt Processing, refer to "Instruction Reference Manual", "INT/RETI Instructions" and "High-speed Processing Instructions".

●Expression sample

DI ();

EI ();

Point **The interrupt is always disabled when the operation starts, and once the EI function is executed, the interrupt is enabled until the DI function is executed.**

Reference When an Interrupt program is executed, other interrupts are disabled until the Interrupt program ends. To enable other interrupts, you need to execute the EI Function while the Interrupt program is running. By executing the EI Function, an Interrupt program with a priority higher than the executed interrupt factor can be enabled (multiple interrupts). However, interrupts of the same factor cannot be enabled.

DIC Sets Interrupt Disable Range

Disable Interrupt
DIC ([EN], In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In*1	Specifies the Storage destination for the Interrupt disabled state before instruction execution.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 When the interrupt state storage device is Word data, only the lowest bit is stored in the interrupt state. (No change except for the lowest bit)

●Description of Operation

Disables the execution of Interrupt program when Execution condition is TRUE.
At the same time, sets In if the interrupt is enabled before the Function is executed, and resets if the interrupt is disabled.

For details, refer to "Instruction Reference Manual" and "High-speed Processing Instructions".

●Expression sample

DIC (bEnable, bFlag);

IEDGE Sets Interrupt Condition

Interrupt condition setting
IEDGE ([EN], Unit, Input, In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Unit	Specifies Expansion unit No. to write the Interrupt input detection condition. (0 to 48)	.U	.U	.U	.U	—	—	—	—	—
Input	Specifies the Interrupt factor No. (0 to 31)	.U	.U	.U	.U	—	—	—	—	—
In	Specified the Detection condition (rising edge: 0 or 1, falling edge: 2, rising edge and falling edge: 3).	.U	.U	.U	.U	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When Execution condition is TRUE, sets the 1-word data specified by In as the Interrupt detection condition for the Input-th factor of the Unit-th unit.

Depending on the value of In, the Detection conditions are as follows.

In	Detection condition
(0 to 1)	Rising edge
2	Falling edge
3	Rising edge and Falling edge

Point

- **If 4 to 65535 is specified for In no operation error will occur, but the detection conditions will not be changed.**
- **The above detection condition settings are changed when the execution of this instruction is completed.**

"About the operation flags (CR2009 to CR2012)", (page 1-8)

For details, refer to "Instruction Reference Manual" and "IEDGE Instructions".

●Expression sample

IEDGE (bEnable, uiUnit, uiValue, INT#3);

PID Function

PIDAT PID-controls with Auto-tuning

PID control with AT

PIDAT ([EN], In, Param, InOut);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In	Specifies the value measured from the Control target.	.S	.S	.S	.S	—	—	—	—	—
Param *1	Specifies the leading to store Parameter.	.U	.U	.U	.U	—	—	—	—	—
InOut	Specifies the PID operation result storage destination.	.S	.S	.S	.S	—	—	—	—	—
R	None	—	—	—	—	—	—	—	—	—

*1 The **Param** parameter occupies consecutive 52 words. For details on Parameter, refer to the PIDAT instructions.

●Description of Operation

Gets **InOut** (PID operation result) from **In** and **Param**.
When calling for the first time (to execute initialization), turns Execution condition on and clears the **Param** work area to 0 to call the PIDAT Function.
When continuing the operation, keeps the Execution condition ON to call the PIDAT function.

Reference After calling the PIDAT Function with Execution condition OFF, executes the initialization process when Execution condition is ON.

For details on Operation flag and **Param**, refer to "Instruction Reference Manual" and "PIDAT Instructions".

●Expression sample

PIDAT (bEnable, iPV , auiPIDParam, iMV);

Log Function

LOGE Enables Logging Specified ID

LOGD Disables Logging Specified ID

Enable Logging
LOGE ([EN], In);
Disable Logging
LOGD ([EN], In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In	Specifies the Storage destination to store Logging ID for enable/disable logging, or the Logging ID. (0 to 9)	.U	.U	.U	.U	—	—	—	—	○*1
R	None	—	—	—	—	—	—	—	—	—

*1 Cannot be specified by 16# (HEX).

●Description of Operation

When Execution condition is TRUE, LOGE Function enables logging the ID specified by In. After executing the Function, the Logging execution relay is set for each Logging ID.
When Execution condition is TRUE, LOGD Function disables logging the ID specified by In. After executing the Function, the Logging execution relay is set for each Logging ID.

Logging ID	Logging execution relay
0	CR0000
1	CR0100
2	CR0200
3	CR0300
4	CR0400
5	CR0500
6	CR0600
7	CR0700
8	CR0800
9	CR0900

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to the User's Manual of the used CPU unit.

●Expression sample

LOGE (bEnable, uiValue);

LOGD (bEnable, uiValue);

TRGD Gets Logging Data

Logging data getting
TRGD ([EN], In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In	Specifies the Logging ID. (0 to 10) *1	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 Specify the Logging ID from 0 to 10. When using Trigger instruction in the Real-time chart monitor, specify 10 for the Logging ID.

●Description of Operation

When Execution condition is TRUE, gets Logging ID data specified by **In**.
Normally, Log/Trace/Real-time chart monitor gets data at the timing of END processing, but by using the TRGD instruction, data can be get at any timing of Ladder.

Point
- **You can only use up to 8 TRGD Functions (instructions) for each ID in one project.**
- **To get data with the TRGD Function (instruction), it is necessary to set the trigger type to "Trigger instruction" in Log/Trace/Real-time chart monitor settings of KV STUDIO. If it is not set to "Trigger instruction", no data will be get even if the TRGD Function (instruction) is executed.**

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to the User's Manual of the used CPU unit.

●Expression sample

TRGD (bEnable, INT#1);

TRGR Gets Device/Variable Data

Device/Variable data getting
TRGR ([EN], In);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
In	Specifies the Operation record ID. (0 to 3)	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When Execution condition is TRUE, gets Record ID data specified by **In**.
Normally, gets data at the timing of END processing, but by using the TRGR instruction, data can be get at any timing of Ladder.

Point

- **TRGR instruction can be used only when KV STUDIO Ver.10.0 or later and the compatible model is KV-8000.**
- **To get data with TRGR Function (instruction), it is necessary to set Sampling cycle to "Trigger instruction" in Operation record settings of KV STUDIO. If it is not set to "Trigger instruction", no data will be get even if the TRGR Function (instruction) is executed.**

"About the operation flags (CR2009 to CR2012)", (page 1-8)

For details, refer to "KV-8000 Series User's Manual" and "Function Record Instructions".

●Expression sample

TRGR (bEnable, INT#1);

Storage Function

MWRIT Writes to Storage (Memory card/CPU memory)

Storage writing
[R:=] MWRIT (Path, Src, Count, Param, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path*6	Specifies the File name storage destination to save the data, or the Save file name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL and folder name))	—	—	—	—	—	—	—	.T	○*2
Src*3*4	Specifies the leading to store the Save data.	.U	.U	.U	.U	—	—	—	—	—
Count	Specifies the Storage destination for the data amount in Save range , or the data amount in Save range.	.U	.U	.U	.U	—	—	—	—	—
Param*4*5	Specifies the Parameter related to the Save format, etc., and the leading to store the Data save status.	.U	.U	.U	.U	—	—	—	—	—
Result*2	Specifies Bit data to notify Save result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. Omissible. ON if the Function starts successfully, OFF if fails. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified to **Result**, consecutive 2 bits are used as the notify bits, and when Word data is specified, each lowest bit of consecutive 2 words is used as the notify bit data.
*2 To specify String using Constant, enclose with " ".
*3 To save the digital trimmer value set in Index register Z and Access window, copy it to Data memory DM etc. once and set the Data memory DM.
*4 When specifying Bit data, specify the leading of the channel.
*5 When Bit data is specified for **Param**, occupies consecutive 96 bits. When Word data is specified, occupies consecutive 6 words.
*6 T/C/CR/Z cannot be used.

●Description of Operation

Starting with **Src**, writes the data stored in the range specified by **Count** to Memory card or CPU memory with the File name (string) specified by **Path**.
The Write process is performed according to the Write parameters specified by **Param**, and the Write completion notification and Abnormal end alarm notification are stored in **Result**.
Turns **R** on if MWRIT Function starts normally, off if fails.

For details on Parameters related to Save format specified in **Param**, refer to "Instruction Reference Manual" and "MWRIT Instructions".

Reference

The notification bit of Read result storage destination specified by **Result** has the following meanings.
(Example) When MR2000 is specified

Notification Bit		ON(1, TRUE)	OFF(0, FALSE)
MR2000	Completion notification	Completed	Uncompleted
MR2001	Error notification	Normal end	Abnormal end

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Storage Instructions".

●Expression sample

bFlag := **MWRIT** (strPath, auiValue[0], uiValue, auiMwritParam, abFlag[0]);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When there is no end code (NUL (00H)) in the string specified by **Path**.
- When the string (including the end code (NUL (00H))) specified by **Path** exceeds 247 characters
- When Timer/Counter is specified for **Path** by Indirect designation.
- When Bit data is specified for **Path**/**Src**/**Param** and a position other than the channel leading is specified.
- When a value outside the range is specified for Argument.
- When a value is specified for the Argument reservation.

MREAD Reads from Storage (Memory card/CPU memory)

Storage reading

[R:=] MREAD (Path, Src, Count, Param, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path[*3*6]	Specifies the File name storage destination to read the data, or the Read file name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL and folder name))	—	—	—	—	—	—	—	.T	○[*2]
Src[*3]	Specifies the leading to store the read data.	.U	.U	.U	.U	—	—	—	—	—
Count[*4]	Specifies the Storage destination for the data amount in the Read range, or the Read data amount.	.U	.U	.U	.U	—	—	—	—	○
Param[*5]	Specifies the Parameter related to the Read format, etc., and the leading to store the Data read status.	.U	.U	.U	.U	—	—	—	—	—
Result[*1*3]	Specifies the leading of Bit data to notify Read result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified to **Result**, consecutive 2 bits are used as the notify bits, and when Word data is specified, each lowest bit of consecutive 2 words is used as the notify bit data.
*2 To specify String using Constant, enclose with " ".
*3 When specifying Bit data, specify the leading of the channel.
*4 When reading ASCII string data, specify the maximum number of characters (bytes) in the string.
*5 When Bit data is specified for **Param**, occupies consecutive 96 bits. When Word data is specified, occupies consecutive 6 words.
*6 T/C/CR/Z cannot be used.

●Description of Operation

Reads and stores the File name data (string) specified by **Path** from the storage in the range specified by **Src**, starting with **Count**. The Read process is performed according to the Read parameters specified by **Param**, and the Read completion notification and Abnormal end alarm notification are stored in **Result**.
Turns **R** on if MREAD Function starts normally, off if fails.

For details on Parameters related to Read format specified in **Param**, refer to "Instruction Reference Manual" and "MREAD Instructions".

Reference

The notification bit of **Result** has the following meanings.
(Example) When MR2000 is specified

Notification Bit		ON	OFF
MR2000	Completion notification	Completed	Uncompleted
MR2001	Error notification	Normal end	Abnormal end

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Storage Instructions".

●Expression sample

bFlag := **MREAD** (strPath, auiValue[0], uiValue, auiParam[0], abFlag[0]);

Point

The Function will not be executed if any of the following conditions is TRUE:

- When there is no end code (NUL (00H)) in the string specified by **Path**.
- When the string (including the end code (NUL (00H))) specified by **Path** exceeds 247 characters
- When Timer/Counter is specified for **Path** by Indirect designation.
- When Bit data is specified for **Path**/**Count**/**Param** and a position other than the channel leading is specified.
- When a value outside the range is specified for Argument.
- When a value is specified for the Argument reservation.

MFREE Gets Storage (Memory card/CPU memory) Free Space in Bytes

Storage free space getting
[R:=] MFREE (Drive, InOut, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Drive *3	Specifies the Drive No. (Memory card:0, CPU memory:1) for checking the free space.	.U	.U	.U	.U	—	—	—	—	—
InOut *1	Specifies the Storage destination to store Free space (in bytes) get from Memory card (CPU memory). (32-bit data)	—	—	.D	.D	—	—	—	—	—
Result *2	Specifies Bit data to notify Free space get result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified, operation is performed with consecutive 32 bits.
*2 When Bit data is specified to **Result**, consecutive 2 bits are used as the notify bits, and when Word data is specified, each lowest bit of consecutive 2 words is used as the notify bit data.
*3 You can select Free space check storage from Memory card or CPU memory.

●Description of Operation

Gets the Free space (in bytes) of Memory card/CPU memory specified by **Drive**, stores it as 32-bit BIN data (2-word) in **InOut**, and stores Free space get completion notification and abnormal termination alarm in **Result**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Storage Instructions".

●Expression sample

bFlag := **MFREE** (uiValue, udiValue, abFlag[0]);

● Notification bit data (Result: Completion notification, Result+(1): Error notification)

Item		Details
Result	Completion notification	OFF when Instruction execution starts, ON when Free space get process from Memory card (or CPU memory) ends. Turns on when the instruction ends normally or abnormally.
Result+(1)	Error notification	The instruction turns off when it ends normally, and turns on when it ends abnormally.

Point

- **Results of Completion notification and Error notification are written by END processing after Instruction execution is completed.**
- **It may take several scans to get free space.**
- **If the file is operated via FTP server function during Get process, or if the logging function is used, the get capacity may differ from the actual free space.**
- **MFREE Function is not executed while Storage instruction is being executed (CR3214 is ON).**
- **If you specify a Relay other than the channel leading for InOut, the Function will not be executed.**
- **It cannot be used in Interrupt program, or used for Initialization module.**

MFREEK Gets Storage (Memory card/CPU memory) Free Space in Bytes

Storage free space getting
[R:=] MFREEK (Drive, InOut, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Drive[*3]	Specifies the Drive No. (Memory card:0, CPU memory:1) for checking the free space.	.U	.U	.U	.U	—	—	—	—	—
InOut[*1]	Specifies the Storage destination to store Free space (in bytes) get from Memory card (CPU memory). (32-bit data)	—	—	.D	.D	—	—	—	—	—
Result[*1]	Specifies Bit data to notify Free space get result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 When Bit data is specified, operation is performed with consecutive 32 bits.
*2 When Bit data is specified to **Result**, consecutive 2 bits are used as the notify bits, and when Word data is specified, each lowest bit of consecutive 2 words is used as the notify bit data.
*3 You can select Free space check storage from Memory card or CPU memory.

●Description of Operation

Gets the Free space (in kilobytes) of Memory card or CPU memory specified by **Drive**, stores it as 32-bit BIN data (2 words) in **InOut**, and stores Completion notification/ Abnormal end alarm notification of Free space acquisition in **Result**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "MFREEK Instructions".

●Expression sample

bFlag := **MFREEK** (uiValue, udiValue, abFlag[0]);

● Notification bit data (Result: Completion notification, Result+(1): Error notification)

Item		Details
Result	Completion notification	Turns of when Instruction execution starts, and turns on when Free space get process from Memory card (CPU memory) ends. Turns on when the instruction ends normally or abnormally.
Result+(1)	Error notification	The instruction turns off when it ends normally, and turns on when it ends abnormally.

Point

- **Results of Completion notification and Error notification are written by END processing after Instruction execution is completed.**
- **It may take several scans to get free space.**
- **If a file operation is performed via FTP server function while getting, or if Logging Function is used, the get capacity may differ from the actual free space.**
- **MFREEK Function is not executed while Storage instruction is being executed (CR3214 is ON).**
- **If you specify a Relay other than the channel leading for InOut, the Function will not be executed.**
- **It cannot be used in Interrupt program, or used for Initialization module.**

MMKDIR Creates Storage (Memory card/CPU memory) Folder

Storage folder creating
[R:=] MMKDIR (Path, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path*2*4	Specifies the Create folder path name, or its storage destination. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	—	—	—	—	—	—	—	.T	○*1
Result*3	Specifies Bit data to notify Create result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 To specify String using Constant, enclose with " ".
*2 When specifying Bit data, specify the leading of the channel.
*3 When Bit data is specified, occupies consecutive 2 bits. When Word data is specified, the lowest bit of each consecutive 2 words are used as Notification bit data. No change except for the lowest bit.
*4 T/C/CR/Z cannot be used.

● Description of Operation

Creates a folder in Memory card or CPU memory with the path name (string) specified in Path.
The Create result is stored in Result.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Storage Instructions".

● Expression sample

bFlag := **MMKDIR** (strPath, abFlag[0]);

● Notification bit data (Result: Completion notification, Result+(1): Error notification)

Item		Details
Result	Completion notification	Turns off at the start of Function execution and turns on at the end of Folder creation. Always ON regardless of whether Function ends normally or abnormally.
Result+(1)	Error notification	Turns off when it ends normally, and turns on when it ends abnormally.

Point

- It may take several scans to create a folder.
- The result of notification bit data is written by END processing after Function execution is completed.
- When Indirect designation or Index modify is specified for Notification bit data, the Storage destination is created when Execution condition rises.
- MMKDIR Function is not executed while Storage instruction is being executed (CR3214 is ON).
- The Function will not be executed if any of the following conditions is TRUE:
  - When there is no end code (NUL (00H)) in the string specified by Path.
  - When the string (including the end code (NUL (00H))) specified by Path exceeds 247 characters
  - When Timer/Counter is specified for Path by Indirect designation.
  - When Bit data is specified for Path and a position other than the channel leading is specified.
- It cannot be used in Interrupt program, or used for Initialization module.

MRMDIR Deletes Storage (Memory card/CPU memory) Folder

Storage folder deleting

[R:=] MRMDIR (Path, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path*2*4	Specifies the Delete folder path name, or its storage destination. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	—	—	—	—	—	—	—	.T	○*1
Result*3	Specifies Bit data to notify Delete result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 To specify String using Constant, enclose with " ".
*2 When specifying Bit data, specify the leading of the channel.
*3 When Bit data is specified, occupies consecutive 2 bits. When Word data is specified, the lowest bit of each consecutive 2 words are used as Notification bit data. No change except for the lowest bit.
*4 T/C/CR/Z cannot be used.

●Description of Operation

Deletes the folder with the path name (string) specified in Path from Memory card or CPU memory .
The Delete result is stored in Result.

- **When the folder specified by Path does not exist on Memory card**
  Result + 1 turns on, and 3 (no file or folder) is stored in CM2390.
- **When a file or folder exists in the folder specified by Path**
  After deleting all the files and folders existing in the folder, the specified folder is deleted.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Storage Instructions".

●Expression sample

bFlag := **MRMDIR** (strPath, abFlag[0]);

●Notification bit data storage (Result: Completion notification,Result+(1): Error notification)

Item		Details
Result	Completion notification	Turns off at the start of Function execution and turns on at the end of Folder creation. Always ON regardless of whether Function ends normally or abnormally.
Result+(1)	Error notification	Turns off when it ends normally, and turns on when it ends abnormally.

Point

- It may take several scans to delete a folder.
- The result of notification bit data is written by END processing after Function execution is completed.
- When Indirect designation or Index modify is specified for Notification bit data, the Storage destination is created when Execution condition rises.
- MRMDIR Function is not executed while Storage instruction is being executed (CR3214 is ON).
- The Function will not be executed if any of the following conditions is TRUE:
  - When there is no end code (NUL (00H)) in the string specified by Path.
  - When the string (including the end code (NUL (00H))) specified by Path exceeds 247 characters
  - When Timer/Counter is specified for Path by Indirect designation.
  - When Bit data is specified for Path and a position other than the channel leading is specified.
- It cannot be used in Interrupt program, or used for Initialization module.

MDEL Deletes Storage (Memory card/CPU memory) File

Storage file deleting

[R:=] MDEL (Path, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path*2*4	Specifies the File name storage destination to delete the file, or the Delete file name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	—	—	—	—	—	—	—	.T	○*1
Result*3	Specifies Bit data to notify Delete result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 To specify String using Constant, enclose with " ".
*2 When specifying Bit data, specify the leading of the channel.
*3 When Bit data is specified, occupies consecutive 2 bits. When Word data is specified, the lowest bit of each consecutive 2 words are used as Notification bit data. No change except for the lowest bit.
*4 T/C/CR/Z cannot be used.

●Description of Operation

Deletes the file with the file name (string) specified in Path from Memory card or CPU memory .
The Delete result is stored in Result.

- **When the folder specified by Path does not exist on Memory card**
  Result+(1) turns on, and 3 (no file or folder) is stored in CM2390.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Storage Instructions".

●Expression sample

bFlag := **MDEL** (strpath, abFlag[0]);

●Notification bit data storage (Result: Completion notification,Result+(1): Error notification)

Item		Details
Result	Completion notification	Turns off at the start of Function execution and turns on at the end of File deletion. Always ON regardless of whether Function ends normally or abnormally.
Result+(1)	Error notification	Turns off when it ends normally, and turns on when it ends abnormally.

Point

- It may take several scans to delete a file.
- The result of notification bit data is written by END processing after Function execution is completed.
- When Indirect designation or Index modify is specified for Notification bit data, the Storage destination is created when Execution condition rises.
- MDEL Function is not executed while Storage instruction is being executed (CR3214 is ON).
- The Function will not be executed if any of the following conditions is TRUE:
  - When there is no end code (NUL (00H)) in the string specified by Path.
  - When the string (including the end code (NUL (00H))) specified by Path exceeds 247 characters
  - When Timer/Counter is specified for Path by Indirect designation.
  - When Bit data is specified for Path and a position other than the channel leading is specified.
- It cannot be used in Interrupt program, or used for Initialization module.

MPRINT Writes text to Storage (Memory card/CPU memory) File

Storage string writing
[R:=] MPRINT (Path, In, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path*2*4	Specifies the File name storage destination to save the data, or the Save file name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	.U	.S	.D	.L	.F	.DF	.B	.T	○
In*3	Specifies the leading to store Write string, or the Write string.	.U	.S	.D	.L	.F	.DF	.B	.T	○
Result*3	Specifies Bit data to notify Write result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	.U	.S	.D	.L	.F	.DF	—	—	—

*1 When Bit data is specified for the Write result storage destination, consecutive 2 bits is used for Notification Bit.
*2 To specify String using Constant, enclose with " ".
*3 When specifying Bit data, specify the leading of the channel.
*4 T/C/Z cannot be used.

●Description of Operation

Writes the string specified by In to Memory card at the end of the file name (string) specified by Path.
If the Path name does not exist, creates a new file.

The Write completion notification and Abnormal end alarm notification are stored in Result.
Turns R on if MPRINT Function starts normally, off if fails.

Reference

The notification bit of Result has the following meanings.
(Example) When MR2000 is specified

Notification Bit		ON(1, TRUE)	OFF(0, FALSE)
MR2000	Completion notification	Completed	Uncompleted
MR2001	Error notification	Normal end	Abnormal end

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "Storage Instructions".

●Expression sample

bFlag := **MPRINT** (strPath, strString, abFlag[0]);

MREADL Reads One Line from Storage (Memory card/CPU memory) File

One line reading from Storage
[R:=] MREADL (Path, Dst, Param, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path*5	Specifies the File name storage destination to read the data, or the Read file name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	—	—	—	—	—	—	—	.T	○
Dst*4	Specifies the leading to store the read string.	—	—	—	—	—	—	—	.T	○*3
Param*1	Specifies the read parameter.	.U	.U	.U	.U	—	—	—	—	—
Result*2	Specifies the leading of Bit data to notify Read result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 **Param** occupies consecutive 4 words.
*2 **Result** uses consecutive 4 bits as Notification bit.
*3 To specify String using Constant, enclose with " ".
*4 When specifying Bit data, specify the leading of the channel.
*5 T/C/Z cannot be used.

●Description of Operation

Reads one line from the file name (string) specified by **Path** and reads it into **Dst**.
(The character code representing the line is 0DH (CR), 0AH (LF), 0D0AH (CR + LF))
The maximum byte size of characters to be read is set by **Param**+(2) (upper limit number of read bytes).
If the text data length of one line in the file exceeds the length specified by **Param**+(2),
Result+(3) (no line feed notification) turns on and the text data up to **Param**+(2) bytes is stored in **Dst**.

The Read completion notification and Abnormal end alarm notification are stored in **Result**.
Turns **R** on if MREADL Function starts normally, off if fails.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details on **Param** and **Result** parameters, refer to "Instruction Reference Manual" and "MREADL Instructions".

●Expression sample

bFlag := **MREADL** (strPath, strString, auiValue[0], abFlag[0]);

MCOPY Copies File from Storage (Memory card/CPU memory)

Storage file copying
[R:=] MCOPY (Src, Dst, Param, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Src*4*5	Specifies the Copy source file/folder name storage destination, or the File/Folder name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	—	—	—	—	—	—	—	.T	○
Dst*4*5	Specifies the Copy file/folder name storage destination, or the File/Folder name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL and folder name))	—	—	—	—	—	—	—	.T	○*3
Param*1	Specifies the Copy format parameter.	.U	.U	.U	.U	—	—	—	—	—
Result*2	Specifies Bit data to notify Copy result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 Param occupies 1 word.
*2 Result uses consecutive 2 bits as Notification bit.
*3 To specify String using Constant, enclose with " ".
*4 When specifying Bit data, specify the leading of the channel.
*5 T/C/Z cannot be used.

●Description of Operation

Copies the file name (string) specified by Src to the file specified by Dst.
If a file/folder with the same name exists in Copy destination, it is determined according to Param whether to overwrite or input an error.
When the value of Copy format parameter = 0: If a file with the same name exists, overwrite the file with the same name
When the value of Copy format parameter = 1: When a file with the same name exists, it ends with an error without overwriting.

The Copy completion notification and Abnormal end alarm notification are stored in Result.
Turns R on if MCOPY Function starts normally, off if fails.
Combination table of File/Folder names which can be specified for Src and Dst

		Dst	
		File name	Folder name
Src	File name	Normal operation	Normal operation
	Folder name	Error occurs	Normal operation

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details on Parameter of Param and Copy result of Result, refer to "Instruction Reference Manual" and "MCOPY Instructions".

●Expression sample

bFlag := **MCOPY** (strPathSrc, strPathDst, auiValue[0], abFlag[0]);

MMOV Moves Storage (Memory card/CPU memory) File

Storage file moving
[R:=] MMOV (Src, Dst, Param, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Src*4*5	Specifies the Move source file/folder name storage destination, or the File/Folder name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	—	—	—	—	—	—	—	.T	○
Dst*4*5	Specifies the Move file/folder name storage destination, or the File/Folder name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL and folder name))	—	—	—	—	—	—	—	.T	○*3
Param*1	Specifies the Move format parameters.	.U	.U	.U	.U	—	—	—	—	—
Result*2	Specifies Bit data to notify Move result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 **Param** occupies 1 word.
*2 **Result** uses consecutive 2 bits as Notification bit.
*3 To specify String using Constant, enclose with " ".
*4 When specifying Bit data, specify the leading of the channel.
*5 T/C/Z cannot be used.

●Description of Operation

Moves the File name (string) specified by **Src** to the file specified by **Dst**.
If a file/folder with the same name exists in Move destination, it is determined according to **Param** whether to overwrite or input an error.
If the Move format parameter value = 0: When a file with the same name exists, the file with the same name is overwritten.
If the Move format parameter value = 1: When a file with the same name exists, it ends with an error without overwriting.

The Move completion notification and Abnormal end alarm notification are stored in **Result**.
Turns **R** on if MMOV Function starts normally, off if fails.

Combination table of File/Folder names which can be specified for **Src** and **Dst**

		Dst	
		File name	Folder name
Src	File name	Normal operation	Normal operation
	Folder name	Error occurs	Normal operation

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details on Parameter of **Param** and Copy result of **Result**, refer to "Instruction Reference Manual" and "MMOV Instructions".

●Expression sample

bFlag := **MMOV** (strPathSrc, strPathDst, auiValue[0], abFlag[0]);

MREN Renames Storage (Memory card/CPU memory) File

Storage file renaming
[R:=] MREN (Path, Name, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path[*3*4]	Specifies the Rename source file/folder name storage destination, or the File/ Folder name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL))	—	—	—	—	—	—	—	.T	○
Name[*3*4]	Specifies the Renamed source file/folder name storage destination, or the File/ Folder name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL and folder name))	—	—	—	—	—	—	—	.T	○[*2]
Result[*1]	Specifies Bit data to notify Rename result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. ON if the Function starts successfully, OFF if fails. Omissible. If omitted, it is stored in Internal register.	—	—	—	—	—	—	.B	—	—

*1 **Result** uses consecutive 2 bits as Notification bit.
*2 To specify String using Constant, enclose with " ".
*3 When specifying Bit data, specify the leading of the channel.
*4 T/C/Z cannot be used.

●Description of Operation

Renames the File/Folder name (string) specified by **Path** to the File/Folder name specified by **Name**.
The Rename completion notification and Abnormal end alarm notification are stored in **Result**.
Turns **R** on if MREN Function starts normally, off if fails.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details on operation, refer to "Instruction Reference Manual" and "MREN Instructions".

●Expression sample

bFlag := **MREN** (strPath, strString, abFlag[0]);

MSTAT Gets Storage (Memory card/CPU memory) File Status

Storage file status getting
[R:=] MSTAT (Path, InOut, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
Path*4	Specifies the Storage destination for the File name to be status-get, or the File name. (Specifies a string in the range of 1 to 247 characters (including the terminal code NUL and folder name))	—	—	—	—	—	—	—	.T	○*2
InOut*3	Specifies the File status storage destination.	.U	.U	.U	.U	—	—	—	—	—
Result*1	Specifies Bit data to notify Get result.	—	—	—	—	—	—	.B	—	—
R	Return value that stores Operation result. Omissible. If omitted, it is stored in Internal register. SEG Function has .T identification type.	—	—	—	—	—	—	.B	—	—

*1 **Result** uses consecutive 3 bits as Notification bit.
*2 To specify String using Constant, enclose with " ".
*3 File status storage destination occupies consecutive 10 words.
*4 T/C/Z cannot be used.

●Description of Operation

Gets the File status specified by **Path** and stores it in **InOut**.
The Get completion notification, Abnormal end alarm notification and File existence info are stored in **Result**.
Turns **R** on if MSTAT Function starts normally, off if fails.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual" and "MSTAT Instructions".

●Expression sample

bFlag := **MSTAT** (strPath, auiValue, abFlag[0]);

Access Window Function

AWSHOW Displays User's Message

User message displaying
AWSHOW (EN, ID);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	When Execution condition turns from off to on, a user message will be displayed in Access window.	—	—	—	—	—	—	.B	—	○
ID	Specifies the User message ID to be displayed in Access window. (1 to 63)	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When **EN** turns from off to on, displays User message specified by **ID**.

Reference To hide User message, operate on Access window or execute AWHIDE instruction.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual", "Access window Instructions", or the "Access window Instructions" in User's Manual of the used CPU unit.

●Expression sample

AWSHOW (bFlag, uiValue);

Point
- **It does not change even if the ID value is changed when Execution condition is ON.**
- **The CR is set for the User message with the ID specified at Execution condition rising edge.**
- **If AWSHOW and AWHIDE Functions are executed at the same time, the behavior of the later one will be followed.**
- **If there are multiple AWSHOW Functions in the project, the Access window status will follow the first AWSHOW Function executed.**
- **It cannot be used in Interrupt program, or used for Initialization module.**

AWHIDE Hides User's Message

User message hiding
AWHIDE (EN, ID);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	When Execution condition turns from off to on, the user message displayed in Access window is hidden.	—	—	—	—	—	—	.B	—	—
ID	Specifies the User message ID to be hidden in Access window. (1 to 63)	.U	.U	.U	.U	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When **EN** turns from off to on, hides User message specified by **ID**.

"About the operation flags (CR2009 to CR2012)", (page 1-8)
For details, refer to "Instruction Reference Manual", "Access window Instructions", or the "Access window Instructions" in User's Manual of the used CPU unit.

●Expression sample

AWHIDE (bFlag, uiValue);

Point

- **It does not change even if the ID value is changed when Execution condition is ON.**
- **The CR is set for the User message with the ID specified at Execution condition rising edge.**
- **If AWSHOW and AWHIDE Functions are executed at the same time, the behavior of the later one will be followed.**
- **It cannot be used in Interrupt program, or used for Initialization module.**

FREE RUN Counter Function

RFSFRC Refreshes FREE RUN Counter

FREE RUN Counter refreshing
RFSFRC ([EN]);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
R	None	—	—	—	—	—	—	—	—	—

●Description of Operation

When Execution condition is TRUE, refreshes FREE RUN counter to the latest value.
The refreshed FREE RUN Counters are as follows.

CM1030 to CM1031：1ms Counter
CM1032 to CM1033：10μs Counter
CM1034 to CM1035：1μs Counter
CM1036 to CM1037：0.1μs Counter

For details, refer to "Instruction Reference Manual" and "FREE RUN Counter Instructions".

●Expression sample

RFSFRC (bEnable);

Sensor Setting Function

SPRD Reads Sensor Parameter

Reads Sensor Parameter
SPRD (EN, Unit, Node, Slot, Param, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. (0 to 48, '0' for Main unit)	—	—	—	—	—	—	—	—	○
Node*3	Specifies Node address, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Slot*3	Specifies Slot No., or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Param*3	Specifies Parameter No., or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Result*2*4*5	Specifies the leading of Completion code/Parameter read result storage device.	.U	.U	.U	.U	.F	.DF	.B	.T	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.
*2 Z cannot be specified.
*3 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16 bits will be processed across the next channel.
*4 When specifying Bit data, specify the leading of the channel.
*5 The number of words stored (4 words or more) changes depending on Execution result.

●Description of Operation

When **EN** is ON, reads Parameters specified by **Param** sensor with **Node** address and **Slot** No. that connected to **Unit** No. unit. After completion, stores Completion bit in the 0th bit of **Result**, Execution failure bit in the 1st bit of **Result**, Completion code in **Result**＋(1), Detailed completion code in **Result**＋(2), the number of Array elements (1 if it is not Array) in **Result**＋(3), and stores the read Parameter contents in **Result**＋(4) onward.

Result to : Execution result storage destination

Execution result storage destination			Details
Result	0 bit	Completion bit	OFF when starting to execute Instruction, ON when completed. Always ON regardless of whether Instruction ends normally or abnormally.
	1 bit	Execution failure bit	If Instruction execution fails, turns on simultaneously with Completion bit. Turns OFF when starting to execute Instruction.
	2 to 15 bit	System-reserved	OFF (fixed value)
Result＋(1)	Completion code		Completion code is stored when Parameter read is complete. (0 when normal)
Result＋(2)	Detailed completion code		Detailed completion code is stored when Parameter read is complete.
Result＋(3)	Number of array elements		Stores the number of elements in the read parameter array. 1 is stored if it is not Array, and 0 is stored if the instruction completes abnormally.
Result＋(4 to)	Read result		The value of Parameter is stored. The occupied data amount changes depending on the parameters to be read.

Point **The execution will fail if an "EtherNet/IP reserved" Sensor is specified.**

If Sensor parameter read instruction is executed at the rising edge of Execution condition, the processing will continue even if the input condition is turned off in the next scan.
Completion bit/Failure bit of Result storage destination are turned on at the time of completion and turned off at the start of execution.
When the instruction is executed, the number of Sensor setting instruction executions is incremented by 1, and is decremented when the execution ends

For details, refer to "Instruction Reference Manual" and "Sensor Setting Instructions".

●Expression sample

SPRD (bFlag, 1, uiNode, uiSlot, uiValue, auiValue[0]);

SPWR Writes Sensor Parameter

Writes Sensor Parameter

SPWR (EN, Unit, Node, Slot, Param, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. (0 to 48, '0' for Main unit)	—	—	—	—	—	—	—	—	○
Node*3	Specifies Node address, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Slot*3	Specifies Slot No., or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Param*3	Specifies Parameter No., or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Result*2*4*5	Specifies the leading of Completion code/Parameter read result storage device.	.U	.U	.U	.U	.F	.DF	.B	.T	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.
*2 Z cannot be specified.
*3 When Bit data is specified, operation is performed with consecutive 16/32 bits. When a position other than the channel leading such as R002 is specified, 16 bits will be processed across the next channel.
*4 When specifying Bit data, specify the leading of the channel.
*5 The number of words stored (4 words or more) changes depending on Execution result.

●Description of Operation

When **EN** is ON, writes the Set value specified by **Result**＋(4) onward. After completed, stores Completion bit as the 0th bit of **Result**, Execution failure bit as the 1st bit of **Result**, Completion code as **Result**＋(1), and Detailed completion code as **Result**＋(2), into the Parameter specified by **Param** sensor with **Node** address and **Slot** No. that connected to **Unit** No. unit

Result ～ : Execution result storage destination

Execution result storage destination			Details
Result	0 bit	Completion bit	OFF when starting to execute Instruction, ON when Parameter write is complete. Always ON regardless of whether Instruction ends normally or abnormally.
	1 bit	Execution failure bit	If Instruction execution fails, turns on simultaneously with Completion bit. Turns OFF when starting to execute Instruction.
	2 to 15 bit	System-reserved	OFF (fixed value)
Result＋(1)	Completion code		Completion code is stored when Parameter write is complete. (0 when normal)
Result＋(2)	Detailed completion code		Completion code is stored when Parameter write is complete.
Result＋(3)	System-reserved		0 (fixed value)
Result＋(4 to)	Set value		Stores Set value The occupied data amount changes depending on the parameters to be written.

Point **The execution will fail if an "EtherNet/IP reserved" Sensor (Adapter) is specified.**

If Sensor parameter write instruction is executed at the rising edge of Execution condition, the processing will continue even if the input condition is turned off in the next scan.
Completion bit/Failure bit of Result storage destination are turned on at the time of completion and turned off at the start of execution.
When the instruction is executed, the number of Sensor setting instruction executions is incremented by 1, and is decremented when the execution ends

For details, refer to "Instruction Reference Manual" and "Sensor Setting Instructions".

●Expression sample

SPWR (bFlag, 1, uiNode, uiSlot, uiValue, auiValue[0]);

SSVC Executes Sensor Service

Executes Sensor Service
SSVC (EN, Unit, Node, Slot, Service, Result);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Specifies Bit data to be used for Execution condition.	—	—	—	—	—	—	.B	—	—
Unit*1	Specifies Unit No. (0 to 48, '0' for Main unit)	—	—	—	—	—	—	—	—	○
Node*3	Specifies Node address, or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Slot*3	Specifies Slot No., or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Service*3	Specifies Parameter No., or its Storage destination.	.U	.U	.U	.U	—	—	—	—	○
Result*2*4*5	Specifies the leading of Completion code/Parameter read result storage device.	.U	.U	.U	.U	.F	.DF	.B	.T	—
R	None	—	—	—	—	—	—	—	—	—

*1 16# (HEX specifying) cannot be used.
*2 Z cannot be specified.
*3 When Bit data is specified, operation is performed with consecutive 16 bits. When a position other than the channel leading such as R002 is specified, 16 bits will be processed across the next channel.
*4 When specifying Bit data, specify the leading of the channel.
*5 Use up to 6 words depending on the specified service.

●Description of Operation

When **EN** is ON, executes the Service specified by **Service** sensor with **Node** address and **Slot** No. that connected to **Unit** No. unit. After completed, stores Completion bit in the 0th bit of **Result**, Execution failure bit in the 1st bit of **Result**, Completion code in **Result**＋(1), and Detailed completion code in **Result**＋(2).

Result ~ : Execution result storage destination

Execution result storage destination			Details
Result	0 bit	Completion bit	OFF when starting to execute Instruction, ON when Service execute is complete. Always ON regardless of whether Instruction ends normally or abnormally.
	1 bit	Execution failure bit	If Instruction execution fails, turns on simultaneously with Completion bit. Turns OFF when starting to execute Instruction.
	2 to 15 bit	System-reserved	OFF (fixed value)
Result＋(1)	Completion code		Completion code is stored when Service execute is complete. (0 when normal)
Result＋(2)	Detailed completion code		Completion code is stored when Service execute is complete.
Result＋(3)	System-reserved		0 (fixed value)
Result＋(4) Result＋(5)	Send data		Specifies the data to be sent (up to 2 words) according to the service to be executed.

Point **The execution will fail if an "EtherNet/IP reserved" Sensor is specified.**

If Sensor service execute instruction is executed at the rising edge of Execution condition, the processing will continue even if the input condition is turned off in the next scan.

Completion bit/Failure bit of Result storage destination are turned on at the time of completion and turned off at the start of execution.
When the instruction is executed, the number of Sensor setting instruction executions is incremented by 1, and is decremented when the execution ends
For details, refer to "Instruction Reference Manual" and "Sensor Setting Instructions".

●Expression sample

SSVC (bFlag, 1, uiNode, uiSlot, uiValue, auiValue[0]);

Cyclic Communication Refresh Function

RFSCI Input-refreshes Cyclic Communication

Input-refreshes Cyclic Communication

RFSCI ([EN], Start, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Start*1	Specifies the leading to refresh Cyclic communication input data.	.U	.S	.D	.L	.F	—	—	—	—
Count	Specifies the data amount to be refreshed.	—	—	—	—	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When specifying Bit data, specify the leading of the channel.
When specifying Array type variable, Device/Variable cannot be specified to the element number.

●Description of Operation

When Execution condition is ON, refreshes Cyclic communication input data for the data amount specified by **Count** from the data assigned to **Start**. (When the leading is 32-bit data, the number of words to be refreshed is the amount of data x 2.)

For details, refer to "Instruction Reference Manual" and "Cyclic Communication Instructions".

●Expression sample

RFSCI (bEnable, auiValue[0], 128);

RFSCO Output-refreshes Cyclic Communication

Output-refreshes Cyclic Communication

RFSCO ([EN], Start, Count);

Argument/ Return value	Description	Identification type								Constant
		.U	.S	.D	.L	.F	.DF	.B	.T	
EN	Execution condition. Omissible. If omitted, always (every scan) is executed.	—	—	—	—	—	—	.B	—	—
Start*1	Specifies the leading to refresh Cyclic communication output data.	.U	.S	.D	.L	.F	—	—	—	—
Count	Specifies the data amount to be refreshed.	—	—	—	—	—	—	—	—	○
R	None	—	—	—	—	—	—	—	—	—

*1 When specifying Bit data, specify the leading of the channel.
When specifying Array type variable, Device/Variable cannot be specified to the element number.

●Description of Operation

When Execution condition is ON, refreshes Cyclic communication output data for the data amount specified by **Count** from the data assigned to **Start**. (When the leading is 32-bit data, the number of words to be refreshed is the amount of data x 2.)

For details, refer to "Instruction Reference Manual" and "Cyclic Communication Instructions".

●Expression sample

RFSCO (bEnable, auiValue[0], 128);

Appendix

CR/CM List A-2
ASCII Code Table A-10
List of Characters That Cannot Be Used A-11
Unusable Function List A-13
Index A-15
Functions Index A-17

CR/CM List

Control relay (CR)

Control relays (CR0000 to CR7915) can be used to perform operations such as checking the operation state of the CPU unit and setting and monitoring the built-in functions.
System reserved control relays cannot be used.

CR	Attribute R: Read only Blank: Read/Write possible	Function	
CR0000		Log/Trace Function(ID0)	Log/Trace execution (ON: enable, OFF: disable)
CR0001	R		Log/Trace operation in progress
CR0002	R		Log/Trace file writing completion
CR0003	R		Log buffer less than 50% warning
CR0004	R		Log buffer overflow
CR0005	R		Data written during log in RUN mode
CR0006 to CR0007	–		System reserved
CR0008	R		Log/Trace error
CR0009	R		No log/trace memory capacity error
CR0010	R		Log/Trace device/Trigger setting error
CR0011	R		Trace file save trigger monitor
CR0012	R		Trace data get completion
CR0013	R		Data written for log/trace setting
CR0014 to CR0015	–		System reserved
CR0100 to CR0915		Log/Trace function (ID1 to ID9)	
CR1000 to CR1015		Real-time chart monitor (ID10)	
CR1100 to CR2001	–	System reserved	
CR2002	R	Always turns ON	
CR2003	R	Always turns OFF	
CR2004	R	10 ms clock pulse (duty cycle: 50%)	
CR2005	R	100 ms clock pulse (duty cycle: 50%)	
CR2006	R	1 s clock pulse (duty cycle: 50%)	
CR2007	R	Turns OFF for just one scan when operation starts	
CR2008	R	Turns ON for just one scan when operation starts	
CR2009		Turns ON when the calculation result is negative or when an overflow occurs	
CR2010	R	Turns ON when the calculation result is zero	
CR2011	R	Turns ON when the calculation result is positive	
CR2012	R	Turns ON when a calculation execution error occurs	
CR2013 to CR2302	–	System reserved	
CR2303		Fixed scan time operation (ON: enabled, OFF: disabled)	
CR2304	R	Scan time setting (turns ON for just one scan when the fixed scan time operation and END process time setting are exceeded)	
CR2305	–	System reserved	
CR2306	R	Backup battery errors (ON: error, OFF: normal)	
CR2307		Minimum/maximum scan time value clear	
CR2308		END process time setting (ON: enabled, OFF: disabled; if CR2303 turns ON at the same time, CR2303 is given priority)	
CR2309 to CR2310	–	System reserved	
CR2311	R	RTC low voltage error (ON: error, OFF: normal)	
CR2312	–	System reserved	
CR2313		VT sensor application screen forced switch (ON: do, OFF: undo)	
CR2314	R	Project password authentication prohibited state (ON: disabled, OFF: enabled)	
CR2315 to CR2813	–	System reserved	
CR2814		Zero suppression during DASC and FASC instruction execution (for all suffixes) (ON: do, OFF: undo)	
CR2815		Sign of "+" omission during DASC.S, DASC.L, and FASC instruction execution (ON: do OFF: undo)	
CR2900 to CR3005	–	System reserved	
CR3100	R	Inter-unit sync-refresh time exceeded	
CR3101 to CR3115	–	System reserved	
CR3200		CPU memory, Save request	
CR3201	R	CPU memory, Saving	
CR3202	R	CPU memory, Save completed	
CR3203	R	CPU memory, Save error	

CR	Attribute R: Read only Blank: Read/Write possible	Function
CR3204 to CR3206	—	System reserved
CR3207		Instruction RAM mode operation Enable the File register instruction and the storage instruction RAM mode for CPU memory (ON: enable, OFF: disable)
CR3208 to CR3209	—	System reserved
CR3210	R	Specified log/trace is operating
CR3211 to CR3213	—	System reserved
CR3214	R	Storage instruction in progress
CR3215	—	System reserved
CR3300 to CR3415		Alarm relay
CR3500	R	Alarm operation relay (ON: one of CR3300 to CR3415 turns ON)
CR3501		Record log when Alarm relay switches from ON to OFF (ON→OFF: record, OFF: do not record)
CR3502		Clear Alarm relay log when it switches from OFF→ to ON
CR3503 to CR3512	—	System reserved
CR3513	R	Internal log buffer overflow
CR3514		HKEY instruction, Multiple keys disabled
CR3515	R	HKEY instruction, Scan end
CR3600 to CR3615	R	HKEY information storage area
CR3700 to CR3901	—	System reserved
CR3902	R	Maximum number of serious error history records
CR3903	R	Maximum number of mild error history records
CR3904	R	RUN-PRG selector switch state (ON: RUN, OFF: PRG)
CR3905	R	A serious error has occurred
CR3906	R	A mild error has occurred
CR3907 to CR3908	—	System reserved
CR3909		Clear the current error (set it in the Ladder program)
CR3910		Clear all errors and event history when it switches from OFF to ON
CR3911 to CR4000	—	System reserved
CR4001 to CR4300	R	Expansion unit state (expansion units 1 to 48)
CR4301 to CR4315	—	System reserved
CR4400	R	AW Up key being pressed
CR4401	R	AW Down key being pressed
CR4402	R	AW Left key being pressed
CR4403	R	AW Right key being pressed
CR4404	R	AW MENU key being pressed
CR4405	R	AW ENT key being pressed
CR4406 to CR4414	—	System reserved
CR4415	R	AW User message being displayed
CR4500		AW Line 1 blinking
CR4501		AW Line 2 blinking
CR4502		AW Line 3 blinking
CR4503		AW Line 4 blinking
CR4504 to CR4600	—	System reserved
CR4601 to CR4915		AW user messages 1 to 63
CR5000 to CR5315	—	System reserved
CR5400		Memory card mount request
CR5401		Memory card unmount request
CR5402	R	Memory card being used
CR5403	R	Memory card recognized
CR5404	R	Memory card mounted
CR5405	R	Memory card write is protected
CR5406 to CR5407	—	System reserved
CR5408		CPU memory mount request
CR5409		CPU memory unmount request
CR5410	R	CPU memory being used
CR5411	R	CPU memory recognized
CR5412 to CR4915	—	System reserved
CR5500		Project load execution request
CR5501	—	System reserved
CR5502		Project save execution request
CR5503	—	System reserved
CR5504	R	Project load execution completed
CR5505	R	Project load execution failure

CR	Attribute R: Read only Blank: Read/Write possible	Function	
CR5506	R	Project save execution completed	
CR5507	R	Project save execution failure	
R5508		Recipe load request	
R5509	—	System reserved	
R5510	R	Recipe load completed	
R5511	R	Recipe load failure	
R5512		Recipe save request	
R5513	—	System reserved	
R5514	R	Recipe save completed	
R5515	R	Recipe save failure	
CR5600		Operation record function(ID0)	Operation record 0, Save request
CR5601	R		Operation record 0, Operating
CR5602	R		Operation record 0, Save trigger monitor
CR5603	R		Operation record 0, Saving
CR5604	R		Operation record 0, Collecting
CR5605	R		Operation record 0, Setting error
CR5606	—		System reserved
DR5607	R		Operation record 0, Save completed
CR5608	R		Operation record 0, Save error
CR5609	R		Device record 0, Buffer shortage
CR5610 to CR5613	—		System reserved
CR5614			Operation record 0 Buffer area all save mode
CR5615	—		System reserved
CR5700 to CR5919		Operation record function (ID1 to ID3)	
CR6000 to CR7915	—	System reserved	

Control memory CM

Control memories (CM0000 to CM7599) can be used to perform operations such as checking the operation state of the CPU unit and setting and monitoring the built-in functions.
System reserved control memories cannot be used.

CM	Attribute R: Read only Blank: Read/write	Function	Hold Yes: Hold
Cm0000 to CM0699	—	System reserved	
CM0700	R	Read from RTC (year)	
CM0701	R	Read from RTC (month)	
CM0702	R	Read from RTC (day)	
CM0703	R	Read from RTC (hour)	
CM0704	R	Read from RTC (minute)	
CM0705	R	Read from RTC (second)	
CM0706	R	Read from RTC (day of the week)	
CM0707 to CM709	—	System reserved	
CM0710	R	File register current bank No.	
CM0711 to CM0899	—	System reserved	
CM900	R	CPU function version	
CM901 to CM906	—	System reserved	
CM907	R	Project version	
CM908 to CM999	—	System reserved	
CM1000 to CM1001	R	Scan time measurements (1 μs) (32-bit units)	
CM1002 to CM1003	R	END process time measurements (1 μs units) (32-bit)	
CM1004 to CM1005	R	Minimum scan time (1 μs units) (32-bit)	
CM1006 to CM1007	R	Maximum scan time (1 μs units) (32-bit)	
CM1008 to CM1009		Set value when running in fixed scan time (in 1 μs units) (32-bit)	
CM1010 to CM1011		END process time Set value (in 1 μs units) (32-bit)	
CM1012 to CM1013	R	Scan time when setting is exceeded (in 1 μs units) (32-bit)	
CM1014 to CM1015	R	END process time exceed value (in 1 μs units) (32-bit)	
CM1016 to CM1029	—	System reserved	
CM1030 to CM1031	R	32-bit 1 ms Free Run counter	

CM	Attribute R: Read only Blank: Read/write	Function	Hold Yes: Hold
CM1032 to CM1033	R	32-bit 10 μs Free Run counter	
CM1034 to CM1035	R	32-bit 1 μs Free Run counter	
CM1036 to CM1037	R	32-bit 0.1 μs Free Run counter	
CM1038 to CM1499	–	System reserved	
CM1500 to CM1509	R	Log/Trace (ID0 to ID9) File save counter	
CM1510 to CM1629	–	System reserved	
CM1630 to CM1639	R	Minimum free space in log/ trace ring buffer (ID0 to ID9) (in KByte unit)	
CM1640 to CM1649	–	System reserved	
CM1650	R	Number of sensor setting instruction executions	
CM1651 to CM1657	–	System reserved	
CM1658	R	Number of consecutive project password authentication failures	Yes
CM1659	R	Total number of project password authentication failures	Yes
CM1660 to CM1669	R	Log/Trace ring buffer free space (ID0 to ID9) (KByte unit)	
CM1670 to CM1679	–	System reserved	
CM1680 to CM1689	R	Log/Trace File save counter (ID0 to ID9) (Memory card use)	Yes
CM1690 to CM1699	–	System reserved	
CM1700 to CM1701		Digital trimmer 0 upper limit (32-bit)	
CM1702 to CM1703		Digital trimmer 1 upper limit (32-bit)	
CM1704 to CM1705		Digital trimmer 2 upper limit (32-bit)	
CM1706 to CM1707		Digital trimmer 3 upper limit (32-bit)	
CM1708 to CM1709		Digital trimmer 4 upper limit (32-bit)	
CM1710 to CM1711		Digital trimmer 5 upper limit (32-bit)	
CM1712 to CM1713		Digital trimmer 6 upper limit (32-bit)	
CM1714 to CM1715		Digital trimmer 7 upper limit (32-bit)	
CM1716 to CM1719	–	System reserved	
CM1720	R	AW user message display number	
CM1721 to CM1735	–	System reserved	
CM1736		AW Initial display language setting	
CM1737		AW Initial display comment number setting	
CM1738		AW Operation prohibition setting	
CM1739	–	System reserved	
CM1740 to CM1772	R	Ladder project name (up to 32 characters + end code)	
CM1773 to CM1779	–	System reserved	
CM1780		AW Initial screen setting, Enable	
CM1781		AW Initial screen setting, Device type	
CM1782		AW Initial screen setting, Screen ID	
CM1783		AW Initial screen setting, Screen ID	
CM1784		AW Initial screen setting, Device No.	
CM1785		AW Initial screen setting, Device No.	
CM1786		AW Initial screen setting, Display format	
CM1787		AW Initial screen setting, Device name	
CM1788		AW Initial screen setting, Key lock state	
CM1789	–	System reserved	
CM1790		Auto load, Folder No. (request)	
CM1791	R	Auto load, Completion code	
CM1792	R	Auto load, Folder No. (completion)	
CM1793	–	System reserved	
CM1794		Run load, Folder No. (request)	
CM1795	R	Run load, Completion code	
CM1796	R	Run load, Folder No. (completion)	
CM1797 to CM1868	–	System reserved	
CM1869	R	Number of connected expansion units	
CM1870		Recipe load, Drive No.	Yes
CM1871		Recipe load, Folder No.	Yes
CM1872	—	System reserved	
CM1873	R	Recipe load, Completion code	Yes
CM1874	—	System reserved	
CM1875		Recipe save, Drive No.	Yes
CM1876		Recipe save, Reference folder No.	Yes
CM1877		Recipe save, Output folder No.	Yes
CM1878	R	Recipe save, Completion code	Yes
CM1879	—	System reserved	
CM1880		Project load drive No. (request)	
CM1881		Project load folder No. (request)	
CM1882	R	Project load drive No. (completion)	
CM1883	R	Project load folder No. (completion)	
CM1884	R	Project load completion code	

CM	Attribute R: Read only Blank: Read/write	Function		Hold Yes: Hold
CM1875 to CM1877	—	System reserved		
CM1888		Project save drive No. (request)		
CM1889		Project save folder No. (request)		
CM1890	R	Project save drive No. (completion)		
CM1891	R	Project save folder No. (completion)		
CM1892	R	Project save completion code		
CM1893 to CM1999	—	System reserved		
CM2000 to CM2001	R	Operation record function (ID0)	Device record 0, Average sampling period	
CM2002 to CM2003	R		Device record 0, Average number of Device changes	
CM2004 to CM2005	R		Device record 0, Maximum collection period	
CM2006 to CM2009	—		System reserved	
CM2010 to CM2039		Operation record function (ID1 to ID3)		
CM2040 to CM2389	—	System reserved		
CM2390	R	Error code for Storage device instruction		
CM2391		Power OFF during storage access (Another value other than 0 if occurred. Change it to 0 from the program.)		
CM2392		Memory card retry occurrences (still hold in case of power off)		
CM2393		Number of memory card verification errors (still hold in case of power off)		
CM2394		Memory card timeout occurrences (still hold in case of power off)		
CM2395		Reset delay time when memory card error occurred		
CM2396 to CM2399	—	System reserved		
CM2400 to CM2401	R	Inter-unit sync refresh start time (0.1μs)		
CM2402 to CM2403	R	Inter-unit sync refresh end time (0.1μs)		
CM2404 to CM2405		Maximum inter-unit sync refresh start time (0.1μs) (To reset, change it to 0 from the program.)		Yes
CM2406 to CM2407		Inter-unit sync refresh finishing time maximum (0.1μs) (To reset, change it to 0 from the program.)		Yes
CM2408 to CM2409	R	Inter-unit sync refresh operation time (0.1μs)		
CM2410 to CM5099	—	System reserved		
CM5100	R	Latest severe error details	Year and month (Decimal values. The last two digits show Month, and previous digits from hundred show Year. Example: 1210 → October 2012)	Yes
CM5101	R		Day	Yes
CM5102	R		Hour	Yes
CM5103	R		Minute	Yes
CM5104	R		Second	Yes
CM5105	R		Error NO.	Yes
CM5106	R		Valid data count	Yes
CM5107	R		Detailed information 01	Yes
CM5108	R		Detailed information 02	Yes
:	R			Yes
CM5126	R		Detailed information 20	Yes
CM5127 to CM5149	—	System reserved		
CM5150	R	Latest mild error details	Year and month (Decimal values. The last two digits show Month, and previous digits from hundred show Year. Example: 1210 → October 2012)	Yes
CM5151	R		Day	Yes
CM5152	R		Hour	Yes
CM5153	R		Minute	Yes
CM5154	R		Second	Yes
CM5155	R		Error number	Yes
CM5156	R		Valid data count	Yes
CM5157	R		Detailed information 01	Yes
CM5158	R		Detailed information 02	Yes
:	R			Yes
CM5176	R		Detailed information 20	Yes
CM5177 to CM5871	—	System reserved		
CM5872 to CM5873	R	Used amount of CPU memory (32-bit, in Byte unit)		Yes
CM5874 to CM5999	—	System reserved		

CM	Attribute R: Read only Blank: Read/write	Function		Hold Yes: Hold
CM6000		Operation record function(ID0)	Operation record 0, Save completion code	
CM6001	–		System reserved	
CM6002			Operation record 0, Save counter	
CM6003	–		System reserved	
CM6004	R		Operation record 0, Save progress details 1	
CM6005	R		Operation record 0, Save progress details 2	
CM6006	R		Operation record 0, Save progress	
CM6007 to CM6009	–		System reserved	
CM6010 to CM6171	R		Operation record 0, Save path	
CM6172 to CM6199	–		System reserved	
CM6200 to CM6799		Operation record function (ID1 to ID3)		
CM6800 to CM7599	–	System reserved		

■Error details

- In case of error No. 128 (calculation error)

Error No. n details	Description
Error No.1 details (M0712/CM5157)	Details of the error 0 (Unused) 1 Incorrect Indirect designation 2 The indirectly specified object is T/C 3 Incorrect simple Indirect designation 4 Outside of operand range 5 Floating-point real number overflow 6 Incorrect device specification 7 The ROOT instruction operand is negative 8 No bits turned ON by the ENCO instruction 9 Floating-point real number format error 10 Incorrect trigonometric function instruction operand 11 Data cannot be converted 12 Incorrect table specification 13 Incorrect time comparison instruction operand 14 Incorrect cam switch operand 15 Incorrect frequency counter instruction operand 16 Divide by zero 17 Incorrect log ID 18 Direct process error 19 Unit that cannot be processed directly 20 Jump destination subroutine does not exist 21 Instruction cannot be executed when writing in RUN mode 22 Expansion unit-dedicated instruction error (unit number) 23 System reserved 24 PIDAT, Setting outside of range 25 PIDAT, Sampling period outside of range 26 PIDAT, Comparison constant outside of range 27 PIDAT, Integral constant outside of range 28 PIDAT, Differential constant outside of range 29 PIDAT, Proportional constant 2-degree of freedom parameter outside of range 30 PIDAT Integral constant 2-degree of freedom parameter outside of range 31 PIDAT Differential constant 2-degree of freedom parameter outside of range 32 PIDAT, Operation quantity upper/lower limit outside of range 33 PIDAT, Operation quantity change width limit outside of range 34 PIDAT, Measurement value change width outside of range 35 PIDAT, AT state switch error 36 PIDAT, Operation control flag outside of range 37 Calculation period exceeded 38 PIDAT, AT adjustment parameter outside of range 39 PIDAT, AT hysteresis outside of range 40 PIDAT, AT timeout time outside of range 41 PIDAT, AT calculation control mode setting outside of range 42 PIDAT, AT timeout 43 PIDAT, AT comparison constant exceeded 44 PIDAT, AT integral constant exceeded 45 PIDAT, AT differential constant exceeded 46 Root of negative number cannot be calculated 47 Incorrect ABS instruction operand 48 Calculation overflow 49 UR, UM, or UV unit not connected 50 Unit device (R) not assigned 51 Unit device (DM) not assigned 52 Unit categories disagree 53 Incorrect operation recorder ID 54 Stack overflow
Error No.2 details (CM5158)	Fixed to 00H
Error No.3 details (CM5159)	Upper bit of the step number at which the error occurred
Error No.4 details (CM5160)	Lower bit of the step number at which the error occurred
Details of error No.5 to 20 (CM5161 to CM5176)	The Module name is saved according to the project language settings (any parts other than the Module name are "00H")

* When CR2012 is ON, CM0712 stores the value immediately, and CM5150 to CM5176 store the value after several scans.

- In case of No. 50, 53, 55, 56, 58, 59, 63 (unit number errors)

Error No. n details	Description
Error No.1 details (CM5107)	Unit number (the unit number will not be stored if it cannot be determined according to the error occurrence conditions)
Details of error No.2 to 20 (CM5108 to CM5126)	Fixed to 00H

- In case of No. 54 (unit count error)

Error No. n details	Description
Error No.1 details (CM5107)	The number of units specified in the unit settings
Error No.2 details (CM5108)	The number of units that are actually connected
Details of error No.3 to 20 (CM5109 to CM5126)	Fixed to 00H

- No. 127 (automatic loading failure error)

Error No. n details	Description
Error No.1 details (CM5157)	Details of the error
Details of error No.2 to 20 (CM5158 to CM5176)	Fixed to 00H

- No. 129 (unit error))

Expansion Info No. n	Description
Expansion info No.1 (CM5157)	Unit number
Expansion info No.2 (CM5158)	Error number of each unit
Expansion info No.3 to 20 (CM5159 to CM5176)	Fixed to 00H

- Error numbers other than those listed above

Expansion Info No. n	Description
Expansion info No.1 to 20	Fixed to 00H

- *1 Valid only for CPU units that support Bluetooth communication.

ASCII Code Table

ASCII code table is shown below.

Lower 4-bit \ Upper 4-bit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p				ー	タ	ミ		
1	SOH	DC1	!	1	A	Q	a	q			。	ア	チ	ム		
2	STX	DC2	”	2	B	R	b	r			「	イ	ツ	メ		
3	ETX	DC3	#	3	C	S	c	s			」	ウ	テ	モ		
4	EOT	DC4	$	4	D	T	d	t			、	エ	ト	ヤ		
5	ENQ	NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	BEL	ETB	’	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	VT	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	FF	→	,	<	L	\	l	|			ャ	シ	フ	ワ		
D	CR	←	—	=	M	]	m	}			ュ	ス	ヘ	ン		
E	SO	↑	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	SI	↓	／	?	O	_	o	DEL			ッ	ソ	マ	゜		

List of Characters That Cannot Be Used

This section describes the characters that cannot be used in project names, Module names, Macro names, labels, and Device comments.

● Characters that cannot be used in project names

Characters that cannot be used	! ″ # $ % & ‘ * , . / : ; < = > ? @ ¥ ^ ` | ~ (space)

● Characters that cannot be used in program names (Module name, Function block name, Macro name)

Characters that cannot be used	! ″ # $ % & ’ ( ) * , . / : ; < = > ? @ [¥] ^ ` { | } } ~ (space)
Characters that cannot be used at the start	+ − 0 1 2 3 4 5 6 7 8 9 _
Characters or strings that cannot be used at the end	_ _0 _1 _2 _3 _4 _5 _6 _7 _8 _9
Character strings that cannot be used	ｸﾞﾛｰﾊﾞﾙ グローバル .

● Characters that cannot be used in Variable names

Characters that cannot be used	! ″ # $ % & ’ ( ) * + , − . / : ; < = > ? @ [¥] ^ ` { | } ~ (space)
Characters that cannot be used at the start*1	0 1 2 3 4 5 6 7 8 9 _ U _
Character strings that cannot be used	ｸﾞﾛｰﾊﾞﾙ グローバル
	(The same names as instructions)
	(Device names)
	• One alphabet character + R or M + number • One alphabet character + number • X +(A to F) • Y +(A to F)
	(Same name as ST Language Function) (Same name as KV Script Function) • Keywords specified in KV script !!, #PRAGMA,', (,), *, * =, +, + =,-,-=, .B, .D, .DF, .F, .L, .S, .T, .U , /, / =, <, <=, <>, =,>,> =, AND, AS, B, BREAK, C, CASE, CM, CR, CTC, CTH, DI M, DM, DO, DOUBLE, DR, ELSE, ELSE IF, EM, END, END IF, END SELECT, END TYPE, END WHILE, FALSE, FM, FOR, IF, IS, LR, MC, MCR, MOD, MR, NEXT, NOT, OFF, ON, OR, P, R, REM, SELECT, SELECT CASE, STEP, STRWORK, T, THEN, TM, TO, TRUE, TYPE, UM, UNTIL, UR, UV, V, W, WHILE, XOR, Z, ZF, [, \\, \\ A, \\ B, \\ F, \\ N, \\ R, \\ T, \\ V,]
	• Keywords specified in IEC61131-3 TRUE, FALSE TIME, DATE,TIME_OF_DAY, TOD, DATE_AND_TIME, DT BOOL/UINT/INT/UDINT/DINT/REAL/LREAL/STRING/Array/Structure DWORD, LWORD ANY, ANY_DERIVED, ANY_ELEMENTARY, ANY_MAGNITUDE, ANY_NUM, ANY_REAL, ANY_INT, ANY_BIT, ANY_STRING, ANY_DATE ACTION, END_ACTION, ARRAY, OF, AT, CASE, END_CASE, CONFIGURATION, END_CONFIGURATION, CONSTANT, EN, ENO, EXIT, F_EDGE, FOR, TO, BY, DO, END_FOR, FUNCTION, END_FUNCTION, FUNCTION_BLOCK, END_FUNCTION_BLOCK, IF, THEN, ELSIF, ELSE, END_IF,

Character strings that cannot be used	INITIAL_STEP, END_STEP, NOT, MOD, AND, XOR, OR, PROGRAM, WITH, END_PROGRAM, R_EDGE, READ_ONLY, READ_WRITE, REPEAT, UNTIL, END_REPEAT, RESOURCE, END_RESOURCE, RETAIN, NON_RETAIN, RETURN, STEP, STRUCT, END_STRUCT, TASK, TRANSITION, FROM, END_TRANSITION, TYPE, END_TYPE, VAR,VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT, VAR_TEMP, VAR_EXTERNAL, VAR_ACCESS, VAR_CONFIG,VAR_GLOBAL, END_VAR, WHILE, DO, END_WHILE

● Characters that cannot be used in Structure member names

Characters that cannot be used	( ) * / + − < > = & ! ¥ ″ # $ % % & | ' ~ ` @ [ ] { : ; , , . ? ^ (1-byte space) (Full-width space)
Characters that cannot be used at the start	0 1 2 3 4 5 6 7 8 9
Characters or strings that cannot be used at the end	_

Character strings that cannot be used	ｸﾞﾛｰﾊﾞﾙ グローバル Global 全局 全域 글로벌
	• Character strings that cannot be distinguished from numeric values
	  • K
	  • K + value
	  • K + value + suffix (U/S/D/L/F/DF)
	  • K + value + E
	  • K + value + E + value
	  • K + value + E + value + suffix (F/DF)
	  • H + hexadecimal value (decimal value or ABCDEF string)
	• Key words specified in ST/Script
	(For the contents of keyword, refer to "Characters that cannot be used in Variable/Label names")
	• Keywords specified in IEC61131-3
	(For the contents of keyword, refer to "Characters that cannot be used in Variable/Label names")

● Characters that cannot be used in Device comments

Characters that cannot be used	,

Unusable Function List

This section describes the Functions that cannot be used in the Initialization Module, Function block, Macro, and Interrupt program.

● Initialization Module/Interrupt program

Function type	Function	
Contact	LDP/LDF	Load pulse/Load pulse fall
	LDPB/LDFB	Load pulse bar/Load pulse fall bar
Timer/Counter	TMR/TMH/TMS/TMU	Timer/High-speed timer/High-speed 1ms timer/High-speed 10μs timer
	C	Counter
File register	FRSTM/FRLDM	File register batch save/File register batch read
Storage	MWRIT/MREAD/ MFREE/ MMKDIR	Storage write/Storage read Storage free space get/Storage folder create
	MRMDIR/MDEL	Storage folder delete/Storage file delete
	MPRINT/MREADL/MCOPY/ MMOV/MREN/MFREEK/ MSTAT	Storage text write/One line storage read/ Storage file copy/Storage file move/ Storage file rename/Storage free space get/ Storage state get
Access Window	AWSHOW/AWHIDE	User message display/User message hide

● FEND (Function block end) Function is written in Function block. It cannot be used with other types of programs such as Function and Module.

● Function

Function type	Function	
Function block	FBSTRT	Function block

● Macro

Function type	Function	
Function block	FBSTRT	Function block/Function

List of Functions supported as Standard Functions

The following Functions can be used as Standard Functions for PLC OPEN.

Function type	Function
Logic operate	ABS
Data Shift	SHR, SHL, ROR, ROL
Data Control	LIMIT
Data Convert	*_TO_**,TO_**,*_TRUNC_**,TRUNC_**, *_BCD_TO_**,BCD_TO_**, *_TO_BCD_** ,TO_BCD_**
Floating point real number	EXP, EXPT, LOG, LN, SQRT, SIN, COS, TAN, ASIN, ACOS, ATAN
Character Process	LEN, RIGHT, LEFT, MID, REPLACE, INSERT, DELETE, FIND, CONCAT
Data Process	MIN, MUX, SEL, MAX

Index

A

About Assignment Statement
- Bit assignment 1-4
- Numerical value assignment 1-4
- Text string assignment 1-4

About Control Statement
- Conditional transfer control statements 1-5

About processing time 4-16
About timer actions in a loop control statement 4-17
Actual nesting actions 4-16
Area script is used in the following cases 2-11
Arguments used in the Function block only are as shown below. 3-3
Array 2-16
ASCII Code Table A-9

B

Bit assignment 1-4
Bit assignment statement 4-2
Bit Device
- B 3-9
- LR(Latch relay) 3-10
- MR 3-10
- R 3-9
- Status of Hold Control Bit Date 4-18

Bit status 3-7
Boolean value 3-7

C

Call function list 2-30
Calling Function Blocks 2-27
Comparison operator 4-4
Conditional transfer control statements 1-5
Constant 3-22
- $ HEX constant 3-23
- Double precision floating point type constant 3-24
- Fixed text string (text string type constant) 3-24
- Single precision floating point type constant 3-23
- (#) DEC constant 3-22

Control memory CM 3-19
Control relay CR 3-13
Control statements include the following 2 types 4-6
Copy area script 2-14
Counter (current value) C 3-18
Counter (contact) C 3-12
How to process setting/current value of the counter 3-13
Creating ST 2-2
Customization 2-34

D

Data memory/Extended data memory DM/EM/FM 3-13
Data Type (suffix) 3-31
Delete area script 2-12, 2-14
Delete box script 2-4, 2-7
Device list (for KV Nano Series) Macro argument device 3-4
Device List (for KV-8000)
Bit devices used on KV-8000 3-2
- Device list (for KV-8000)
- Constant 3-3
- KKV-8000 series word device 3-2
- Local device list 3-3, 3-4

Device list (for KV-8000)Function block argument
- Function block argument 3-3

Device list (for KV-8000)Internal register Internal register 3-3
Device number list A-2
Devices 3-2
Direct input 2-3
Double precision floating point type constant 3-24

E

Error Message
- When conversion error occurs 2-24

Error display in output window 2-25
Error Message
- Error display in output windowError 7 2-25

F

Features of Box ST 2-4
File register ZF/FM 3-15
Function
- Script Function List 2-30

Function Version(KV-8000) 0-2

G

Global variables and Local variables. 2-15

H

HELP Function 2-32
How to Input Scripts
- Direct input 2-3
- Window input 2-3

How to Input Scripts Input(How to Input Scripts) 2-3
How to process setting/current value of the timer 3-12
How to Use An Assignment Statement 4-3
- Bit assignment statement 4-2
- Numerical value assignment statement 4-2
- String assignment statement 4-2
- When it can not be assigned to left side 4-3
- When the type on right side is different from that on left side 4-3

How to use output window 2-25
How to write ST 2-2

I

IF ～ ELSE 4-8
IF ～ ELSIF 4-9
Index constant
- Modify via index constant 3-26

Index modification
- Modify by index register 3-26

Index constant
- Modify the device with suffix 3-27
- Modify via the device and expression 3-27
- Invalid index modification 3-27

Index modification 3-25, 3-27
Index Modification and Indirect Specifying 3-25
Index register Z 3-18
Type of indirect specifying device 3-29
Indirect specifying programming 3-27
Indirect specifying principle 3-27
Application method of indirect specifying 3-28
Example for indirect specifying 3-29
What is indirect specifying(*) 3-27
I/O relay R 3-7
Input(How to Input Scripts)
- Window input 2-3

Input(How to Input Scripts)
Direct input 2-3
Internal auxiliary relay R/MR/LR 3-9

L

LDP 5-3
Link register W 3-16
Link relay B 3-8
List of Characters That Cannot Be Used
Characters that cannot be used in structure member names ... A-12
List of Characters That Cannot be Used
Character that cannot be used in device comments A-12
Characters that cannot be used in program names (module names, function block names, macro names) A-10
Characters that cannot be used in project names A-10
List of operators 1-5
Loop control statement 4-7
Loop control statement and processing time 4-16

M

Making Box Script 2-4, 2-9, 2-11
MOD 4-4

O

Omitting suffix 1-7
Data range processed by operation formula 3-36
Operation flag 1-8
Operation statement +=,-=,*=,/= 4-3
Operator 4-4
Operator List 4-4
Arithmetic operator 4-4
Comparison operator 4-4
Logic operator 4-4
String Connection Operator 4-4
The priority between multiplication, division, addition, and subtraction 4-5
Operator type 4-4

P

Precautions about use of timer function in Box Script and IF statement . 4-18
Precautions on Script Programming 4-18
Precautions on using Assignment Statement 4-3
Precautions on Using Box Script
When using timer function 2-9
Precautions on Using Control Statement 4-16
Loop control statement and processing time 4-16
Precautions when programming script 2-34
Precautions when using a timer function 3-12
Precautions when using control statement
About timer actions in a loop control statement 4-17
Precautions when using the timer 4-18
Precautions when using the timer function
Specify timer function using Box Script 4-19
Use timer function in IF statement of area script 4-18

R

Reedit area script 2-13
Reedit Box Script 2-7
Register monitor 2-28
Relay 3-9
I/O relay R 3-7
Internal auxiliary relay R/MR/LR 3-9
Reserved word 2-31

S

Script Function List 2-30
SELECT CASE 4-11
Simple indirect specifying #TM 3-30
Single precision floating point type constant 3-23
Specify timer function using Box Script 4-19
ST Types 1-2
String assignment statement 4-2
String Connection Operator 4-4
Suffix
About .U and .S 3-32
About .D and .L 3-32
About .DF 3-33
About □ .B 3-34
About .T 3-34
About .F 3-33

T

Temporary data memory TM 3-17
Text string assignment 1-4
Text string type constant 3-21, 3-22
Fixed text string (text string type constant) 3-24
Timer (current value) T 3-18
Timer (contact) T 3-11
Timer error 3-11
Type(Data Type (suffix)) 3-31
Type(Data Type(suffix))
Value change when exceeding max./min. value 3-35
Type conversion leads to data loss 2-24

U

Use timer function in IF statement of area script 4-18
Using Box ST in the following cases 2-4

V

Variable programming 2-15
Variable setting 2-20
Variable suffix 1-7
Variables 3-21, 3-22

W

Watch window 2-29
When changing, if Current value exceeds Set value, the Current value is changed to the value identical to the Set value. 3-13
When conversion error occurs 2-24
When looking up notes on indirect specifying and index modification 1-8
When processing as the bit device at any bit. 3-20
When processing as the bit device at the lowe rest bit 3-20
When using differential execution type functions 2-9
When using timer function 2-9
WHILE ～ 4-14, 4-15
Window input 2-3
Work Devices 2-35

Functions Index

Symbol

*_BCD_TO_** 7-43
*_TO_** 7-39
*_TO_BCD_** 7-42
*_TRUNC_** 7-40

A

ABS 7-13
ACNT 6-23, 6-24
ACOS 7-55
ADRADD 6-11, 6-12
ADRDEC 6-10
ADRINC 6-9
ADRSET 6-9
AJST 8-24
ANDA 7-9
APR 7-24
ASC 7-59
ASIN 7-55
ASLA 7-16
ASRA 7-15
ATAN 7-56
ATAN2 7-57
AVG 8-9
AWHIDE 8-46
AWSHOW 8-46

B

BANDC 7-22
BCD_TO_** 7-43
BCMP 8-13
BCMPI 8-14
BCNT 8-3
BLD 5-5
BLDB 5-7
BMOV 7-2
BOUB 5-10
BOUT 5-10
BRES 5-12
BSET 5-11
BSL 7-21
BSR 7-20
BSUM 8-10
BSWAP 7-35
BYBMOV 7-4
BYLMOV 7-5

C

CALL 6-2
CHR 7-85
CNT 5-16
COM 7-12
CONCAT 7-87
COS 7-53
CPMGET 7-38
CPMSET 7-37
CPSASC 7-86
CRC 8-11

D

DASC 7-60
DCNT 8-3
DEC 7-7
DECO 7-36
DEG 7-51
DELETE 7-76
DFLOAT 7-44
DI 8-25
DIC 8-26
DISB 7-34
DISF 7-46
DISN 7-32
DISS 7-84
DMX 7-30
DSER 8-5

E

ECALL 6-2
EI 8-25
ENCO 7-37
ENRA 7-11
EORA 7-10
EXP 7-48

F

FASC 7-65
FBSTRT 6-4
FDEL 8-21
FEND 6-5
FIFOR 8-17
FIFOW 8-17
FIND 7-83
FINS 8-20
FLOAT 7-44
FMOV 7-2
FRLDM 6-14
FRSET 6-13
FRSTM 6-13
FWRIT 8-19

G

GRY 7-31

H

HASC 7-63

I

IEDGE 8-26
INC 7-6
INSERT 7-75
INSTR 7-82
INT 7-46
INTG 7-45

L

LDF 5-3
LDFB 5-5
LDP 5-3
LDPB 5-4
LEFT 7-70
LEN 7-67
LIFOR 8-19
LIFOW 8-18
LIMIT 7-22
LLFLT 7-27
LN 7-49

LOG 7-50
LOGD 8-29
LOGE 8-29
LTRIM 7-78

M

MAX 8-5
MAX_RANGE 8-6
MCOPY 8-42
MDEL 8-39
MDSTOP 6-6
MDSTRT 6-5
MFREE 8-34
MFREEK 8-36
MID 7-72
MIN 8-7
MIN_RANGE 8-7
MMKDIR 8-37
MMOV 8-43
MPRINT 8-40
MPX 7-30
MREAD 8-33
MREADL 8-41
MREN 8-44
MRMDIR 8-38
MSTAT 8-44
MUX 8-8
MWRIT 8-32

N

NEG 7-12

O

ORA 7-9

P

PIDAT 8-28
PMOV 7-3

R

RAD 7-50
RAMP 7-25
RASC 7-59
RCPSASC 7-86
RDASC 7-62
REPLACE 7-73
RES 5-8
RFASC 7-66
RFSCI 8-53
RFSCO 8-53
RFSFRC 8-48
RFSX 5-18
RFSY 5-18
RGRY 7-32
RHASC 7-64
RIGHT 7-69
RLA 7-17
RND 8-14
ROL 7-18
ROOT 7-7
RRA 7-16
RRNCA 7-17
RSEC 8-23
RTRIM 7-78

S

SDEL 7-75
SEC 8-22
SEG 8-2
SEL 8-8
SER 8-4
SET 5-8
SFIND 7-81
SFINDN 7-82
SIN 7-52
SINS 7-74
SLA 7-14
SLEFT 7-69
SMID 7-71
SORT 8-15
SORTN 8-16
SPLIT 7-80
SPRD 8-49
SPWR 8-50
SQRT 7-8
SRA 7-14
SRGHT 7-68
SRPLC 7-72
SSVC 8-51
STR 7-61
STRIM 7-79
SWAP 7-35

T

TAN 7-54
TBCD 7-29
TBIN 7-29
TMH 5-14
TMR 5-13
TMS 5-15
TMU 5-15
TO_** 7-40
TO_BCD_** 7-42
TPOUT 7-26
TRGD 8-30
TRGR 8-30
TRIM 7-77
TRUNC_** 7-41

U

UFILL 6-22
UFRSM 6-18
UFSUS 6-17
UMALLOC 6-20
UNIB 7-34
UNIF 7-47
UNIN 7-33
UNIS 7-85
UPSTOP 6-16
UPSTRT 6-15
UREAD 6-21
UWRIT 6-21

V

VAL 7-67

W

WSIZE 6-25
WSL 7-19

WSR 7-19
WSUM 8-10
WTIME 8-22

Z

ZONE 7-23
ZPOP 6-8
ZPUSH 6-7
ZRES 8-12

Revision History

Printing Date	Version	Details of Revision
Apr 2021	First version	
Sep 2021	Revised 1st edition	
Feb 2022	2nd revision 1st edition	

WARRANTIES AND DISCLAIMERS

(1) KEYENCE warrants the Products to be free of defects in materials and workmanship for a period of one (1) year from the date of shipment. If any models or samples were shown to Buyer, such models or samples were used merely to illustrate the general type and quality of the Products and not to represent that the Products would necessarily conform to said models or samples. Any Products found to be defective must be shipped to KEYENCE with all shipping costs paid by Buyer or offered to KEYENCE for inspection and examination. Upon examination by KEYENCE, KEYENCE, at its sole option, will refund the purchase price of, or repair or replace at no charge any Products found to be defective. This warranty does not apply to any defects resulting from any action of Buyer, including but not limited to improper installation, improper interfacing, improper repair, unauthorized modification, misapplication and mishandling, such as exposure to excessive current, heat, coldness, moisture, vibration or outdoors air. Components which wear are not warranted.

(2) KEYENCE is pleased to offer suggestions on the use of its various Products. They are only suggestions, and it is Buyer's responsibility to ascertain the fitness of the Products for Buyer's intended use. KEYENCE will not be responsible for any damages that may result from the use of the Products.

(3) The Products and any samples ("Products/Samples") supplied to Buyer are not to be used internally in humans, for human transportation, as safety devices or fail-safe systems, unless their written specifications state otherwise. Should any Products/Samples be used in such a manner or misused in any way, KEYENCE assumes no responsibility, and additionally Buyer will indemnify KEYENCE and hold KEYENCE harmless from any liability or damage whatsoever arising out of any misuse of the Products/Samples.

(4) **OTHER THAN AS STATED HEREIN, THE PRODUCTS/SAMPLES ARE PROVIDED WITH NO OTHER WARRANTIES WHATSOEVER. ALL EXPRESS, IMPLIED, AND STATUTORY WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS, ARE EXPRESSLY DISCLAIMED. IN NO EVENT SHALL KEYENCE AND ITS AFFILIATED ENTITIES BE LIABLE TO ANY PERSON OR ENTITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, PUNITIVE, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, ANY DAMAGES RESULTING FROM LOSS OF USE, BUSINESS INTERRUPTION, LOSS OF INFORMATION, LOSS OR INACCURACY OF DATA, LOSS OF PROFITS, LOSS OF SAVINGS, THE COST OF PROCUREMENT OF SUBSTITUTED GOODS, SERVICES OR TECHNOLOGIES, OR FOR ANY MATTER ARISING OUT OF OR IN CONNECTION WITH THE USE OR INABILITY TO USE THE PRODUCTS, EVEN IF KEYENCE OR ONE OF ITS AFFILIATED ENTITIES WAS ADVISED OF A POSSIBLE THIRD PARTY'S CLAIM FOR DAMAGES OR ANY OTHER CLAIM AGAINST BUYER.** In some jurisdictions, some of the foregoing warranty disclaimers or damage limitations may not apply.

BUYER'S TRANSFER OBLIGATIONS:

If the Products/Samples purchased by Buyer are to be resold or delivered to a third party, Buyer must provide such third party with a copy of this document, all specifications, manuals, catalogs, leaflets and written information provided to Buyer pertaining to the Products/Samples.

Specifications are subject to change without notice.

KEYENCE CORPORATION

1-3-14, Higashi-Nakajima, Higashi-Yodogawa-ku, Osaka, 533-8555, Japan PHONE: +81-6-6379-2211

www.keyence.com/glb

AUSTRIA
Phone: +43 (0)2236 378266 0

BELGIUM
Phone: +32 (0)15 281 222

BRAZIL
Phone: +55-11-3045-4011

CANADA
Phone: +1-905-366-7655

CHINA
Phone: +86-21-3357-1001

CZECH REPUBLIC
Phone: +420 220 184 700

FRANCE
Phone: +33 1 56 37 78 00

GERMANY
Phone: +49-6102-3689-0

HONG KONG
Phone: +852-3104-1010

HUNGARY
Phone: +36 1 802 7360

INDIA
Phone: +91-44-4963-0900

INDONESIA
Phone: +62-21-2966-0120

ITALY
Phone: +39-02-6688220

KOREA
Phone: +82-31-789-4300

MALAYSIA
Phone: +60-3-7883-2211

MEXICO
Phone: +52-55-8850-0100

NETHERLANDS
Phone: +31 (0)40 206 6100

PHILIPPINES
Phone: +63-(0)2-8981-5000

POLAND
Phone: +48 71 368 61 60

ROMANIA
Phone: +40 (0)269 232 808

SINGAPORE
Phone: +65-6392-1011

SLOVAKIA
Phone: +421 (0)2 5939 6461

SLOVENIA
Phone: +386 (0)1 4701 666

SWITZERLAND
Phone: +41 (0)43 455 77 30

TAIWAN
Phone: +886-2-2721-8080

THAILAND
Phone: +66-2-078-1090

UK & IRELAND
Phone: +44 (0)1908-696-900

USA
Phone: +1-201-930-0100

VIETNAM
Phone: +84-24-3772-5555

A4WW1-MAN-2121

Copyright (c) 2021 KEYENCE CORPORATION. All rights reserved. 084699GB 2022-1 C60GB Printed in Japan

C60GB-1